Caravel SOC Introduction

Counter， Timer， UART

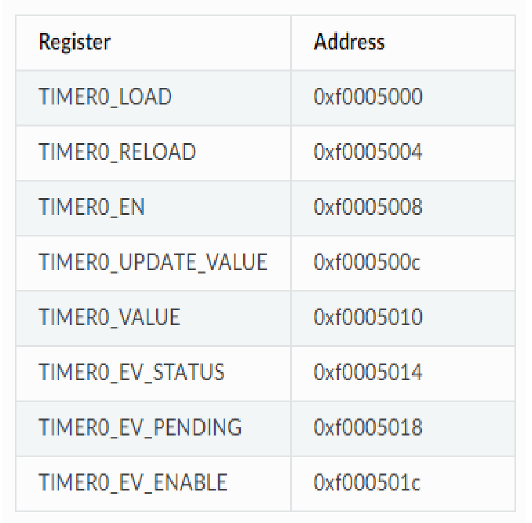
Hurry

Topics

- System Block Diagram (modules)
- Processor VexRISCV functions & its interface
- Reset / POR
- Management Protect Are – power control
- Clocking / DLL, configuration SPI register
- Housekeeping SPI registers
- GPIO
- SPI
- IRQ
- Memory-mapped IO address
- SRAM – Management area/Storage area
- Bus - Wishbone
- Peripherals – Counter/Timer/UART
- User project design (counter) Interface
- Testbench
- Firmware code

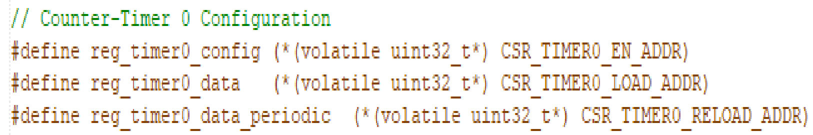
Counter Timer

- The Timer is implemented as a countdown timer that can be used in various modes
  - Polling : Returns current countdown value to software
  - One-Shot: Loads itself and stops when value reaches 0
  - Periodic: Reloads itself when value reaches 0
- Register description
  - LOAD: If using One-Shot mode, then set count value into this register. when Timer is enabled, the value of this register will be loaded to VALUE registers.
  - RELOAD: If using Periodic mode, then set count value into this register. when Timer is enabled, the value of this register will be loaded to VALUE registers and reload value when Timer reaches 0.
  - EN: Set to 1 to enable/start the Timer. Set to 0 to disable the Timer.
  - UPDATE_VALUE: A write to this register latches the current countdown value to value register.
  - VALUE: Latched countdown value. This value is updated by writing to UPDATE_VALUE register.
  - EV_STATUS: This register contains the current raw level of the zero event trigger.
  - EV_PENDING: When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.
  - EV_ENABLE: This register enables the corresponding zero events.



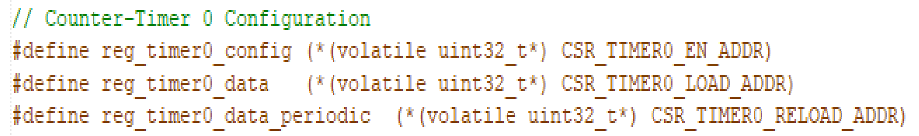
Register	Address
TIMER0_LOAD	0xf0005000
TIMER0_RELOAD	0xf0005004
TIMER0_EN	0xf0005008
TIMER0_UPDATE_VALUE	0xf000500c
TIMER0_VALUE	0xf0005010
TIMER0_EV_STATUS	0xf0005014
TIMER0_EV_PENDING	0xf0005018
TIMER0_EV_ENABLE	0xf000501c

defs.h



```
// Counter-Timer 0 Configuration
#define reg_timer0_config (*(volatile uint32_t*) CSR_TIMER0_EN_ADDR)
#define reg_timer0_data   (*(volatile uint32_t*) CSR_TIMER0_LOAD_ADDR)
#define reg_timer0_data_periodic  (*(volatile uint32_t*) CSR_TIMER0_RELOAD_ADDR)
```

