MODUL PRAKTIKUM JARINGAN KOMUNIKASI PJK-05
PROTOKOL PENGIRIMAN DATA *ASYNCHRONOUS* BERBASIS MIKROKONTROLER

A. TUJUAN

1. Mengenal dan memahami protokol komunikasi data *asynchronous* berbasis mikrokontroler.
2. Mengenal dan memahami prinsip kerja protokol komunikasi UART dan MODBUS pada mikrokontroler.

B. KOMPETENSI DASAR

1. Rangkaian Listrik
2. Dasar Informatika
3. Elektronika Analog
4. Pengolahan Data
5. Elektronika Digital
6. Jaringan Komunikasi

C. DASAR TEORI

1. UART (*Universal Asynchronous Receiver – Transmitter*)

UART adalah bagian perangkat keras komputer yang menerjemahkan bit-bit paralel data dan bit-bit serial. UART biasanya berupa sirkuit terintegrasi yang digunakan untuk komunikasi serial pada komputer atau *port serial* perangkat *peripheral*. Perangkat yang menggunakan protokol komunikasi UART dapat terhubung langsung pada perangkat lain yang juga mendukung protokol UART secara *wired* dan juga secara *wireless* dengan menggunakan modul RF433, *bluetooth*, Wi-Fi, dll. Dalam pengiriman data, *baudrate* antara pengirim dan penerima harus sama karena paket data dikirim tiap bit mengandalkan *baudrate* tersebut. Inilah salah satu keuntungan model *asynchronous* dalam pengiriman data karena dengan hanya satu kabel transmisi maka data dapat dikirimkan. UART merupakan antarmuka yang digunakan untuk komunikasi serial, seperti pada RS-232, RS-422, RS-485 [1].

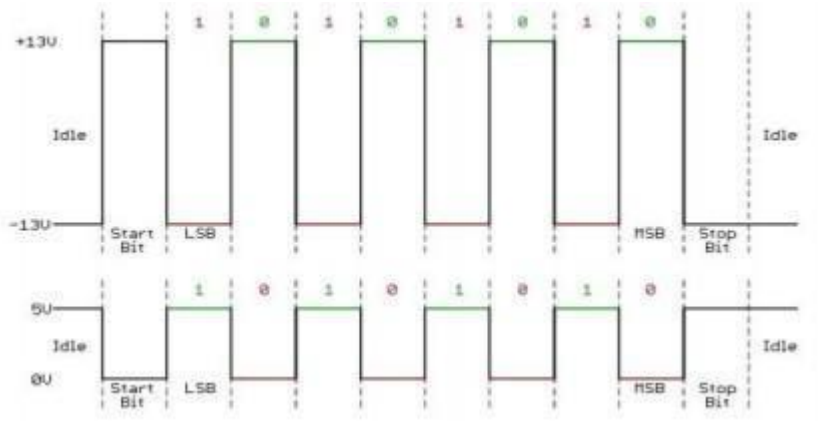
Komponen UART dapat dikonfigurasikan untuk *Full Duplex* (dua arah bersamaan), *Half Duplex* (dua arah bergantian), *Simplex* (hanya pengiriman atau penerimaan). Pada PSOC UART dapat dikonfigurasikan pada baudrate dari 110 sampai dengan 921600 bps dengan penyangga RX dan TX dari 4 sampai 65535 [2].

Kebanyakan mikrokontroler saat ini telah membuat UART yang bisa digunakan untuk menerima dan mengirim data secara serial, metode ini biasanya disebut sebagai TTL serial (*transistor - transistor logic*). Komunikasi serial pada TTL selalu menggunakan batasan tegangan 0V dan Vcc yang biasanya sebesar 5V atau 3,3 V. Logika HIGH (‘1’) direpresentasikan oleh Vcc, sedangkan logika LOW (‘0’) direpresentasikan oleh 0V [3].

UART dapat dengan mudah dikonfigurasikan dengan memilih *baudrate, parity, bit data*, dan angka pada *start bits*. Pada konfigurasi RS232 biasanya disebut dengan “8N1” yang merupakan singkatan dari delapan *data bit*, tanpa *parity*, dan memiliki 1 *stop bit*, hal ini merupakan konfigurasi default pada komponen UART [2].

Port serial pada komputer mengikuti standar telekomunikasi RS-232 (*Recommended Standard* 232). Sinyal dari RS-232 sama dengan sinyal serial mikrokontroler apabila sinyal tersebut ditransmisikan pada satu bit dalam satu waktu, dengan *baudrate* tertentu dan dengan atau tanpa *parity* maupun *stop bits*. Pada Standar RS-232 maka *logic* HIGH (‘1’) direpresentasikan oleh tegangan negatif (-3V sampai – 25 V) dan logic LOW direpresentasikan dengan tegangan positif (3V sampai 25V). Kebanyakan PCs (*personal communication system*) memiliki sinyal dari -13 V sampai 13 V [4].

Karena mempunyai rentang tegangan yang lebar maka sinyal RS232 membuat data yang dikirim tahan terhadap derau, gangguan, dan degradasi, sehingga dapat dikatakan bahwa sinyal RS-232 dapat melakukan komunikasi pada jarak yang jauh dibanding dengan serial TTL [5].



