BIRZEIT UNIVERSITY

Department of Electrical & Computer Engineering
ENCS4110 – Computer Design Laboratory

Experiment 09

Universal Asynchronous Receiver-Transmitter (UART)

Date: December 2025

9 Universal Asynchronous Receiver-Transmitter (UART)

Learning Objectives

After completing this experiment, you will be able to:

- Understand the principles of asynchronous serial communication including start bits, stop bits, and data frames.
- Configure UART baud rate using integer and fractional divisors for precise timing.
- Configure GPIO pins for UART alternate function using AFSEL, PCTL, and DEN registers.
- Implement basic UART transmission and reception functions in C.
- Send and receive strings via serial communication using polling methods.
- Interface with a PC terminal emulator for debugging and data visualization.
- Calculate and verify baud rate divisors for different system clock frequencies.
- Understand UART FIFO buffering and status flag operations.

Experiment Overview

This experiment introduces serial communication using the TM4C123's UART modules. You will configure UART0 for 115200 baud operation on pins PA0 (RX) and PA1 (TX), implement character and string transmission functions, and create an echo server that receives data from a PC terminal and echoes it back. By the end of this lab, you will understand how asynchronous serial communication enables data exchange between embedded systems and external devices, forming the foundation for debugging, sensor interfacing, and wireless communication modules.

Contents

1 Theoretical Background

1.1 Introduction to Serial Communication

Serial communication is a method of transmitting data one bit at a time over a single communication line, as opposed to parallel communication which sends multiple bits simultaneously. It is widely used in embedded systems due to its simplicity, low pin count, and ability to communicate over long distances.

1.1.1 Asynchronous vs Synchronous Communication

Serial communication can be classified into two categories:

- **Asynchronous (UART)**: No shared clock signal; timing is established by agreed-upon baud rate. Uses start and stop bits for synchronization.
- **Synchronous (SPI, $\mathbf{I^2C}$)**: Dedicated clock line synchronizes transmitter and receiver. More efficient but requires additional pin.

UART (Universal Asynchronous Receiver-Transmitter) is an asynchronous protocol that requires only two wires: TX (transmit) and RX (receive). It is commonly used for debugging, GPS modules, Bluetooth communication, and PC interfacing.

1.2 UART Frame Structure

A UART transmission consists of a **data frame** that includes:

1. **Start Bit**: Logic low (0) that signals the beginning of transmission
2. **Data Bits**: 5 to 9 bits of actual data (typically 8 bits)
3. **Parity Bit**: Optional error-checking bit (even, odd, or none)
4. **Stop Bits**: Logic high (1) that signals the end of transmission (1, 1.5, or 2 bits)

The most common configuration is **8N1**: 8 data bits, No parity, 1 stop bit.

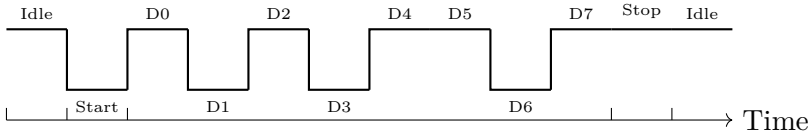

Figure 9.1: UART Frame Structure (8N1 Format)

1.3 Baud Rate Generation

The **baud rate** defines the number of bits transmitted per second, measured in bits per second (bps). Common baud rates include 9600, 19200, 38400, 57600, and 115200 bps. Both transmitter and receiver must use the same baud rate for successful communication.

1.3.1 Baud Rate Divisor Calculation

The TM4C123 UART uses a **22-bit baud rate divisor (BRD)** consisting of a 16-bit integer part (BRDI) and a 6-bit fractional part (BRDF). This fractional divisor allows the UART to generate all standard baud rates with high precision.

The baud rate divisor is calculated from the system clock frequency as:

$$\text{BRD} = \text{BRDI} + \text{BRDF} = \frac{\text{UARTSysClk}}{\text{ClkDiv} \times \text{BaudRate}} \tag{9.1}$$

where:

- **UARTSysClk**: System clock connected to UART (typically 50 MHz)
- **ClkDiv**: Clock divider (16 if HSE = 0 in UARTCTL, or 8 if HSE = 1)
- **BaudRate**: Desired baud rate (e.g., 115200 bps)

The integer part (BRDI) is loaded into the `UARTIBRD` register, and the fractional part is calculated and loaded into the `UARTFBRD` register using:

$$\text{UARTFBRD[DIVFRAC]} = \text{integer}(\text{BRDF} \times 64 + 0.5) \tag{9.2}$$

The multiplication by 64 and addition of 0.5 accounts for the 6-bit fractional representation and rounding errors.

