Đại học Sư Phạm Kỹ Thuật Tp.HCM
Khoa Cơ Khí Chế Tạo Máy
Bộ môn Cơ Điện tử

BÀI GIẢNG
VI XỬ LÝ (MICO236929)

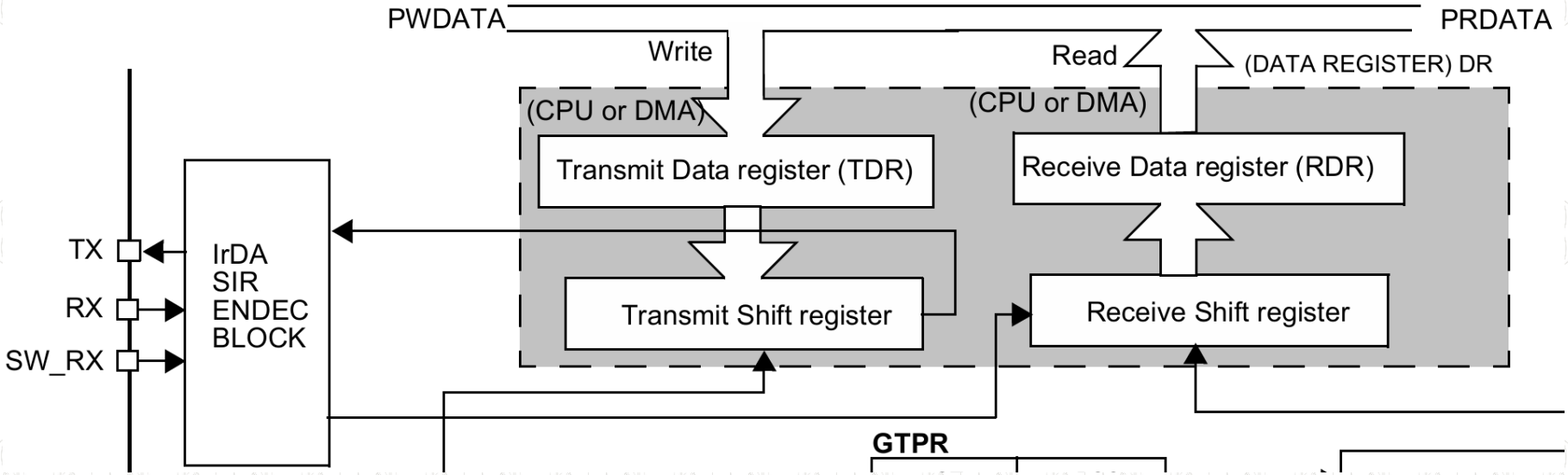
11/2024

STM32 UART Introduction

Universal **A**synchronous **R**eceiver/**T**ransmitter is the hardware circuitry (module) being used for serial communication. UART is sold/shipped as a standalone integrated circuit (IC) or as an internal module within microcontrollers.

UART – Universal Asynchronous Receiver/Transmitter
USART – Universal Synchronous/Asynchronous Receiver/Transmitter

Figure 279. USART block diagram



```
HAL_UART_Transmit(UART_HandleTypeDef * huart,
                  const uint8_t * pData,
                  uint16_t Size,
                  uint32_t Timeout);
```

Gởi <**Size**> dữ liệu trong RAM tại địa chỉ <**pData**> qua <**huart**>. Nếu gởi xong Size bytes hoặc hết thời gian Timeout thì CPU kết thúc lệnh gởi để thực hiện lệnh tiếp theo.

Ví dụ:

```
uint8_t tx_buff[]={65,66,67,68,69,70,71,72,73,74};
//ABCDEFGHIJ in ASCII code
HAL_UART_Transmit(&huart1, tx_buff, 10, 100);

uint8_t MSG[20] = {};
sprintf(MSG, "Hello World \n");
HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
```

HAL_UART_**Receive**(UART_HandleTypeDef * **huart**,
uint8_t * **pData**,
uint16_t **Size**,
uint32_t **Timeout**);

Chờ nhận <Size> byte dữ liệu từ <huart> để cất vào RAM tại địa chỉ pData. Nếu nhận đủ Size byte hoặc quá thời gian Timeout thì ngừng lệnh và thực hiện lệnh tiếp theo.