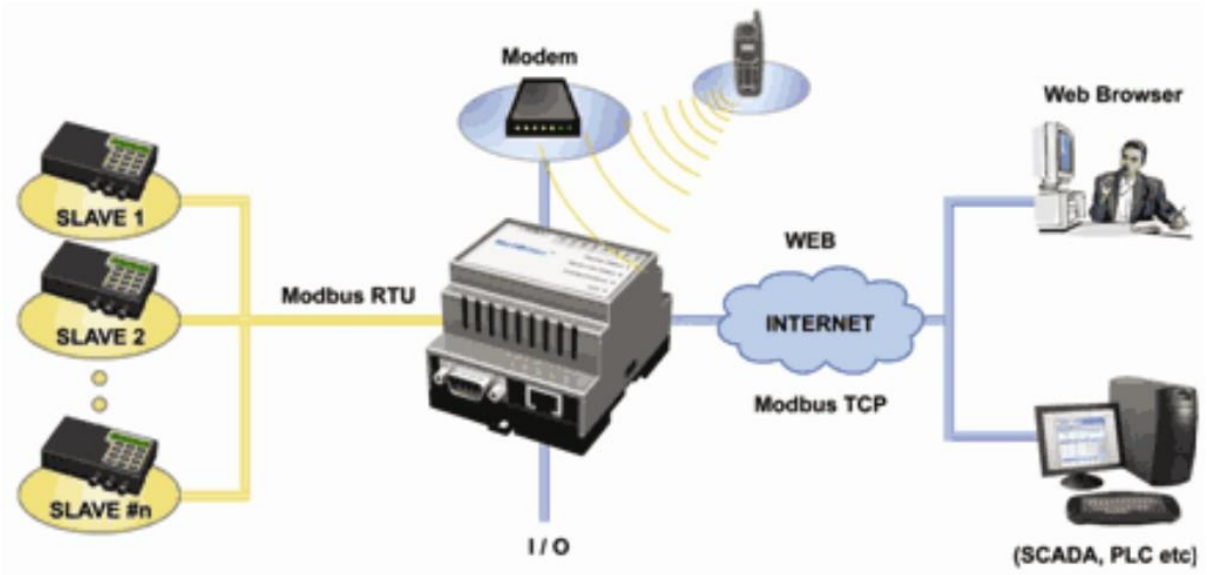
Gambar 5.1. Pengiriman sinyal TTL dan RS-232.

2. MODBUS

2.1. Pendahuluan

Modbus adalah protokol komunikasi serial yang dipublikasikan oleh Modicon pada tahun 1979 untuk diaplikasikan ke dalam programmable logic controller (PLC) [6]. Modbus terletak pada level 7 dari model OSI (OSI Layer). Modbus sudah menjadi standar protokol yang umum digunakan untuk menghubungkan peralatan elektronik industri. Terdapat beberapa alasan protokol ini banyak digunakan [7], yaitu:

1) Modbus dipublikasikan secara terbuka dan bebas royalti.
2) Mudah digunakan dan dipelihara.
3) Memindahkan data bit atau word tanpa banyak membatasi vendor.



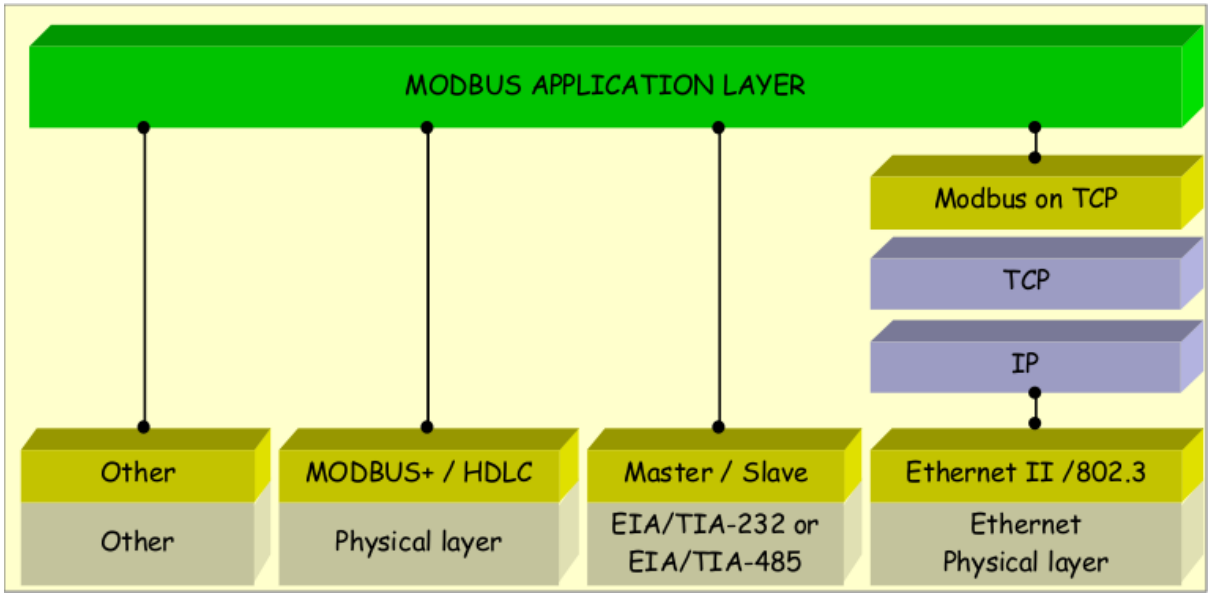
Gambar 5.2. Ilustrasi Protokol Komunikasi Modbus [12].

Modbus digunakan untuk komunikasi antar banyak perangkat dalam satu jaringan. Modbus banyak digunakan untuk menghubungkan komputer pemantau dengan *remote terminal unit* (RTU) pada *sistem supervisory control and data aquisition* (SCADA) [8].