1.3.2 Baud Rate Generation Process

The UART generates an internal baud rate reference clock at either 8x or 16x the target baud rate (called Baud8 or Baud16), depending on the HSE bit setting. This reference clock is then divided by 8 or 16 to generate the actual transmit clock and is used for error detection during receive operations.

1.3.3 Calculation Example

Example: For 115200 baud with 50 MHz system clock and ClkDiv = 16 (default, HSE = 0):

$$\begin{aligned}
\text{BRD} &= \frac{50{,}000{,}000}{16 \times 115{,}200} = \frac{50{,}000{,}000}{1{,}843{,}200} = 27.1267361 \\
\text{BRDI} &= 27 \\
\text{BRDF} &= 0.1267361 \\
\text{UARTFBRD[DIVFRAC]} &= \text{integer}(0.1267361 \times 64 + 0.5) \\
&= \text{integer}(8.111 + 0.5) = \text{integer}(8.611) = 8
\end{aligned}$$

Therefore: `UARTIBRD = 27` and `UARTFBRD = 8`
The actual baud rate achieved can be verified:

$$\text{Actual Baud Rate} = \frac{50{,}000{,}000}{16 \times (27 + 8/64)} = \frac{50{,}000{,}000}{16 \times 27.125} = 115{,}207.4 \text{ bps} \tag{9.3}$$

This gives an error of only 0.0064%, well within acceptable tolerance for reliable communication.

1.4 TM4C123 UART Modules

The TM4C123GH6PM provides **eight UART modules** (UART0-UART7) with the following features:

- **Programmable Baud Rate**: Generated from system clock with 16-bit divisor
- **Data Format**: 5 to 9 data bits, optional parity (even, odd, stick), 1 or 2 stop bits
- **FIFO Buffers**: 16-byte transmit and receive FIFOs for reduced interrupt overhead
- **Full-Duplex Operation**: Simultaneous transmission and reception
- **Error Detection**: Framing, parity, overrun, and break detection
- **Interrupt Generation**: Configurable interrupts for TX, RX, and error conditions
- **DMA Support**: μDMA channels for high-speed data transfer
- **IrDA and ISO 7816**: Optional infrared and smart card support
- **9-Bit Mode**: For RS-485 multi-drop networks

1.4.1 FIFO Operation

Each UART module contains two 16x8 FIFOs: one for transmit and one for receive. Both FIFOs are accessed through the `UARTDR` register. Read operations return a 12-bit value consisting of 8 data bits and 4 error flags, while write operations place 8-bit data into the transmit FIFO.

FIFO Status Monitoring:

FIFO status can be monitored using two registers:

1. **UARTFR (Flag Register)**: Contains empty and full flags
   - **TXFE** (bit 7): Transmit FIFO Empty
   - **TXFF** (bit 5): Transmit FIFO Full
   - **RXFE** (bit 4): Receive FIFO Empty
   - **RXFF** (bit 6): Receive FIFO Full

1.4.2 UART Pin Mapping

Each UART module is connected to specific GPIO pins. The table below shows the default pin assignments:

UART Module	TX Pin	RX Pin
UART0	PA1	PA0
UART1	PB1	PB0
UART2	PD7	PD6
UART3	PC7	PC6
UART4	PC5	PC4
UART5	PE5	PE4
UART6	PD5	PD4
UART7	PE1	PE0

Table 9.1: UART Pin Assignments

Note: UART0 on PA0/PA1 is commonly used because it connects to the LaunchPad's on-board USB-to-Serial converter, allowing direct PC communication via USB cable.

1.5 UART Registers

The TM4C123 UART modules are configured through memory-mapped registers. Key registers include:

RCGCUART — UART Run Mode Clock Gating Control

Enables clock for UART modules. Each bit corresponds to a UART module (1 = enable, 0 = disable).

Figure 9.2: RCGCUART Register — UART Run Mode Clock Gating Control

UARTCTL — UART Control

Configures UART operation including enable, transmit enable, and receive enable.