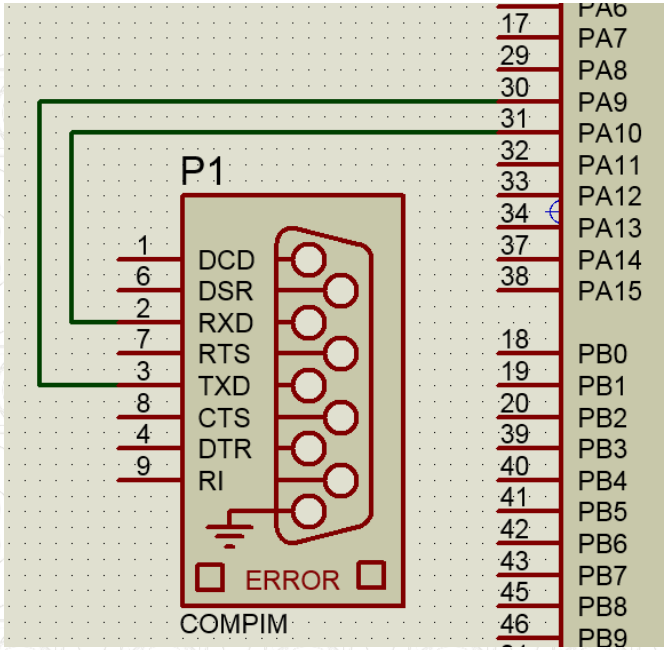
Ví dụ:

```
uint8_t rxBuffer[12] = {0};
HAL_UART_Receive(&huart1, rxBuffer, 12, 5000);
```

```
struct __FILE {
  int handle;
};

FILE __stdout;
int fputc(int ch, FILE *f) {
     HAL_UART_Transmit(&huart1,(uint8_t *)&ch, 1, HAL_MAX_DELAY);
    return ch;
}

FILE __stdin;
int fgetc(FILE *f) {
    uint8_t ch = 0;
     __HAL_UART_CLEAR_OREFLAG(&huart1);
     HAL_UART_Receive(&huart1,(uint8_t *)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
```




```
while (1)
  {
      char str[80];
      int i;
      printf("Enter a number: ");
      scanf("%d", &i);
      printf("You enter %d \n", i);

      printf("Enter text: ");
      scanf("%s", str);
      printf("You enter: %s\n", str);
}
```

```
HAL_UART_Transmit_IT(UART_HandleTypeDef * huart,
                     const uint8_t * pData,
                     uint16_t Size);

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
  // Handle UART TX Interrupt Here!
}
```

Gởi **Size** byte dữ liệu tại **pData** ra **huart** theo cơ chế ngắt:
Trong khi gởi 1 byte, MCU không đợi hoàn tất các bit mà chỉ ghi vào "buffer gởi đi" rồi quay lại chương trình chính. Khi nào bit cuối cùng gởi xong thì MCU lại ngừng chương trình chính để quay lại ghi vào buffer gởi byte tiếp theo.
Khi quá trình gởi **Size** byte hoàn tất, hàm ngắt **TxCpltCallback** được gọi để báo hiệu đã gởi xong chuỗi **Size** byte tại **pData**.

```
HAL_UART_Receive_IT(UART_HandleTypeDef * huart,
                    uint8_t * pData,
                    uint16_t Size);

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  // Handle UART RX Interrupt Here!
}
```

Nhận **Size** byte dữ liệu ghi vào **pData** theo cơ chế ngắt**:**
Khi "buffer nhận" nhận đủ 1 byte dữ liệu, CPU ngắt chương trình chính để ghi dữ liệu từ buffer nhận vào pData, sau đó quay lại chương trình chính.
Khi quá trình nhận đủ **Size** bytes, hàm ngắt **RxCpltCallback** được gọi để báo hiệu đã nhận xong chuỗi **Size** byte và ghi hết vào **pData**.

```
int main(void)
{
  HAL_Init();  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  HAL_UART_Receive_IT(&huart1, UART1_rxBuffer, 12);
  while (1)  {
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  HAL_UART_Transmit(&huart1, UART1_rxBuffer, 12, 100);
  HAL_UART_Receive_IT(&huart1, UART1_rxBuffer, 12);
}
```

```
int main(void)
{
  HAL_Init();  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  HAL_UART_Receive_IT(&huart1, UART1_rxBuffer, 12);
  while (1)  {
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  HAL_UART_Transmit(&huart1, UART1_rxBuffer, 12, 100);
  HAL_UART_Receive_IT(&huart1, UART1_rxBuffer, 12);
}
```

- **Time-Out Mechanism**: Nhận dữ liệu liên tục; Khi nào không có dữ liệu đến thì bắt đầu đếm timeout để khi tràn timeout thì ngừng nhận dữ liệu và xử lý dữ liệu đã nhận.

- **Fixed-Size Buffer + Overrun Detection**: Đếm số lượng byte đã nhận, nếu đủ số qui ước thì ngừng nhận và xử lý dữ liệu đã nhận.

- **Delimiter-Based Reception**: Kiểm tra ký tự kết thúc chuỗi nhận (tự qui ước). Nếu phát hiện ký tự kết thúc chuỗi nhận thì ngừng nhận và xử lý dữ liệu đã nhận.

- **Packet Header With Data Length Info**: Qui ước byte nhận đầu có thông tin kích thước gói dữ liệu

- **UART IDLE Line Detection**: Utilize the STM32's UART hardware feature to detect periods of inactivity on the receiving line.

UART IDLE Line Detection

```
uint8_t UART1_RxBuffer[40] = {0};
uint16_t RxDataLen = 0;
int main(void)
{
 HAL_UARTEx_ReceiveToIdle_IT(&huart1, UART1_RxBuffer, 40);
// Nhận RxBuffer theo cơ chế ngắt;
// Khi nhận đủ 40 bytes hoặc line idle thì gọi hàm ngắt
 while (1)  {
    // Nothing ToDo Here!
  }
}

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
   RxDataLen = Size;
   HAL_UART_Transmit(&huart1, UART1_RxBuffer, RxDataLen, 100);
   HAL_UARTEx_ReceiveToIdle_IT(&huart1, UART1_RxBuffer, 40);
}
```