csr.h



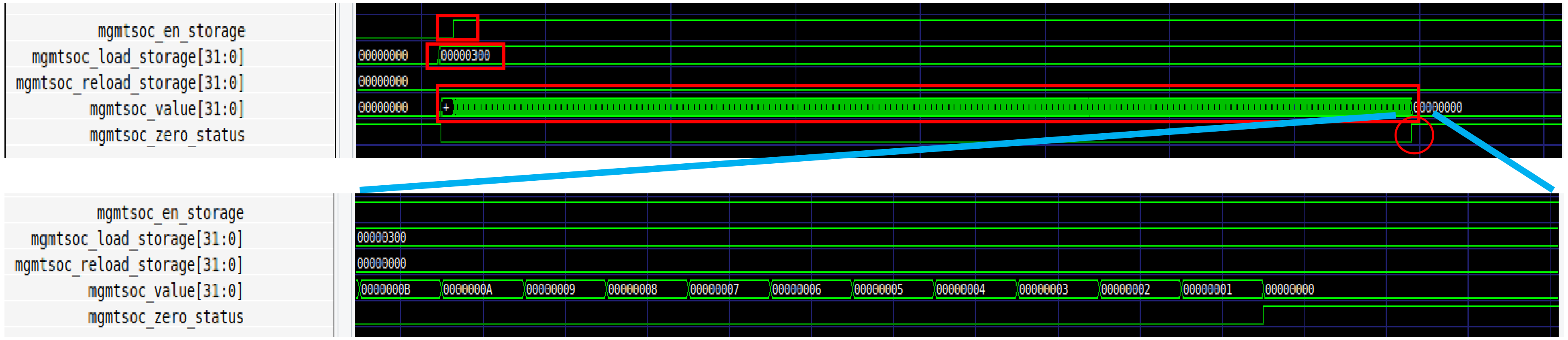
```
// Counter-Timer 0 Configuration
#define reg_timer0_config (*(volatile uint32_t*) CSR_TIMER0_EN_ADDR)
#define reg_timer0_data   (*(volatile uint32_t*) CSR_TIMER0_LOAD_ADDR)
#define reg_timer0_data_periodic  (*(volatile uint32_t*) CSR_TIMER0_RELOAD_ADDR)
```

Counter Timer – Programming Guide

- To use the Timer in One-Shot mode, the user needs to:
    - Disable the timer
    - Set the load register to the expected duration
    - Enable the Timer
- To use the Timer in Periodic mode, the user needs to:
    - Disable the Timer
    - Set the load register to 0
    - Set the reload register to the expected period
    - Enable the Timer
- For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with update_value and value to know the elapsed duration)

Counter Timer – Example

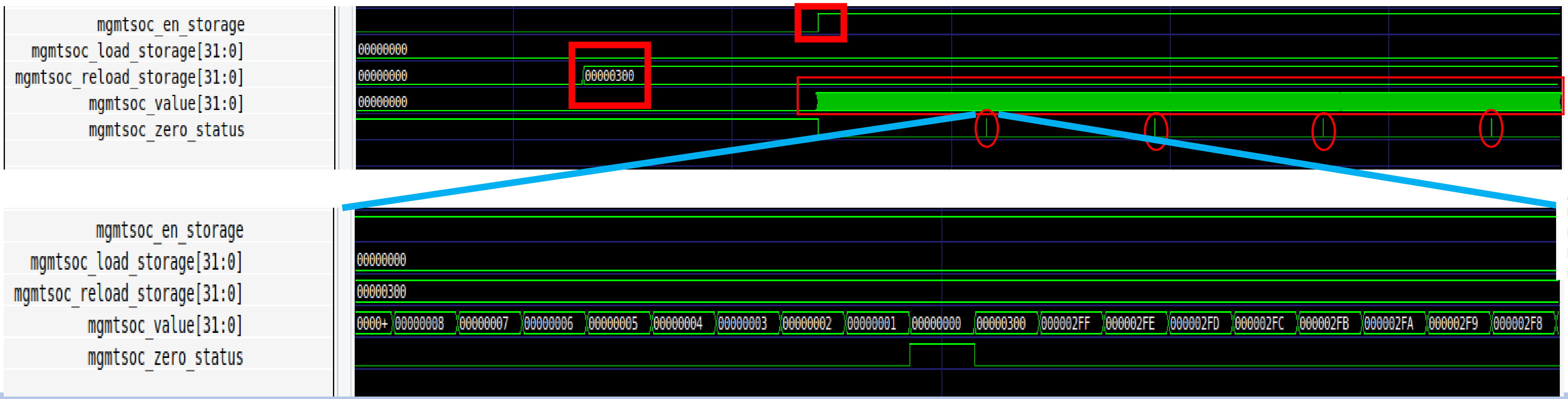
- One-Shot mode: Set the load register to 0x300



Counter_la.c

```
//count timer test for one-shot
reg_timer0_config = 0x00;
reg_timer0_data = 0x300;
reg_timer0_config = 0x01;
```

