# Universal Asynchronous Receiver and Transmitter UART

## TM4C123GH6PM Launchpad

Dr. Munesh Singh

Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

February 24, 2020

## Serial Port Programming

- A serial port programming is a method of transferring data serially by the means of a few wire.
- Parallel port which requires many wires for data transfer and limited to a short distance.
- Serial port programming can be used for transferring the data to a larger distance
- The advantage of using this serial port is that it is very cheap as compare to the parallel port
- Serial data communication uses two methods:
  - Asynchronous: transfers a single byte at a time.
  - Synchronous: transfers a block of data (characters) at a time

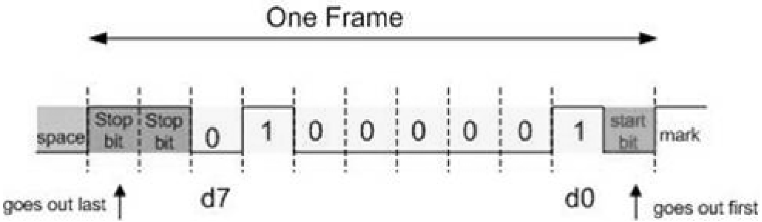
# Asynchronous serial communication

- The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s;
- it is difficult to make sense of the data unless the sender and receiver agree on a set of rules:
  - A protocol
  - How the data is packed
  - How many bits constitute a character
  - When the data begins and ends.

# Start and stop bits

## Asynchronous serial data communication

- Uses character-oriented transmissions.
- Each ASCII characters is packed between **start** and **stop** bits called framing
- The **start bit** is always one bit but the **stop bit** can be one or two bits.
- The **start** bit is always a 0 (low) and the **stop** bit(s) is 1 (high).



# Parity bit

## Asynchronous serial data communication

- In order to maintain data integrity, the parity bit of the character byte is included in the data frame
- For each character of 8-bit, we have a single parity bit in addition to start and stop bits.
- The parity bit may be odd or even:
  - In the case of an odd-parity the number of data bits, including the parity bit, has an odd number of 1s.
  - Similarly, in an even-parity the total number of bits, including the parity bit, is even
- The ASCII character "A", binary 0100 0001, has 0 for the even-parity bit.

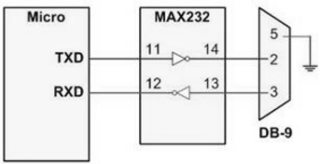
# Data transfer rate

## Asynchronous serial data communication

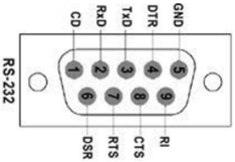
- The rate of data transfer in serial data communication is stated in bps (bits per second).
- Another widely used terminology for bps is Baud rate
- Baud rate is defined as number of signal changes per second.
- Baud rate is defined as the rate of data transfer in serial communication
- Usually the baud rates use 1200, 2400, 4800, 9600,19200, 38400, 57600 and 115200 bps.

## Asynchronous serial data communication

- RS232 was set by the Electronics Industries Association (EIA) in 1960
- RS232 is the most widely used serial I/O interfacing standard
- The table below provides the pins and their labels for the RS232




| Pin | Description |
|---|---|
| 1 | Data carrier detect (DCD) |
| 2 | Received data (RxD) |
| 3 | Transmitted data (TxD) |
| 4 | Data terminal ready (DTR) |
| 5 | Signal ground (GND) |
| 6 | Data set ready (DSR) |
| 7 | Request to send (RTS) |
| 8 | Clear to send (CTS) |
| 9 | Ring indicator (RI) |



# UART on TM4C123GH

- TM4C123GH6PM controllers include eight UART ports (UART0 UART7).
  - Each of these ports features separate 16-by-8 transmit and receive FIFOs
  - A programmable baud rate generator
  - Automatic generation and removal of the start, stop, and parity bits
  - line break generation and detection, a choice of five to eight data bits, multiple parity types, and one or two stop bits, modem and flow control.
- The UART is configured for transmit and/or receive via the TXE and RXE bits of the UART Control (UARTCTL) register.