2.2. Jenis – jenis Modbus

Terdapat beberapa jenis versi Modbus, antara lain:

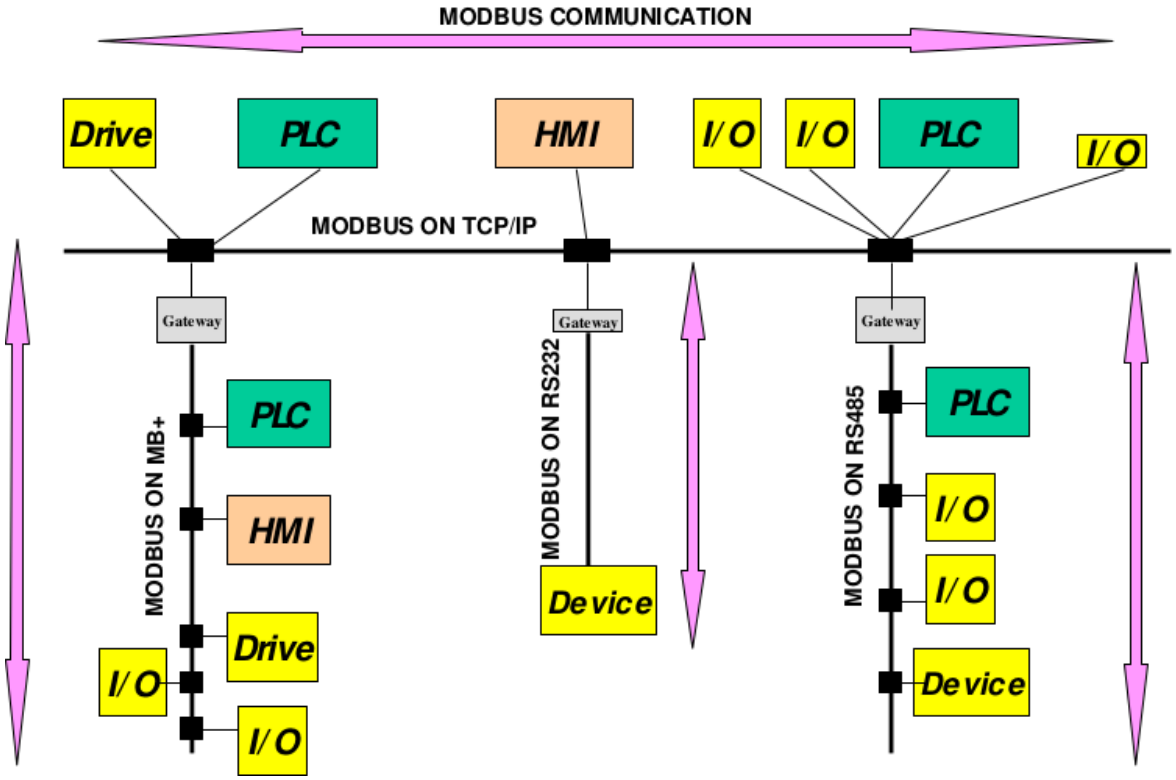
1) Modbus RTU: Varian Modbus yang ringkas dan digunakan pada komunikasi serial. Format RTU dilengkapi dengan mekanisme *cyclic redundancy error* (CRC) untuk memastikan keandalan data. Modbus RTU merupakan implementasi protokol Modbus yang paling umum digunakan. Setiap frame data dipisahkan dengan periode *idle* (*silent*) [9].
2) Modbus ASCII: Digunakan pada komunikasi serial dengan memanfaatkan karakter ASCII. Format ASCII menggunakan mekanisme *longitudinal redundancy check* (LRC). Setiap data frame data Modbus ASCII diawali dengan titik dua (":") dan baris baru yang mengikuti (CR/LF) [9].
3) Modbus TCP/IP atau Modbus TCP merupakan Modbus yang digunakan pada jaringan TCP/IP. Komunitas internet dapat mengakses Modbus di Port sistem 502 pada TCP/IP [9].



Gambar 5.3. Lapisan Penggunaan Modbus [9].

2.3. Arsitektur Jaringan Modbus

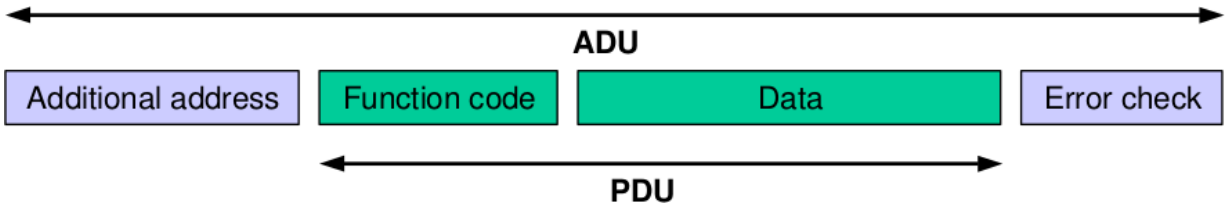
Setiap jenis perangkat, baik itu PLC, HMI, Control Panel, Driver, Motion Control, Perangkat I/O, dapat menggunakan protokol Modbus untuk memulai sebuah komunikasi jarak jauh. Komunikasi yang sama dapat dilakukan juga pada jalur serial sama seperti jaringan ethernet TCP/IP. Perangkat Gateway memungkinkan komunikasi antar beberapa tipe bus atau jaringan menggunakan protokol Modbus [9].



Gambar 5.4. Arsitektur Komunikasi Modbus [9].

2.4. Gambaran Protokol

Modbus adalah protokol *request*/*reply* dan menawarkan layanan yang ditentukan oleh kode fungsi. Kode fungsi Modbus merupakan unsur *request*/*reply* PDU Modbus. Protokol Modbus mendefinisikan sebuah protokol data unit (PDU) independen pada lapisan komunikasi. Pemetaan protkol Modbus pada bus yang spesifik atau jaringan dapat mengenalkan beberapa penambahan isian data pada sebuah Application Data Unit (ADU) [9].



Gambar 5.5. Gambaran umum protokol Modbus [9].

Application Data Unit (ADU) dari sebuah Modbus protokol dibangun oleh klien yang menginisiasi sebuah transaksi data. Fungsi tersebut mengindikasikan ke server jenis aksi yang akan dilakukan. Kemudian protokol Modbus membangun format yang telah diminta oleh klien [9].