- Periodic mode: Set the reload register to 0x300

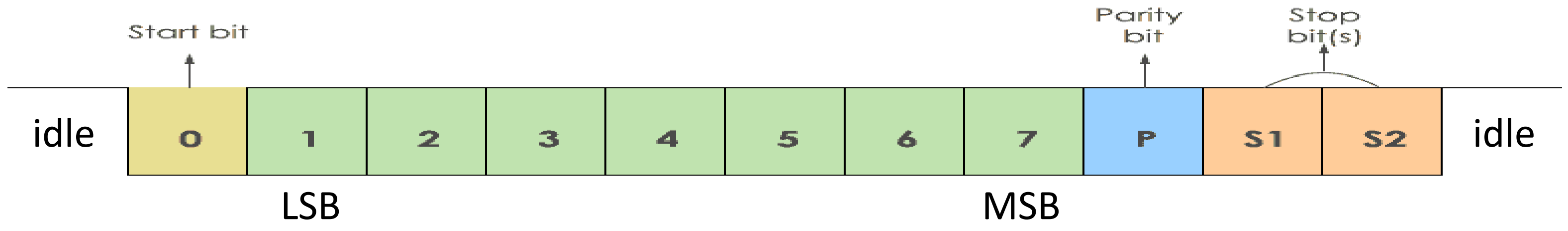


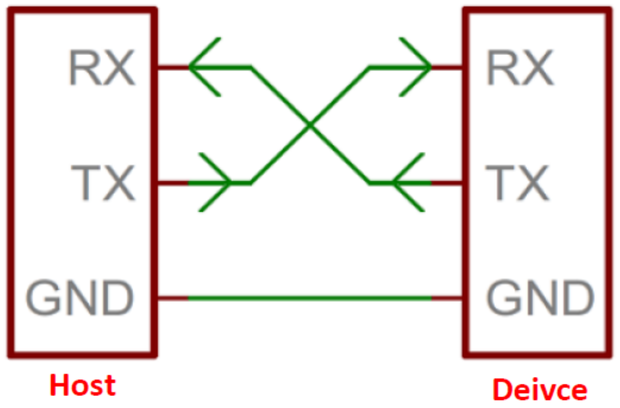
Counter_la.c

```
//count timer test for periodic
reg_timer0_config = 0x00;
reg_timer0_data = 0x0;
reg_timer0_data_periodic = 0x300;
reg_timer0_config = 0x01;
```

UART - Introduction

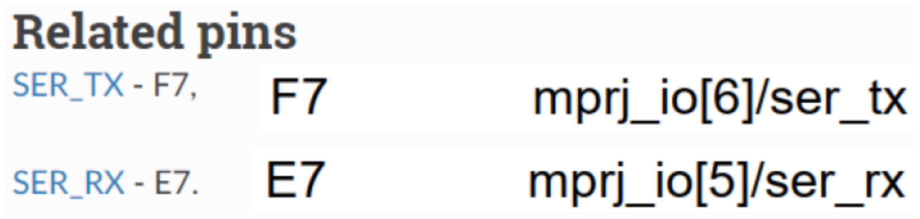
- The UART provide general serial communication with the management SoC. The baud rate is configured at 9600 bps (104.2us per bit).
- Protocol Format