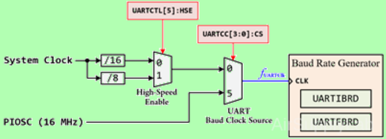
# UART Configuration

## Baud Rate Generation

- Baud rate affects how fast the data can be sent.
- Baud rate is basically the frequency of the TX or RX pulses.
- The baud rate can be evaluated by the following formula:
  $BaudRate = \frac{f_{UARTCLK}}{BRD}$

The UART clock source for **TM4C123G**:

| UARTCC[3:0] CS | UARTCTL[5] HSE | UART Clock Source |
|---|---|---|
| 0 | 0 | $f_{UARTClk} = \frac{f_{SysClk}}{16}$ |
| 0 | 1 | $f_{UARTClk} = \frac{f_{SysClk}}{8}$ |
| 5 | X | $f_{UARTClk} = PIOSC = 16MHz$ |



# UART Configuration

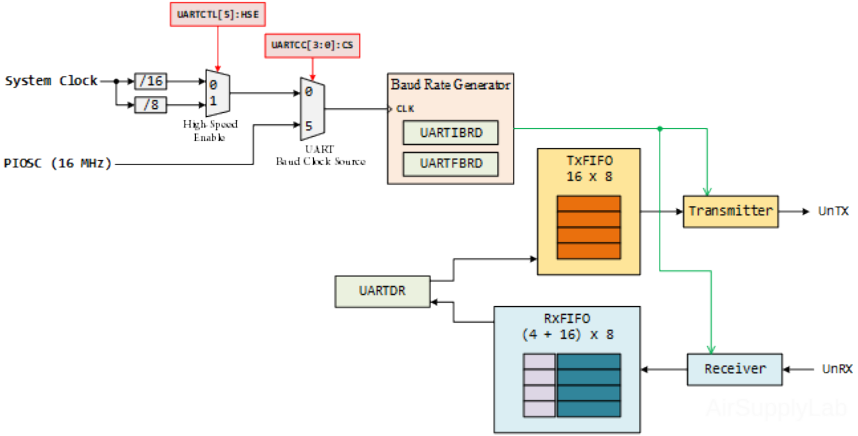
## Baud Rate Generation

- The baud rate generator allowing speeds up to 5Mbps for regular speed (system clock divided by 16)
- 10Mbps for high speed (system clock divided by 8)
- The baud-rate divisor (BRD) is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part.
  - The 16-bit integer is loaded through the UART Integer Baud-Rate Divisor (UARTIBRD) register
  - The 6-bit fractional part is loaded with the UART Fractional Baud-Rate Divisor (UARTFBRD) register.
- The baud-rate divisor has the following relationship to the UART:

$$BRD = \frac{f_{UARTClk}}{BaudRate} = BRD_{Integer} + BRD_{Fraction}$$

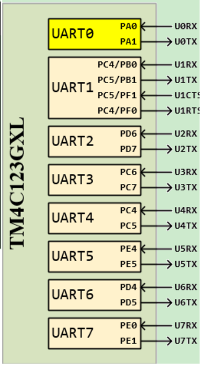
## Baud Rate Generation

- By default, the UART clock source is connected to (System Clock / 16).
- Therefore, the calculation of baud rate divisor will be translated as follows:
  $BRD = \frac{f_{UARTClk}}{16xBaudRate} = BRD_{Integer} + BRD_{Fraction}$
- The UARTIBRD is the integer part of the BRD.
  $UARTIBRD = BRD_{Integer}$
- The UARTFBRD is the integer part of the BRD.
  $UARTFBRD = Integer(BRD_{Fraction}x64 + 0.5)$
- Assume system clock frequency = 8 MHz, and the baud rate which we want the UART to be running at is 19200 bps. Therefore, the value of divisor is:


UARTCTL[5]:HSE
UARTCC[3:0]:CS
System Clock
/16
/8
0
1
High-Speed Enable
0
5
UART Baud Clock Source
PIOSC (16 MHz)
Baud Rate Generator
CLK
UARTIBRD
UARTFBRD
TxFIFO
16 x 8
Transmitter
UnTX
UARTDR
RxFIFO
(4 + 16) x 8
Receiver
UnRX