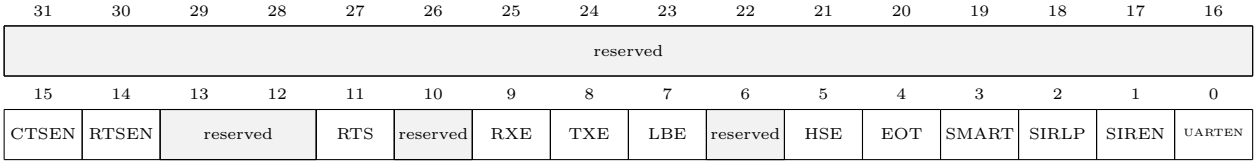

Figure 9.3: UARTCTL Register — UART Control

Where:

- **UARTEN** (bit 0): UART Enable (1 = enable, 0 = disable)
- **TXE** (bit 8): Transmit Enable (1 = enable transmitter)
- **RXE** (bit 9): Receive Enable (1 = enable receiver)
- **HSE** (bit 5): High-Speed Enable (1 = baud rate clock = SysClk/8, 0 = SysClk/16)

UARTIBRD — UART Integer Baud-Rate Divisor

Contains the integer part of the baud rate divisor.

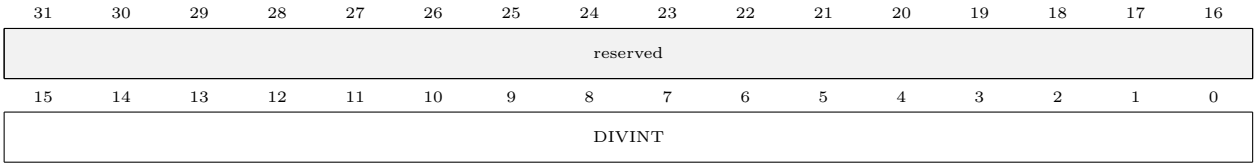

Figure 9.4: UARTIBRD Register — UART Integer Baud-Rate Divisor

Where:

- **DIVINT**: Integer baud rate divisor (0-65535)

UARTFBRD — UART Fractional Baud-Rate Divisor

Contains the fractional part of the baud rate divisor.

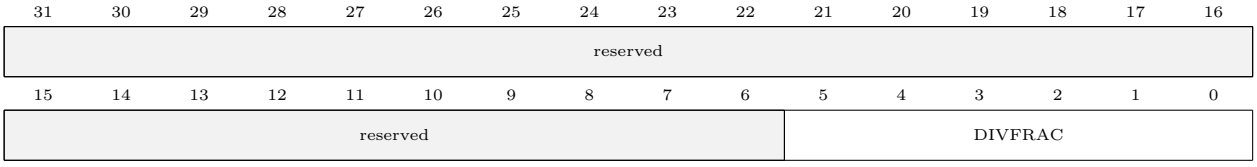

Figure 9.5: UARTFBRD Register — UART Fractional Baud-Rate Divisor

Where:

- **DIVFRAC**: Fractional baud rate divisor (0-63)

UARTLCRH — UART Line Control

Configures data format including word length, FIFO enable, parity, and stop bits.

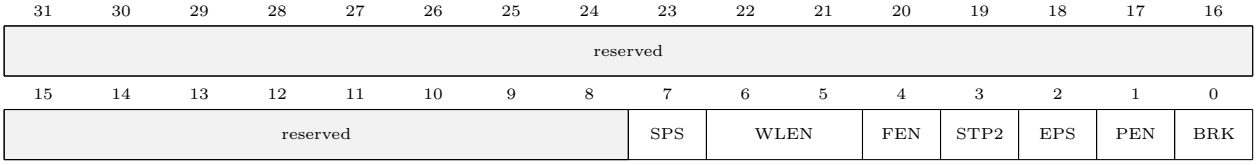

Figure 9.6: UARTLCRH Register — UART Line Control

Where:

- **WLEN** (bits 6-5): Word Length (0x3 = 8 bits, 0x2 = 7 bits, 0x1 = 6 bits, 0x0 = 5 bits)
- **FEN** (bit 4): FIFO Enable (1 = enable 16-byte FIFOs, 0 = disable)
- **STP2** (bit 3): Two Stop Bits (1 = two stop bits, 0 = one stop bit)
- **EPS** (bit 2): Even Parity Select (1 = even parity, 0 = odd parity)
- **PEN** (bit 1): Parity Enable (1 = enable parity, 0 = no parity)
- **BRK** (bit 0): Send Break (1 = send break condition)

Common Configuration (8N1 with FIFO):

- WLEN = 0x3 (8 bits)
- FEN = 1 (FIFO enabled)
- STP2 = 0 (one stop bit)
- PEN = 0 (no parity)

Result: `UARTLCRH = 0x60` (0b01100000)

UARTCC — UART Clock Configuration

Selects the clock source for UART baud rate generation.