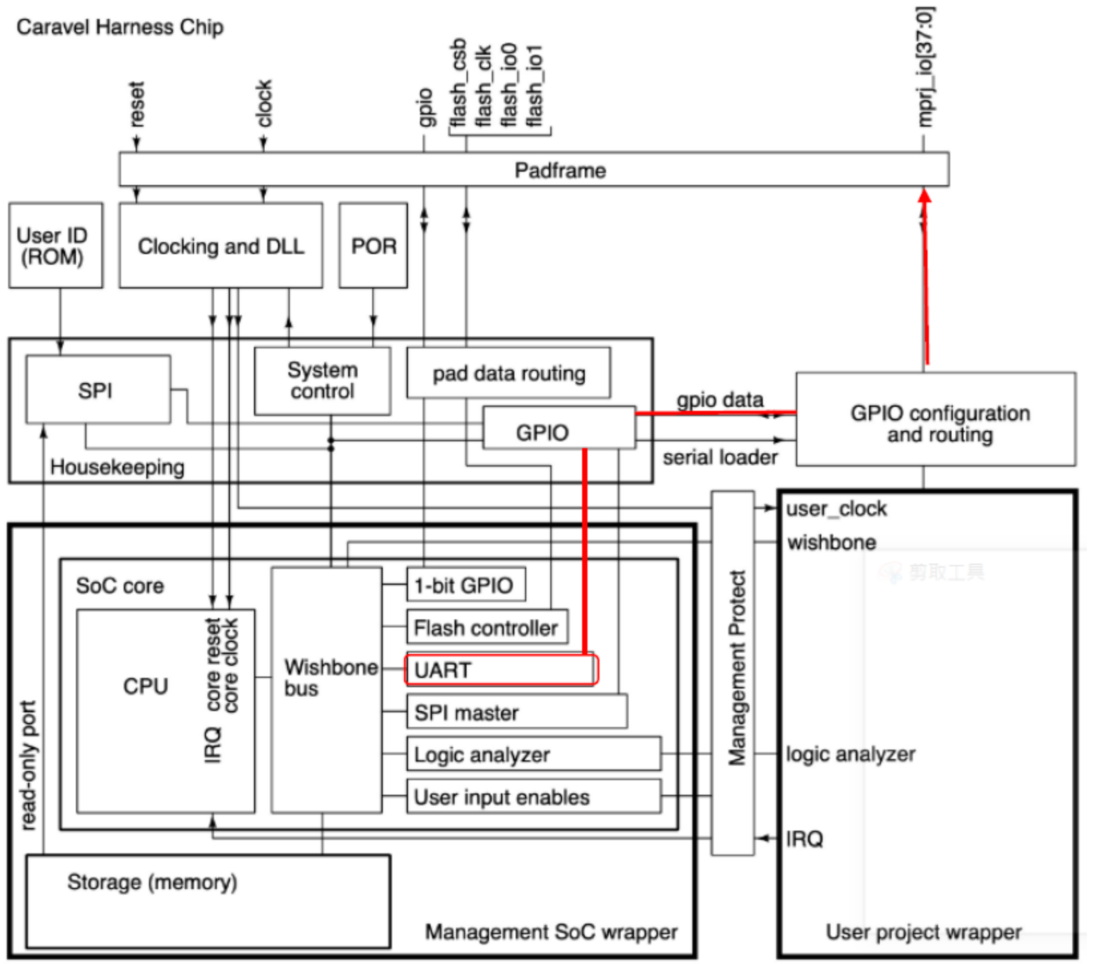
- For UART to work the following settings need to be the same on both the transmitting and receiving side
  - Voltage level
  - Baud Rate
  - Parity bit (0 ~ 1 bit)
  - Data bits size (5 ~ 8 bits)
  - Stop bits size (1 ~ 2 bits)
  - Flow Control
- Caravel
  - Data bits: 8
  - Parity bit: 0
  - Stop bit: 1

Caravel UART ser tx (pin F7) and ser rx (pin E7)



Related pins

SER_TX - F7,	F7	mprj_io[6]/ser_tx
SER_RX - E7.	E7	mprj_io[5]/ser_rx

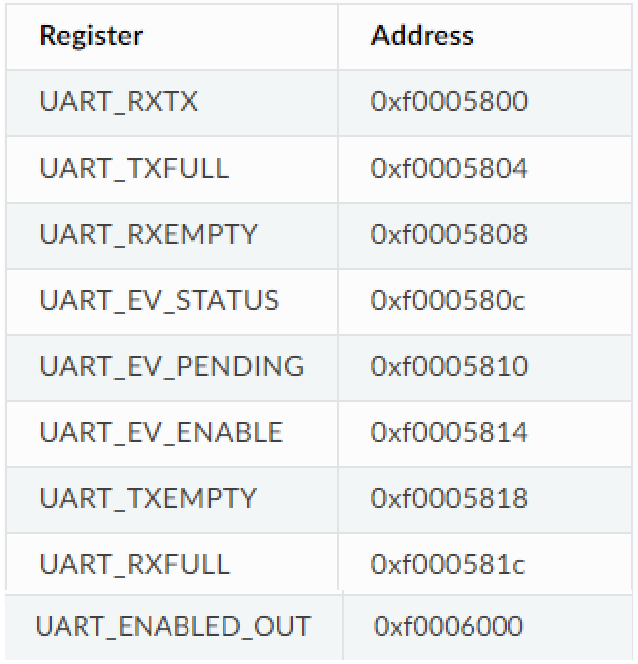


https://caravel harness.readthedocs.io/en/latest/uart.html

UART tx (pin F7) and ser rx (pin E7)

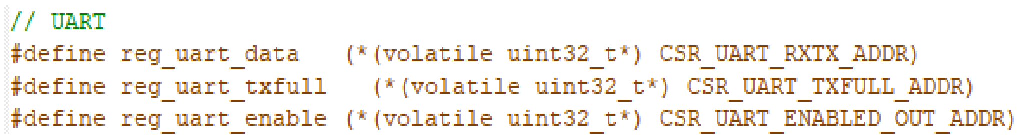
Although the UART operates independently of the CPU, data transfers are blocking operations which will generate CPU wait states until the data transfer is completed.