Sebagian besar peralatan Modbus menggunakan port serial RS-485. Konsep dasar komunikasi Modbus terdiri *master* dan *slave*. Peralatan yang bertindak sebagai slave akan terus idle kecuali mendapat perintah dari master. Setiap Peralatan yang dihubungkan (*slave*) harus memiliki alamat unik. Sebuah perintah Modbus dilengkapi dengan alamat tujuan perintah tersebut. Hanya alamat tujuan yang akan memproses perintah, meskipun peralatan yang lain mungkin menerima perintah tersebut. Setiap perintah modbus memiliki informasi pemeriksaan kesalahan untuk memastikan data diterima tanpa kerusakan. Perintah dasar Modbus RTU dapat memerintahkan peralatan untuk mengubah nilai registernya, mengendalikan dan membaca port I/O, serta memerintahkan peralatan untuk mengirimkan kembali nilai yang ada pada registernya [10].

2.5. Cara Kerja Modbus

Modbus serial adalah komunikasi serial yang menggunakan protokol Modbus. Pada komunikasi serial, pengiriman data besarnya adalah 8 byte atau 8 frame data dalam 1x pengiriman [11].

[byte0] [byte1] [byte2] [byte3] [byte4] [byte5] [byte6] [byte7]

Besarnya nilai data dalam 1 framenya adalah 2^8 = 256 (0-255). maka nilai terbesarnya adalah 256. jika kita mengirimkan ASCII misalkan huruf “A” maka nilai yang digunakan sesuai ASCII table adalah 65. Pada Modbus frame-frame tersebut diisi dengan kesepakatan tertentu. Modbus *frame* untuk *request* dari *master* ke *slave* adalah sebagai berikut [11],

[address RTU] [Function Code] [Reg] [Reg] [Lenght][Lenght] [CRC1] [CRC2]

Keterangan:

Address RTU / Slave: *Address* dari setiap RTU, jika alamat RTU yang direquest masuk ke RTU sesuai, maka RTU tersebut akan meresponnya dengan memberikan pesan balasan. Byte pertama sebagai alamat *slave* terdiri dari 1 byte (0~255). Namun alamat *slave* yang dapat diakses hanya 1 hingga 247. Alamat 0 ditujukan untuk perangkat master [11].

Function Code: Byte kedua berupa *function code*, yaitu perintah fungsi akses data dari master yang harus dilakukan oleh *slave*. Berikut ini adalah jenis-jenis fungsi yang dapat digunakan yaitu [11]:

- 01 Read Coil Status
- 02 Read Input Status
- 03 Read Holding Registers
- 04 Read Input Registers
- 05 Force Single Coil
- 06 Force Single Register
- 07 Read Exception Status
- 08 Diagnostics
- 09 Program 484
- 10 Poll 484
- 11 Fetch Communication Event Counter
- 12 Fetch Communication Event Log
- 13 Program Controller
- 14 Poll Controller
- 15 Force Multiple Coils
- 16 Preset Multiple Registers
- 17 Report Slave ID
- 18 Program 884 / M84
- 19 Reset Comm. Link
- 20 Read General Reference
- 21 Write General Reference
- 22 Mask Write 4X Register
- 23 Read / Write 4X Registers
- 24 Read FIFO Queue.

Tabel 5.1. Register pada Modbus

Data	Read 1 Data	Write 1 Data	Write Multiple Data	No. Awal Register
Coil	FC01	FC05	FC15	00001
Input Digital	FC02			10001
Input Register	FC04			30001
Holding Register	FC03			40001

Reg: merupakan register yang ingin diambil nilainya. Frame ini menggunakan 2 byte data, maka dari itu nilai register sampai 256*256 namun dibatasi dengan jumlah register yang tersedia [11].

Length: merupakan jumlah register yang ingin direquest. Ini juga menggunakan 2 byte frame seperti register [11].

Error Check (CRC1 dan CRC2): CRC adalah Cyclic Redundancy Check yaitu sebuah metode untuk pengecekan error. CRC ini menggunakan 2 byte frame juga, untuk menghitung dan mengetahui lebih jelas dapat dilihat di CRC. Cara kerja CRC secara sederhana adalah menggunakan perhitungan matematika terhadap sebuah bilangan yang disebut dengan Checksum, yang dibuat berdasarkan total bit yang hendak ditransmisikan atau yang hendak disimpan [11].

2.6. Penyimpanan Data Pada Modbus

Pada protokol Modbus terdapat 4 jenis akses data dengan panjang masing-masing 16 bit [12].

Tabel 5.2. *Function Code Modbus*

Nama Register	Tipe Objek	Akses	Keterangan
Coils	Single Bit	Read-Write	Master bisa membaca maupun menulis data coil
Discrete Input	Single Bit	Read Only	Data hanya bisa diubah oleh *slave*
Input Register	16 bit Word	Read Only	Data hanya bisa diubah oleh *slave*

Holding Register	16-bit Word	Read-Write	Master bisa membaca dan menulis register

a) Coil

Register ini merupakan register untuk menyimpan nilai diskrit on atau off. Panjang data setiap register adalah 16 bit. Dimana untuk data yang digunakan untuk mengaktifkan coil (ON) bernilai 0xFF00. Sedangkan untuk menonaktifkan coil (OFF) bernilai 0x0000. Data ini dapat disimpan di register 00000 sampai 09999 [12].

b) Input Relay

Kebalikan dengan coil, input relay digunakan untuk mengetahui status relay apakah sedang dalam kondisi ON atau OFF. Input relay bersifat *read only* bagi master dan hanya bisa dirubah oleh *slave* yang bersangkutan. Data tersebut disimpan di register 10001 sampai 19999 [12].

c) Input Register

Input register digunakan untuk menyimpan data analog dengan *range* nilai 0 hingga 65535 . Input register bersifat *read only* bagi master. Data ini disimpan di register 30001 sampai 39999 [12].

d) Holding Register

Holding register digunakan untuk menyimpan nilai dengan *range* 0 – 65535. Register ini mempunyai alamat register 40001 sampai 49999 [12].

e) Respon Modbus Exception

Respon exception adalah respon dari *slave* ketika terjadi keadaan tidak normal/ *error*. *Slave* menerima *query*, tetapi *slave* tidak dapat menangani perintah tersebut. *Slave* akan mengirimkan sebuah respon exception. Frame respon jika terjadi kesalahan berbedan dengan frame dalam keadaan normal.