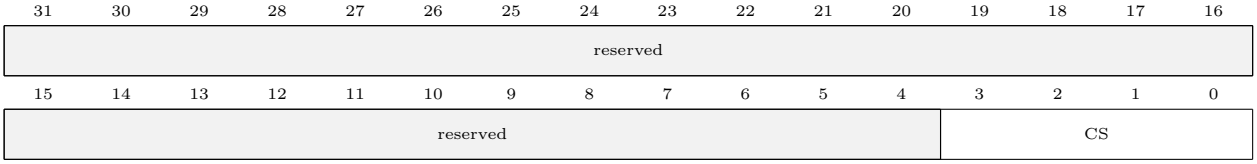

Figure 9.7: UARTCC Register — UART Clock Configuration

Where:

- **CS** (bits 3-0): Clock Source
  - 0x0 = System clock (default, recommended)
  - 0x5 = PIOSC (16 MHz internal oscillator)

UARTFR — UART Flag Register

Contains status flags for transmit/receive operations.

Figure 9.8: UARTFR Register — UART Flag Register

Where:

- **TXFE** (bit 7): Transmit FIFO Empty (1 = empty, 0 = not empty)
- **RXFF** (bit 6): Receive FIFO Full (1 = full, 0 = not full)
- **TXFF** (bit 5): Transmit FIFO Full (1 = full, 0 = not full)
- **RXFE** (bit 4): Receive FIFO Empty (1 = empty, 0 = not empty)
- **BUSY** (bit 3): UART Busy (1 = transmitting, 0 = idle)

Usage:

- Before transmitting: Wait while `TXFF = 1` (transmit FIFO full)
- Before receiving: Wait while `RXFE = 1` (receive FIFO empty)

UARTDR — UART Data Register

Contains the data to transmit or received data. Reading removes data from RX FIFO; writing adds to TX FIFO.

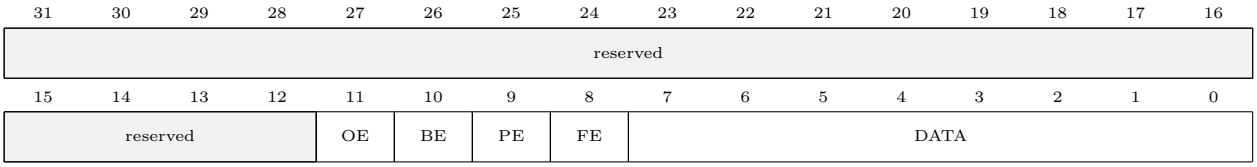

Figure 9.9: UARTDR Register — UART Data Register

Where: Data is contained in the lower 8 bits. The upper 4 bits indicate error flags:

- **FE** (bit 3): Framing Error
- **PE** (bit 2): Parity Error
- **BE** (bit 1): Break Error
- **OE** (bit 0): Overrun Error
- **DATA** (bits 7-0): Received or Transmit Data

2 Procedure

2.1 Configuration Steps

Configuring UART0 on the TM4C123 microcontroller involves two stages:

2.1.1 GPIO Pin Configuration

To prepare GPIO pins for UART function:

1. Enable the clock for UART0 module and GPIO Port A by setting the appropriate bits in the `RCGCUART` and `RCGCGPIO` registers.
2. Enable alternate function on PA0 (RX) and PA1 (TX) using the `GPIOAFSEL` register.
3. Configure the UART function encoding in the `GPIOPCTL` register (encoding = 1 for UART).
4. Enable digital function for PA0 and PA1 pins by setting bits in the `GPIODEN` register.

2.1.2 UART Module Setup

After the pins are configured, the UART module must be initialized:

1. Disable the UART by clearing all bits in the `UARTCTL` register to prevent undefined behavior during configuration.
2. Calculate the baud rate divisor values. For 115200 baud at 50 MHz: BRD = 50,000,000 / (16 x 115,200) = 27.1267361, giving IBRD = 27 and FBRD = 8.
3. Write the integer baud rate divisor to the `UARTIBRD` register.
4. Write the fractional baud rate divisor to the `UARTFBRD` register.
5. Configure the line control register (`UARTLCRH`) for 8 data bits, no parity, one stop bit, and FIFO enable (value = 0x60). Writing to this register latches the baud rate configuration.
6. Select the clock source by writing to the `UARTCC` register (0 = system clock).
7. Enable the UART, transmitter, and receiver by setting UARTEN, TXE, and RXE bits in the `UARTCTL` register.