| UART Peripheral | Function | GPIO Port.Pin (PMCn) | Description |
|---|---|---|---|
| UART0 | U0RX | PA0 (0x1) | UART Module 0 Receive |
| | U0TX | PA1 (0x1) | UART Module 0 Transmit |
| UART1 | U1RX | PC4 (0x2) PB0 (0x1) | UART Module1 Receive |
| | U1TX | PC5 (0x2) PB1 (0x1) | UART Module 1 Transmit |
| | U1CTS | PC5 (0x8) PF1 (0x1) | UART Module1 Clear To Send |
| | U1RTS | PC4 (0x8) PF0 (0x1) | UART Module1 Request To Send |
| UART2 | U2RX | PD6 (0x1) | UART Module 2 Receive |
| | U2TX | PD7 (0x1) | UART Module 2 Transmit |
| UART3 | U3RX | PC6 (0x1) | UART Module 3 Receive |
| | U3TX | PC7 (0x1) | UART Module 3 Transmit |

| UART Peripheral | Function | GPIO Port.Pin (PMCn) | Description |
|---|---|---|---|
| UART4 | U4RX | PC4 (0x1) | UART Module 4 Receive |
| | U4TX | PC5 (0x1) | UART Module 4 Transmit |
| UART5 | U5RX | PE4 (0x1) | UART Module 5 Receive |
| | U5TX | PE5 (0x1) | UART Module 5 Transmit |
| UART6 | U6RX | PD4 (0x1) | UART Module 6 Receive |
| | U6TX | PD5 (0x1) | UART Module 6 Transmit |
| UART7 | U7RX | PE0 (0x1) | UART Module 7 Receive |
| | U7TX | PE1 (0x1) | UART Module 7 Transmit |



# Initialization and Configuration of UART

- TI TM4C123GH6PM micro controller can have up to 8 UART ports
- They are designated as UART0 to UART7.
- The following shows their Base addresses in the memory map
    - UART0 base: 0x4000.C000
    - UART1 base: 0x4000.D000
    - UART2 base: 0x4000.E000
    - UART3 base: 0x4000.F000
    - UART4 base: 0x4001.0000
    - UART5 base: 0x4001.1000
    - UART6 base: 0x4001.2000
    - UART7 base: 0x4001.3000

# Enabling Clock to UART (RCGCUART)

**The RCGCUART register is used to enable the clock to the UART.**

- In this register, there is a bit for each of the UART0 to UART7 modules
- To use UART0, we set to high the D0 of this register.
- Provide clock to PORTA by writing a 1 to RCGCGPIO (SYSCTL_RCGCGPIO_R) register.

Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART)

Base 0x400F.E000
Offset 0x618
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | R7 | R6 | R5 | R4 | R3 | R2 | R1 | R0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# UART Control (UARTCTL)

- Disable the UART0 by writing 0 to UARTCTL (UART0_CTL_R) register of UART0.
- For us the most important bits are RXE, TXE, HSE, and UARTEN.
  - UARTEN (D0) UART enable
  - HSE (D5) High Speed enable:Default Divide-by-16, Divide-by-8 by setting the HSE = 1
  - RXE (D8) Receive enable:We must enable this bit to receive data.
  - TXE (D9) Transmit Enable: We must enable this bit to transmit data.
  - he other bits of this register are used for MODEM signals such as CTS (clear to send), RTS (request to send), parity bit, and so on.

# UART Control (UARTCTL)

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x030
Type RW, reset 0x0000.0300

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CTSEN | RTSEN | reserved | | RTS | reserved | RXE | TXE | LBE | reserved | HSE | EOT | SMART | SIRLP | SIREN | UARTEN |
| Type | RW | RW | RO | RO | RW | RO | RW | RW | RW | RO | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Baudrate Generator (UARTIBRD)

## Two registers are used to set the baud rate:

- They are UART Integer Baud-Rate Divisor (UARTIBRD) and UART Fractional Baud-Rate Divisor (UARTFBRD).

UART Integer Baud-Rate Divisor (UARTIBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x024
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DIVINT | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# UARTFBRD

UART Fractional Baud-Rate Divisor (UARTFBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x028
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | DIVFRAC | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- This is the register we use to configures:
  - To set the number of bits per character (data length).
  - Number of stop bits

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x02C
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | SPS | WLEN | | FEN | STP2 | EPS | PEN | BRK |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# UART Line Control (UARTLCRH)

## Bit Fields

- This is the register has Bit fields:
  - STP2 (D3) stop bit2:The stop bit can be 1 or 2. The default is 1 stop bit at the end of each frame.
  - WLEN (D6 -D5) Word Length: The number of bits per character data in each frame can be 5, 6, 7, or 8.
  - FEN (D4) FIFO enable: 16-byte FIFO (first in first out) buffer to store data for transmission/reception

| D6 | D5 | |
|---|---|---|
| 0 | 0 | 5 bits |
| 0 | 1 | 6 bits |
| 1 | 0 | 7 bits |
| 1 | 1 | 8 bits |

# UARTCTL

- Set UARTEN bit in UARTCTL (UART0_CTL_R) register to enable the UART0.
    - Set TxE and RxE bits in UARTCTL register to enable the transmitter and receiver of UART0.