Gambar 5.6. Perbedaan frame Modbus normal dan saat terjadi exception [12].

Bila terjadi kesalahan pada pengiriman data, maka slave akan mengirimkan informasi kepada master dengan pesan *exception frame*. Berikut ini adalah daftar exception code yang terjadi bila terdapat kesalahan dalam pengiriman data:

Tabel 5.3. Exception Code [12].

Exception Code	Nama Exception	Keterangan
01	*ILLEGAL FUNCTION*	*Function code* salah
02	ILLEGAL DATA ADDRESS	Alamat register salah atau tidak tersedia.
03	ILLEGAL DATA VALUE	Mengandung nilai data yang tidak diizinkan *slave.*
04	SLAVE DEVICE FAILURE	*Slave* gagal melaksanakan perintah master
05	ACKNOWLEDGMENT	Pemberitahuan ke master bahwa pelaksanaan perintah akan memakan waktu yang lama, sehingga mengalami *time out*.
06	SLAVE DEVICE BUSY	*Slave* sedang sibuk

2.7. Modbus Message Respon

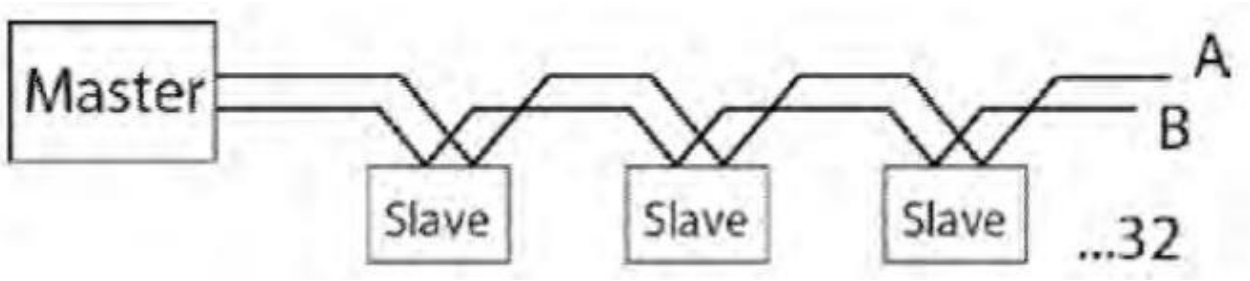
Pesan respon dari *slave* ke RTU memiliki sedikit perbedaan dengan request. Untuk pesan Modbus dari *slave* memiliki *frame* yang menyesuaikan jumlah data yang dikirimkan dimana setiap 1 data menggunaka 2 byte frame. susunan nya adalah:

[RTU Address] [Function Code] [2*panjang data] [data] [data][data…] [data…] [CRC1] [CRC2]

2.8. Komunikasi RS-485

RS485 adalah adalah teknik komunikasi data serial yang dikembangkan di tahun 1983 di mana teknik ini, komunikasi data dapat dilakukan pada jarak yang cukup jauh yaitu 1,2 km [12]. Komunikasi RS485 dapat digunakan untuk komunikasi multidrop yaitu berhubungan secara *one to many* dengan jarak yang jauh. Teknik ini dapat digunakan untuk menghubungkan 32 unit beban sekaligus hanya dengan

menggunakan dua buah kabel saja tanpa memerlukan referensi *ground* yang sama antar unit yang satu dengan unit lainnya [12].



Gambar 5.7. Topologi RS-485 [12].

Bus RS485 adalah mode transmisi *balanced differential*. Bus ini hanya mempunyai dua sinyal, A dan B dengan perbedaan tegangan antara keduanya. Karena line A sebagai referensi terhadap B maka sinyal akan high bila mendapat input low demikian pula sebaliknya. Pada komunikasi RS485, semua peralatan elektronik berada pada posisi penerima hingga salah satu memerlukan untuk mengirimkan data, maka peralatan tersebut akan berpindah ke mode pengirim, mengirimkan data dan kembali ke mode penerima. Setiap kali peralatan elektronik tersebut hendak mengirimkan data, maka terlebih dahulu harus diperiksa, apakah jalur yang akan digunakan sebagai media pengiriman data tersebut tidak sibuk. Apabila jalur masih sibuk, maka peralatan tersebut harus menunggu hingga jalur sepi [12].

Agar data yang dikirimkan hanya sampai ke peralatan elektronik yang dituju, misalkan ke salah satu Slave, maka terlebih dahulu pengiriman tersebut diawali dengan Slave ID dan dilanjutkan dengan data yang dikirimkan. Peralatan elektronik yang lain akan menerima data tersebut, namun bila data yang diterima tidak mempunyai ID yang sama dengan Slave ID yang dikirimkan, maka peralatan tersebut harus menolak atau mengabaikan data tersebut. Namun bila Slave ID yang dikirimkan sesuai dengan ID dari peralatan elektronik yang menerima, maka data selanjutnya akan diambil untuk diproses lebih lanjut [12].