2.2 UART0 Initialization and Code Implementation

The provided code includes three files demonstrating complete UART functionality:

- `uart.h`: Function prototypes and pin definitions
- `uart.c`: Implementation of UART initialization, read, and write functions
- `main.c`: Echo server application

UART Header File (`uart.h`)

```c
#ifndef UART_H
#define UART_H

#include "TM4C123.h"  // Or your MCU's main header

#define U0_TX 2
#define U0_RX 1

void UART0_WriteChar(char c);
void UART0_WriteString(char *str);

char UART0_ReadChar();
void UART0_ReadString(char *buffer, int maxLen);

void UART0_Init();




#endif // UART_H
```

Listing 9.1: UART Header File

UART Implementation File (`uart.c`)

```c
#include "uart.h"
#define MAX_STR_LEN 50
// Function to send a single character via UART0
void UART0_WriteChar(char c) {
    // Wait until the transmit FIFO is not full
    // UART0->FR bit 5 (TXFF) = 1 means FIFO is full, so wait until it becomes 0
    while ((UART0->FR & (1 << 5)) != 0);
    // Write the character to the Data Register to transmit
    UART0->DR = c;
}

// Function to send a null-terminated string via UART0
void UART0_WriteString(char *str) {
    // Loop through each character until null terminator
    while (*str) {
        // Send each character using UART0_WriteChar
        UART0_WriteChar(*(str++));
    }
}

// UART0 initialization function
```

```c
void UART0_Init() {
    // Enable clock to UART0 module (bit 0)
    SYSCTL->RCGCUART |= (1 << 0);
    // Enable clock to GPIO Port A (bit 0)
    SYSCTL->RCGCGPIO |= (1 << 0);

    // Enable alternate function on PA0 (RX) and PA1 (TX)
    GPIOA->AFSEL |= U0_RX | U0_TX;
    // Configure PA0 and PA1 pins for UART function in Port Control Register
    // U0_RX corresponds to PA0 (bits 3:0) and U0_TX corresponds to PA1 (bits 7:4)
    GPIOA->PCTL |= (1 << 0) | (1 << 4);
    // Enable digital function for PA0 and PA1 pins
    GPIOA->DEN |= U0_RX | U0_TX;

    // Disable UART0 while configuring
    UART0->CTL = 0;
    // Set integer baud rate divisor for 115200 baud with 50MHz clock
    UART0->IBRD = 27;
    // Set fractional baud rate divisor
    UART0->FBRD = 8;
    // Configure Line Control for 8 data bits, no parity, one stop bit, and FIFOs
    enabled
    // 0x60 = 0b01100000: bit 6 (FEN) = 1 enable FIFO, bits 5-6 (WLEN) = 11 for 8
    bits
    UART0->LCRH = 0x60;
    // Use system clock for UART
    UART0->CC = 0;
    // Enable UART0, TX and RX
    // Bit 0 = UARTEN, bit 8 = TXE, bit 9 = RXE
    UART0->CTL = (1 << 0) | (1 << 8) | (1 << 9);
}

char UART0_ReadChar(void) {
    while (UART0->FR & (1 << 4));  // Wait while RX FIFO empty
    return (char)(UART0->DR & 0xFF);
}

void UART0_ReadString(char *buffer, int maxLen) {
    int i = 0;
    char c;

    while (i < (maxLen - 1)) {   // Leave space for null terminator
        c = UART0_ReadChar();

        // Echo character back (optional)
        UART0_WriteChar(c);

        if (c == '\r' || c == '\n') {  // End of input
      UART0_WriteString("\r\n");
            break;
        }

        buffer[i++] = c;
    }

    buffer[i] = '\0';  // Null terminate the string
```

```
}
```

Listing 9.2: UART Implementation

Echo Server Main Program (`main.c`)

```
#include <stdio.h>
#include "TM4C123.h"
#include "uart.h"

int main(void) {
    UART0_Init();
    UART0_WriteString("Hello World!\r\n");  // Send greeting with newline

    while (1) {
    char buff[16];
    UART0_ReadString(buff, 16);

    UART0_WriteString("Recived: ");
    UART0_WriteString(buff);
    UART0_WriteString("\r\n");
    }
}
```

Listing 9.3: Echo Server Main Program