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x030
Type RW, reset 0x0000.0300

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CTSEN | RTSEN | reserved | | RTS | reserved | RXE | TXE | LBE | reserved | HSE | EOT | SMART | SIRLP | SIREN | UARTEN |
| Type | RW | RW | RO | RO | RW | RO | RW | RW | RW | RO | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Configure the Port A For UART

- Make PA0 and PA1 pins to be used as Digital I/O (GPIO_PORTA_DEN_R).
- Select the alternate functions of PA0 (RxD) and PA1 (TxD) pins using the GPIOAFSEL (GPIO_PORTA_AFSEL_R).
- Configure PA0 and PA1 pins for UART function (GPIO_PORTA_PCTL_R).

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PMC7 (Pin7) | | | | PMC6 (Pin6) | | | | PMC5 (Pin5) | | | | PMC4 (Pin4) | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PMC3 (Pin3) | | | | PMC2 (Pin2) | | | | PMC1 (Pin1) | | | | PMC0 (Pin0) | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

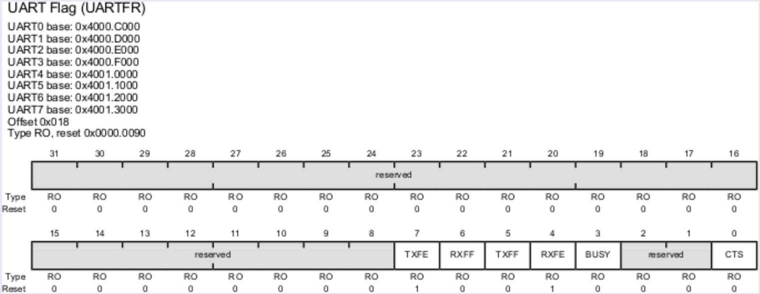
| I/O | Analog Function | Digital Function (GPIO->PCTL PMCn Field Encoding) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 | 0x8 | 0x9 | 0xA | 0xB | 0xC | 0xD | 0xE | 0xF |
| | | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| PA0 | — | GPIO | U0Rx | — | — | — | — | — | — | CAN1Rx | — | — | — | — | — | — | — |
| PA1 | — | GPIO | U0Tx | — | — | — | — | — | — | CAN1Tx | — | — | — | — | — | — | — |
| PA2 | — | GPIO | — | SSI0CLK | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PA3 | — | GPIO | — | SSI0Fss | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PA4 | — | GPIO | — | SSI0Rx | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PA5 | — | GPIO | — | SSI0Tx | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PA6 | — | GPIO | — | — | I2C1SCL | — | M1PWM2 | — | — | — | — | — | — | — | — | — | — |
| PA7 | — | GPIO | — | — | I2C1SDA | — | M1PWM3 | — | — | — | — | — | — | — | — | — | — |
| PB0 | USB0ID | GPIO | U1Rx | — | — | — | — | — | T2CCP0 | — | — | — | — | — | — | — | — |
| PB1 | USB0$V_{BUS}$ | GPIO | U1Tx | — | — | — | — | — | T2CCP1 | — | — | — | — | — | — | — | — |
| PB2 | — | GPIO | — | — | I2C0SCL | — | — | — | T3CCP0 | — | — | — | — | — | — | — | — |
| PB3 | — | GPIO | — | — | I2C0SDA | — | — | — | T3CCP1 | — | — | — | — | — | — | — | — |
| PB4 | AIN10 | GPIO | — | SSI2Clk | — | M0PWM2 | — | — | T1CCP0 | CAN0Rx | — | — | — | — | — | — | — |
| PB5 | AIN11 | GPIO | — | SSI2Fss | — | M0PWM3 | — | — | T1CCP1 | CAN0Tx | — | — | — | — | — | — | — |
| PB6 | — | GPIO | — | SSI2Rx | — | M0PWM0 | — | — | T0CCP0 | — | — | — | — | — | — | — | — |
| PB7 | — | GPIO | — | SSI2Tx | — | M0PWM1 | — | — | T0CCP1 | — | — | — | — | — | — | — | — |
| PC0 | — | GPIO | ***TCK SWCLK*** | — | — | — | — | — | T4CCP0 | — | — | — | — | — | — | — | — |