D. ALAT DAN BAHAN

Tabel 5.4. Alat dan Bahan Praktikum

No	Alat	Jumlah
1	Laptop dengan *software* Arduino IDE, PSoC Creator, dan Visual Studio Code	2
2	Mikrokontroler Arduino UNO	2
3	Mikrokontroler PSoC	2
4	Modul MAX485	1
5	RS485-to-USB	1
6	Kabel 1 m, 2 m, dan 6 m	Masing-masing 2 buah
7	Kabel jumper	Secukupnya
8	Breadboard	1

E. LANGKAH PERCOBAAN

Percobaan 1: Komunikasi UART antar Arduino

1. Unduh Arduino IDE versi 1.8.18 pada tautan berikut
2. Buka Arduino IDE dan buat program untuk pengiriman data protokol komunikasi UART seperti **Gambar 5.8.**

```
void setup(){
  Serial.begin(57600);
}

void loop(){
  Serial.print("Data yang dikirim");
  delay(2000);
}
```

Gambar 5.8. Kode Program Arduino UART Pengirim

3. Buat program untuk penerima data protokol komunikasi UART menggunakan Arduino seperti **Gambar 5.9.**

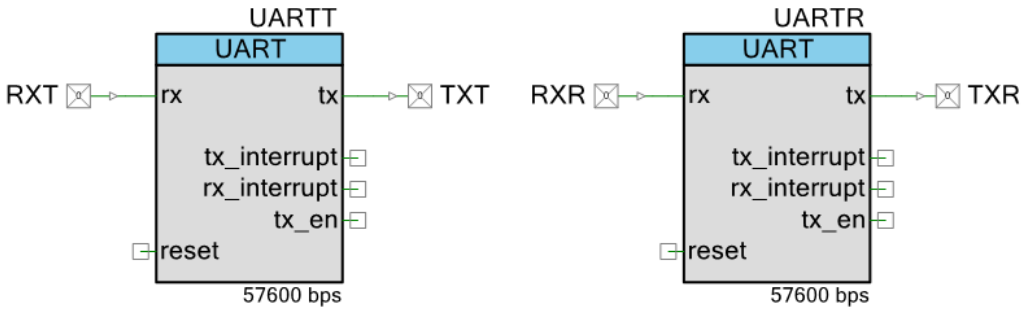
```
String data;

void setup(){
  Serial.begin(9600);
}

void loop(){
  if (Serial.available()){
    data = Serial.readString();
    Serial.print(data);
  }
}
```

Gambar 5.9. Kode Program Arduino UART Penerima

4. *Verify* program untuk memastikan program yang dibuat sudah tepat atau belum. Jika sudah, tekan *upload* untuk memasukkan program yang dibuat ke masing-masing Arduino.
5. Hubungkan RX TX kedua Arduino dengan menyilang RX → TX dan TX → RX.
6. Lakukan pengiriman data sesuai tabel percobaan.
7. Pada laptop penerima, buka serial monitor pada Arduino IDE. Sesuaikan *baudrate* seperti pada tabel percobaan.
8. Catat hasil yang diperoleh pada tabel percobaan.

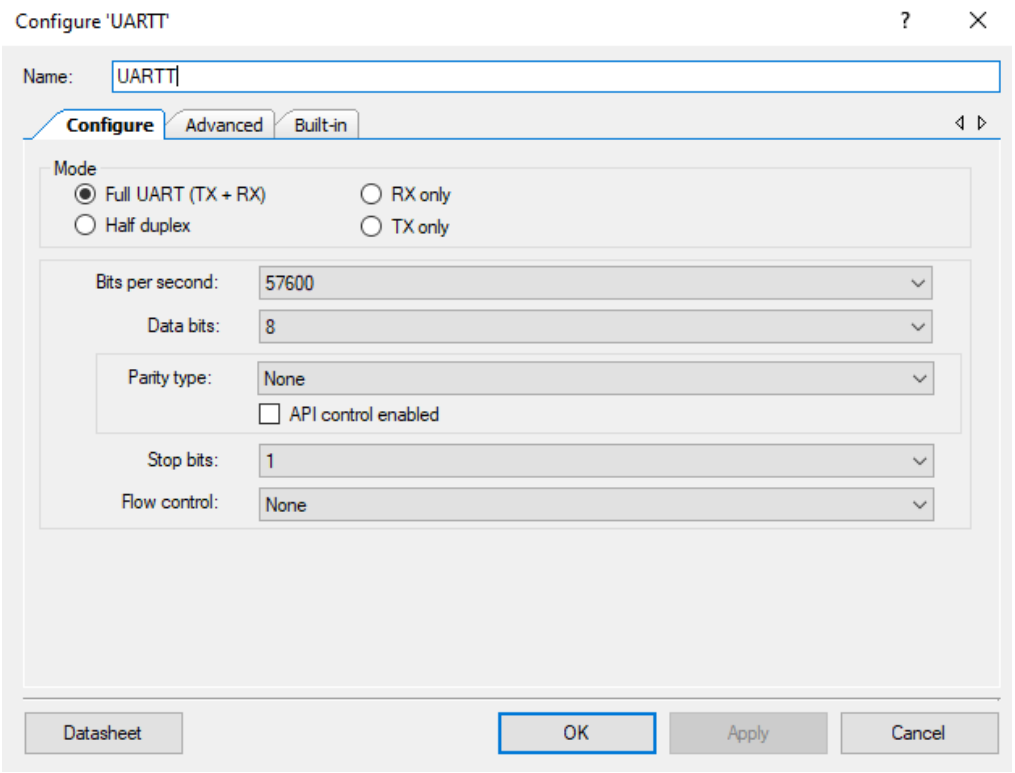
Percobaan 2: Komunikasi UART antar PSoC

1. Buka PSoC Creator dan masuk ke bagian TopDesign.cysh.
2. Masukkan komponen UART yang didapatkan dari Component Catalog seperti berikut.