| PC1 | — | GPIO | ***TMS SWDIO*** | — | — | — | — | — | T4CCP1 | — | — | — | — | — | — | — | — |
| PC2 | — | GPIO | ***TDI*** | — | — | — | — | — | T5CCP0 | — | — | — | — | — | — | — | — |
| PC3 | — | GPIO | ***TDO SWO*** | — | — | — | — | — | T5CCP1 | — | — | — | — | — | — | — | — |
| PC4 | C1- | GPIO | U4Rx | U1Rx | — | M0PWM6 | — | IDX1 | WT0CCP0 | U1RTS | — | — | — | — | — | — | — |
| PC5 | C1+ | GPIO | U4Tx | U1Tx | — | M0PWM7 | — | PhA1 | WT0CCP1 | U1CTS | — | — | — | — | — | — | — |
| PC6 | C0+ | GPIO | U3Rx | — | — | — | — | PhB1 | WT1CCP0 | USB0EPEN | — | — | — | — | — | — | — |
| PC7 | C0- | GPIO | U3Tx | — | — | — | — | — | WT1CCP1 | USB0PFLT | — | — | — | — | — | — | — |
| PD0 | AIN7 | GPIO | SSI3Clk | SSI1Clk | I2C3SCL | M0PWM6 | M1PWM0 | — | WT2CCP0 | — | — | — | — | — | — | — | — |
| PD1 | AIN6 | GPIO | SSI3Fss | SSI1Fss | I2C3SDA | M0PWM7 | M1PWM1 | — | WT2CCP1 | — | — | — | — | — | — | — | — |
| PD2 | AIN5 | GPIO | SSI3Rx | SSI1Rx | — | M0FAULT0 | — | — | WT3CCP0 | USB0EPEN | — | — | — | — | — | — | — |
| PD3 | AIN4 | GPIO | SSI3Tx | SSI1Tx | — | — | — | IDX0 | WT3CCP1 | USB0PFLT | — | — | — | — | — | — | — |
| PD4 | USB0DM | GPIO | U6Rx | — | — | — | — | — | WT4CCP0 | — | — | — | — | — | — | — | — |
| PD5 | USB0DP | GPIO | U6Tx | — | — | — | — | — | WT4CCP1 | — | — | — | — | — | — | — | — |
| PD6 | — | GPIO | U2Rx | — | — | M0FAULT0 | — | PhA0 | WT5CCP0 | — | — | — | — | — | — | — | — |
| PD7 | — | GPIO | U2Tx | — | — | — | — | PhB0 | WT5CCP1 | NMI | — | — | — | — | — | — | — |
| PE0 | AIN3 | GPIO | U7Rx | — | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PE1 | AIN2 | GPIO | U7Tx | — | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PE2 | AIN1 | GPIO | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PE3 | AIN0 | GPIO | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — |
| PE4 | AIN9 | GPIO | U5Rx | — | I2C2SCL | M0PWM4 | M1PWM2 | — | — | CAN0Rx | — | — | — | — | — | — | — |

# Configure the Port A For UART

- Disable analog functionality on PortA0-1 (GPIO_PORTA_AMSEL_R)
- Monitor the RXFE flag bit in UART Flag register (UART0_FR_R)
  - when it goes LOW (buffer not empty)
  - read the received byte from Data register (UART0_DR_R) and save it.
- Monitor the TXFF flag bit in UART Flag register (UART0_FR_R)
  - when it goes LOW (buffer not empty)
  - write received byte to Data register (UART0_DR_R) to be transmitted

# UART Flag Register (UARTFR)

- TXFE (D7) TX FIFO empty:
- RXFF (D6) RX FIFO full
- TXFF (D5) TX FIFO full
- RXFE (D4) RX FIFO empty
- Busy (D3)


UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x018
Type RO, reset 0x0000.0090

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | TXFE | RXFF | TXFF | RXFE | BUSY | reserved | | CTS |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# UART Data (UARTDR)

## UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x000
Type RW, reset 0x0000.0000

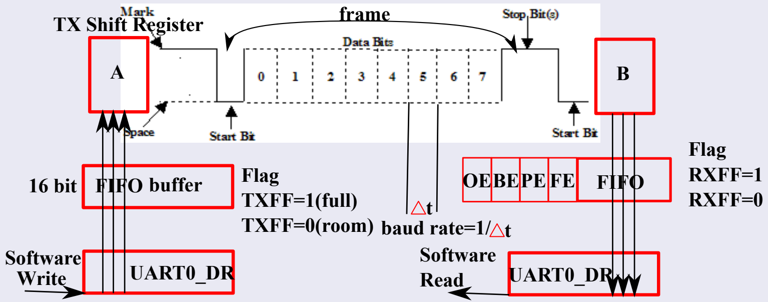
| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | OE | BE | PE | FE | DATA | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

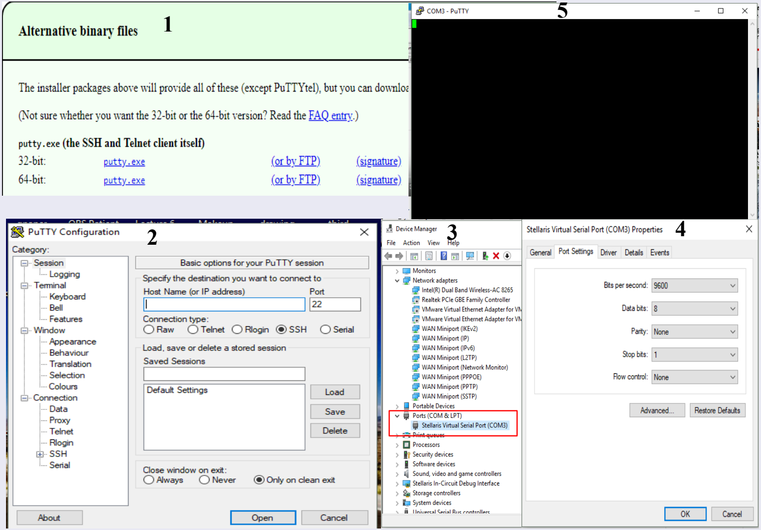
# UART Communication

- Software write the data into the USRT0_DR register
- Data later be transfered to FIFO buffer before it transfer to the TX shift register
- Later the each bit is transfered
- At the receiving end these bits are further transferred to the FIFO buffer.
  - **Overrun Error(OE)** OE, is set when input data are lost because the FIFO is full and more input frames are arriving at the receiver.
  - **Break Error(BE)** The break error, BE, is set when the input is held low for more than a frame.
  - **Parity Error(PE)** PE bit is set on a parity error.
  - **Frame Error(FE)** The framing error, FE, is set when the stop bit is incorrect

# UART Communication



# UART Communication Port Configuration



# UART Programming

```
#include <string.h>
#include <stdlib.h>
#include "tm4c123gh6pm.h"

char readChar(void);
void printChar(char c);
void printString(char * string);
char* readString(char delimiter);

int main()
{
 SYSCTL_RCGCUART_R|= (1<<0); //Enable the UART0 module
 SYSCTL_RCGCGPIO_R |= (1<<0);   //enable clock to PORTA
 GPIO_PORTA_AFSEL_R=(1<<1)|(1<<0); //Enable PA0 and PA1 As Alternate Function PIN
 GPIO_PORTA_PCTL_R|=0x00000001|0x00000010; // make PA0 PA1 UART output pin
 GPIO_PORTA_DEN_R|=(1<<0)|(1<<1); //Enable digital on PA0 PA1
 UART0_CTL_R&= ~(1<<0); //Disable the UART by clearing the UARTEN
 UART0_IBRD_R=104;  // integer portion of the BRD
 UART0_FBRD_R=11;   //fractional portion of the BRD
 UART0_LCRH_R=(0x3<<5)|(1<<4); // 8-bit, no parity, 1-stop bit
 UART0_CTL_R = (1<<0)|(1<<8)|(1<<9); //Enable the UART by setting the UARTEN bit

 SYSCTL_RCGCGPIO_R |= 0x20; /* enable clock to PORTF */
 GPIO_PORTF_DEN_R=(1<<1)|(1<<2)|(1<<3); //Enable digital on PF1 PF2 PF3
 GPIO_PORTF_DIR_R=(1<<1)|(1<<2)|(1<<3); //Enable digital output on PF1 PF2 PF3
 GPIO_PORTF_DATA_R = ~((1<<1)|(1<<2)|(1<<3)); //Disable digital output on PF1 PF2 PF3
 while(1)
  {
```