Gambar 5.10. Blok komponen UART pada PSoC

3. Atur konfigurasi blok UART masing-masing PSoC dengan *double click* blok UART.



Gambar 5.11. Konfigurasi blok UART

4. Buka main.c dan buat program seperti berikut.

```
#include "project.h"
char data, data2;

int main(void)
{
    CyGlobalIntEnable;

    UARTT_Start();
    UARTR_Start();

    for(;;)
    {
        data = UARTT_GetChar();
        if(data){
            UARTT_PutString(data);
        }

        data2 = UARTT_GetChar();
        if(data2){
            UARTT_PutString(data);
        }
    }
}
```

Gambar 5.12. Kode program UART pada main.c

5. Atur konfigurasi port UART PSoC pada bagian Pins seperti berikut.

Tabel 5.5. Konfigurasi port UART pada PSoC

RXT	P2[1]
TXT	P2[2]
RXR	P12[6]
TXR	P12[7]

6. Variasikan *baudrate* masing-masing PSoC dan kirim data ke penerima melalui serial monitor Arduino IDE sesuai tabel percobaan.
7. Buka serial monitor di Arduino IDE pada laptop yang tersambung dengan PSoC penerima.
8. Catat hasil yang diperoleh pada tabel percobaan.

Percobaan 3: *Data Frame* Komunikasi UART

1. Buka PSoC Creator dan masuk ke bagian TopDesign.cysh.
2. Masukkan komponen UART dari Component Catalog dan atur konfigurasinya seperti pada **Gambar 5.10.** dan **Gambar 5.11.**
3. Buat kode program pada main.c seperti berikut dan atur konfigurasi port seperti pada **Tabel 5.5.**

```
#include "project.h"

int main(void)
{
    CyGlobalIntEnable;
    UARTT_Start();
    for(;;)
    {
        UARTR_PutString('D');
    }
}
```

Gambar 5.13. Kode program *data frame* UART pada PSoC

4. *Clean and build* lalu upload program ke PSoC.
5. Variasikan *stop-bit* dan *parity-bit* sesuai dengan tabel percobaan.
6. Buka serial monitor Arduino IDE untuk mengirim karakter.
7. Amati *data frame* menggunakan osiloskop dengan menghubungkan pin TX PSoC dengan *probe* osiloskop.

Percobaan 4: Komunikasi UART antara Laptop, Arduino, dan PsoC

1. Buka ENV python pada Visual Studio Code dan buat program seperti berikut.

```
import serial
import time

com = serial.Serial(port='COM50',
                    baudrate=19200,
                    parity=serial.PARITY_ODD,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS,
                    timeout=3)
data = "Jarkom Seru"
for i in range(20):
    com.write(data)
    print(data)
    time.sleep(1)
com.close()
```

Gambar 5.14. Program python pengiriman data

2. Siapkan Arduino kemudian buka Arduino IDE dan buat program seperti **Gambar 5.15.**
3. Hubungkan pin 10,11 (RX, TX) arduino ke pin PSoC 2[2], 2[1] (TX, RX).

```
#include <SoftwareSerial.h>

SoftwareSerial Serial1 (10, 11);
char data;

void setup(){
  Serial.begin(57600);
  Serial1.begin(57600);
  while (!Serial){
    ;
  }
}

void loop(){
  if (Serial1.available() > 0) {
    data = Serial1.read();
    Serial.write(data);
  }
  if (Serial.available() > 0) {
    data = Serial.read();
    Serial1.write(data);
  }
}
```

Gambar 5.15. Program Arduino

4. *Verify* dan *upload* kode program ke Arduino.
5. Buka PSoC Creator dan masuk ke bagian TopDesign.cysh
6. Masukkan komponen UART, atur konfigurasi bloknya, dan buat kode program pada main.c seperti pada **Gambar 5.10.** hingga **Gambar 5.11.**
7. Atur konfigurasi port UART PSoC pada bagian Pins seperti **Tabel 5.5.**
8. Buat kode program python pada laptop penerima seperti berikut.

```
import serial
import csv
from datetime import datetime

com = serial.Serial(port='COM50',
                    baudrate=9600,
                    parity=serial.PARITY_EVEN,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS,
                    timeout=3)
jumlah_data = 0

while jumlah_data < 20:
    file = open("Data_UART.txt", "w")
    write_file = csv.writer(file)
    write_file.writerow(["Waktu", "Data"])

    while com.inWaiting() == 0:
        pass
    com_bytes = com.readline()
    data = com_bytes.decode("unicode").strip()
    waktu = str(datetime.now()).split(" ")[1].split(".")[1]
    print(waktu, data)

    write_file.writerow([data, waktu])
    jumlah_data += 1
    file.close()

com.close()
```

Gambar 5.16. Kode program python penerima data

Percobaan 5: Modbus

1. Buka Arduino IDE dan tambahkan *library* ModbusRtu dengan menambahkan berkas zip berikut: https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino/tree/master
2. Buat kode program Modbus *slave* dan *upload* ke Arduino.

```
#include <ModbusRtu.h>
#define TXEN 3

uint16_t au16data[16] = {
  0, 25, 50, 11, 33, 45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  int Ledstatus = 0;
  int ledPin = 13;

Modbus slave(1, Serial, TXEN);

void setup(){
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  slave.start();
}

void loop(){
  slave.poll(au16data, 4);
  if(Ledstatus != au16data[0]){
    Ledstatus = au16data[0];
    digitalWrite(ledPin, Ledstatus);
  }
}
```

Gambar 5.17. Program Modbus *slave*

3. Buat kode program Modbus *master* pada laptop seperti berikut.

```
import minimalmodbus

data = minimalmodbus.Instrument("COM50", 5)
data.serial.baudrate = 57600

while True:
    try:
        ledState = input("Set LED state (0/1): ")
        if ledState not in ['0', '1']:
            print("Invalid input. Please enter 0 or 1.")
            continue

        data.write_register(0, int(ledState), 0)
        registerselect = input("Select Register (0-15) to read: ")
        readRegister = data.read_register(int(registerselect), 0)
        print(f"Value from register {registerselect}: {readRegister}")

    except Exception as e:
        print(f"Error: {e}")
```

Gambar 5.18 Program Modbus *master*

4. Hubungkan Arduino UNO dengan modul MAX485 menggunakan RS485-to-USB.

Tabel 5.6. *Pinout* Arduino dengan MAX485

Arduino	**MAX485**
Digital 3	RE
Digital 3	DE
Digital 0	R0
Digital 1	D1
GND	GND
VCC	VCC

5. Hubungkan pin A dan B pada modul MAX485 dengan pin A dan B pada RS485-to-USB.
6. Jalankan program Modbus *master* dan variasikan percobaan sesuai dengan tabel percobaan.