# UART Programming

```c
while(1)
 {
    printString("Type something and press enter: ");
    char* string = readString('\r');
    printString("\n\r");
    printString("You typed: ");
    printString(string);
    printString("\n\r");
    if(*string=='r')
      GPIO_PORTF_DATA_R = (1<<1)|(1<<2)|(1<<3);
    free(string);
 }
 return 0;
}
```

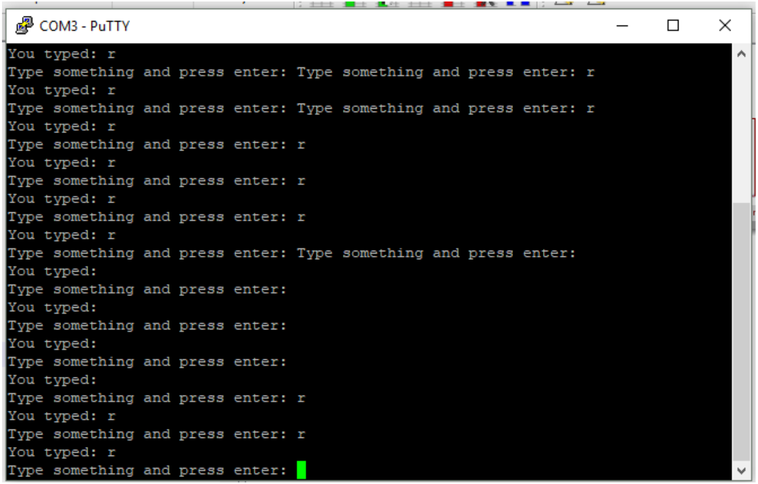
```c
char* readString(char delimiter)
{
  int stringSize = 0;
  char* string = (char*)calloc(10,sizeof(char));
  char c = readChar();
  printChar(c);
  while(c!=delimiter)
  {
    *(string+stringSize) = c;
    stringSize++;
    c = readChar();
    printChar(c); // display the character the user typed
  }
  return string;
}
```

```c
void printString(char * string) //print string function send char by char to FIFO txbuffert o comp
{
 while(*string)
 {
  printChar(*(string++));
 }
}
```

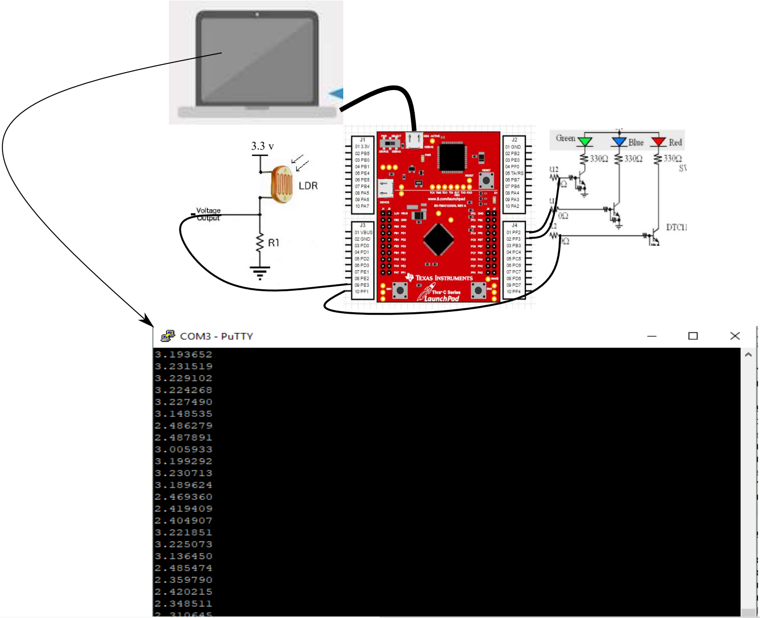
```c
void printChar(char c)  // each char stored in TX FIFO buffer and transfer it computer
{
   while((UART0_FR_R & (1<<5)) != 0);
   UART0_DR_R = c;
}
```

```c
char readChar(void)  // read computer char by char to RX FIFO buffer
{
   char c;
   while((UART0_FR_R & (1<<4)) != 0);
   c = UART0_DR_R;
   return c;
}
```

# UART Output


COM3 - PuTTY
You typed: r
Type something and press enter: Type something and press enter: r
You typed: r
Type something and press enter: Type something and press enter: r
You typed: r
Type something and press enter: r
You typed: r
Type something and press enter: r
You typed: r
Type something and press enter: r
You typed: r
Type something and press enter: Type something and press enter:
You typed:
Type something and press enter:
You typed:
Type something and press enter:
You typed:
Type something and press enter:
You typed:
Type something and press enter: r
You typed: r
Type something and press enter: r
You typed: r
Type something and press enter:

# ADC_UART Control and Visualization



# ADC_UART Programming

```c
#include "tm4c123gh6pm.h"
#include <string.h>
#include <stdlib.h>
void delay();
void printChar(char c);
void printString(char * string);
int main(){
  volatile int result;
  SYSCTL_RCGCADC_R|=1; /* analog clock gating */
  SYSCTL_RCGCUART_R|= (1<<0); //Enable the UART0 module
  SYSCTL_RCGCGPIO_R |= (1<<0)|(1<<4);   //enable clock to PORTA
  GPIO_PORTA_AFSEL_R=(1<<1)|(1<<0); //Enable PA0 and PA1 As Alternate Function PIN
  GPIO_PORTA_PCTL_R|=0x00000001|0x00000010; // make PA0 PA1 UART output pin
  GPIO_PORTA_DEN_R|=(1<<0)|(1<<1); //Enable digital on PA0 PA1
  UART0_CTL_R&= ~(1<<0); //Disable the UART by clearing the UARTEN
  UART0_IBRD_R=104;  // integer portion of the BRD
  UART0_FBRD_R=11;   //fractional portion of the BRD
  UART0_LCRH_R=(0x3<<5)|(1<<4); // 8-bit, no parity, 1-stop bit
  UART0_CTL_R = (1<<0)|(1<<8)|(1<<9); //Enable the UART by setting the UARTEN bit

  GPIO_PORTE_AFSEL_R |= 8;      /* enable alternate function */
  GPIO_PORTE_DIR_R &= ~8;       /* disable digital function */
  GPIO_PORTE_AMSEL_R |= 8;       /* enable analog function */
  ADC0_ACTSS_R &=  ~8;       /* disable SS3 during configuration */
  ADC0_EMUX_R &= ~0xF000;    /* software trigger conversion */
  ADC0_SSMUX3_R = 0;        /* get input from channel 0 */
  ADC0_SSCTL3_R |= 6;       /* take one sample at a time, set flag at 1st sample */
  ADC0_ACTSS_R |= 8;        /* enable ADC0 sequencer 3 */

  SYSCTL_RCGCGPIO_R |= 0x20; /* enable clock to PORTF */
  GPIO_PORTF_DEN_R=(1<<1)|(1<<2)|(1<<3); //Enable digital on PF1 PF2 PF3
  GPIO_PORTF_DIR_R=(1<<1)|(1<<2)|(1<<3); //Enable digital output on PF1 PF2 PF3
  GPIO_PORTF_DATA_R = ~((1<<1)|(1<<2)|(1<<3)); //Disable digital output on PF1 PF2 PF3

while(1)   {
    ADC0_PSSI_R |= 8;       /* start a conversion sequence 3 */
    if((ADC0_RIS_R & 8) == 0)   /* wait for conversion complete */
    {
    result = ADC0_SSFIFO3_R; /* read conversion result */
    char snum[5];
    sprintf(snum, "%f", (result/(4096/3.3)));
    printString(snum);
    if((result/(4096/3.3))>3.2)
      GPIO_PORTF_DATA_R = (1<<1)|(1<<2)|(1<<3);
    else
      GPIO_PORTF_DATA_R = ~((1<<1)|(1<<2)|(1<<3));
    printString("\n\r");
    delay();
    }
   ADC0_ISC_R = 8;        /* clear completion flag */
  }
 return 0;
}
void printString(char * string)
{
  while(*string) {
   printChar(*(string++));
  }
}
void printChar(char c)
{
   while((UART0_FR_R & (1<<5)) != 0);
   UART0_DR_R = c;
}
void delay() {
  int volatile counter=0;
   while(counter<1000000){
     ++counter;
}
 }
```

*Thank You*