F. TABEL PERCOBAAN

Percobaan 1 dan 2: Komunikasi UART antar Arduino dan PSoC

Tabel 5.7. Variasi *Baudrate* Pada Komunikasi UART

Mikrokontroler	**Data yang dikirim***	***Baudrate* Pengirim**	***Baudrate* Penerima**	**Data yang diterima**
Arduino	SSTK	9600	9600	
		115200	4800	
		2400	57600	
PSoC	Admin123	9600	9600	
		115200	4800	
		2400	57600	
Kesimpulan sementara				

Percobaan 3: *Data Frame* Komunikasi UART

Tabel 5.8. *Data Frame* Komunikasi UART Pada PSoC

Karakter yang dikirim*	**Kombinasi Biner 8 bit**	**Stop-bit**		**Parity**			**Tampilan Pada Osiloskop**
		1	**2**	**Odd**	**Even**	**None**	
D		v				v	
		v			v		
		v		v			
			v			v	
Kesimpulan sementara							

Nb: jangan lupa untuk memotret tampilan pada osiloskop

**: Data yang dikirim sesuai dengan instruksi asisten praktikum masing-masing*

Percobaan 4: Komunikasi UART antara Laptop, Arduino, dan PSoC

Tabel 5.9. Komunikasi UART antar Mikrokontroler

Mikrokontroler Pengirim	**Data yang dikirim***	***Baudrate***	**Mikrokontroler Penerima**	**Data yang diterima**
Arduino	Keberlanjutan	9600	PSoC	
PSoC	Hokage	9600	Arduino	
Kesimpulan sementara				

Percobaan 5: Interface Modbus

Tabel 5.10. Komunikasi Modbus

Arduino		**Laptop**			**Panjang kabel (m)**	**Hasil**
Slave address	***Register limit***	***Slave address***	***Register read***	**LED *state***		
1	4	1	2	1	1	
1	16	1	5	0	1	
1	16	5	8	1	1	
1	16	1	5	1	2	
1	16	1	7	1	6	

G. REFERENSI

[1] E. Pena dan M. G. Legaspi, “UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter,” *Analog Dialogue,* vol. 54, no. 4, pp. 1-5, 2020.

[2] Infineon, “Universal Asynchronous Receiver Transmitter (UART),” dalam *PSoC® Creator™ Component Datasheet,* San Jose, Cypress Semiconductor Corporation, 2016, p. 1.

[3] I. Prastyawan, Aplikasi Sistem Kontrol dengan Komunikasi Serial RS-485 Menggunakan Mikrocontroller pada Bangunan Bertingkat, Depok: Universitas Indonesia, 2008.

[4] Maxim Integrated Products, "The evolution of the RS-232 transceiver," 18 Juni 2010. [Online]. Available: https://www.analog.com/en/resources/technical-articles/the-evolution-of-the-rs232-transceivers.html. [Accessed 30 September 2024].

[5] Jimblom, “RS-232 vs. TTL Serial Communication,” Sparkfun, 23 November 2010. [Online]. Available: https://www.sparkfun.com/tutorials/215. [Diakses 30 September 2024].

[6] P. Diyos dan N. Ziyada, “Mengenal Protokol Modbus,” Kamalogis, 18 Agustus 2021. [Online]. Available: https://kamalogis.ft.ugm.ac.id/2021/08/18/mengenal-protokol-modbus/. [Diakses 30 September 2024].

[7] Amstel Rectifiers, “Cathodic Protection Products - Ships,” [Online]. Available: https://amstels.com/cathodic-protection-rectifiers/c-products/ships/13-o-amx-p/56-option-rs1. [Diakses September 30 2024].

[8] D. M. Azhar and Yuniarto, "RANCANG BANGUN SIMULASI DISPLAY ARUS GANGGUAN RELAI PROTEKSI SEL 551 DI MASTER STATION PADA SCADA SURVALENT SEBAGAI UPAYA MEMPERBAIKI SAIDI," *GEMA TEKNOLOGI,* vol. 18, no. 2, pp. 58-60, 2014.

[9] A. P. Abseno, Perancangan Program PLC Untuk Mesin Pengisian Botol pada PT. Kairos Solusi Indonesia, Surabaya: Institut Bisnis dan Informatika STIKOM Surabaya, 2018.

[10] F. Firman, Rancang Bangun Prototype Sistem Kontrol Ketinggian Air Feed Water Tank pada PLTU Berbasis Smart Relay dan Sistem Scada, Malang: Institut Teknologi Malang, 2016.

[11] R. Imanto, “Modbus Protokol dan Serial Standard,” Rifqi-On Com, 6 Desember 2023. [Online]. Available: https://www.rifqion.com/menulis/modbus-protokol-dan-serial-standard/. [Diakses 2024 September 2024].

[12] S. D. Chandra, DESAIN DAN IMPLEMENTASI PROTOKOL MODBUS UNTUK SISTEM ANTRIAN TERINTEGRASI PADA PELAYANAN SURAT, Surabaya: Institut Teknologi Sepuluh Nopember, 2016.