- baud rate is 9600.
- Transmit data: reg_uart_data = “c”; Writing a value to this register will immediately start a data transfer on the ‘<ser_tx>` pin. If the UART write operation is pending, then the CPU will be blocked with wait states until the transfer is complete before starting the new write operation.
- Receive data: Returns 255 (0xff) if no valid data byte ; or returns the value of the received buffer otherwise, and clears the receive buffer for additional reads..
- Note that there is no FIFO associated with the UART.



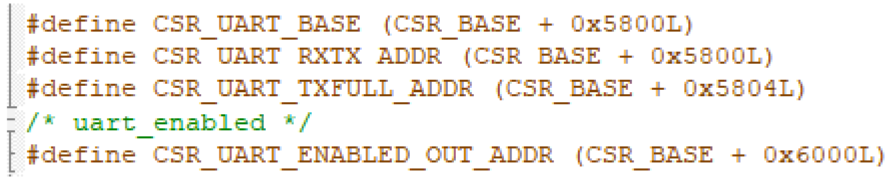
Register	Address
UART_RXTX	0xf0005800
UART_TXFULL	0xf0005804
UART_RXEMPTY	0xf0005808
UART_EV_STATUS	0xf000580c
UART_EV_PENDING	0xf0005810
UART_EV_ENABLE	0xf0005814
UART_TXEMPTY	0xf0005818
UART_RXFULL	0xf000581c
UART_ENABLED_OUT	0xf0006000

defs.h



```
// UART
#define reg_uart_data   (*(volatile uint32_t*) CSR_UART_RXTX_ADDR)
#define reg_uart_txfull   (*(volatile uint32_t*) CSR_UART_TXFULL_ADDR)
#define reg_uart_enable (*(volatile uint32_t*) CSR_UART_ENABLED_OUT_ADDR)
```

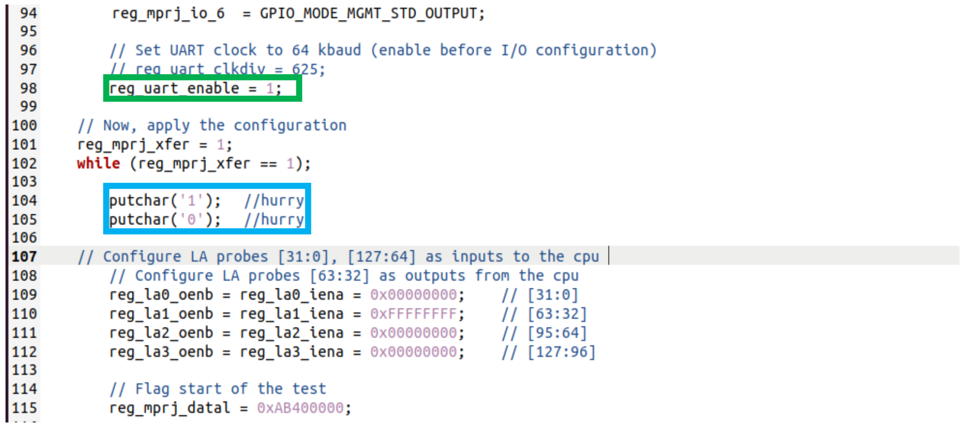
csr.h



```
#define CSR_UART_BASE (CSR_BASE + 0x5800L)
#define CSR_UART_RXTX_ADDR (CSR_BASE + 0x5800L)
#define CSR_UART_TXFULL_ADDR (CSR_BASE + 0x5804L)
/* uart_enabled */
#define CSR_UART_ENABLED_OUT_ADDR (CSR_BASE + 0x6000L)
```

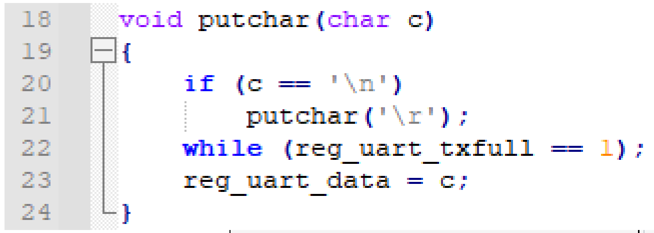
UART-Example

- Counter_la.c



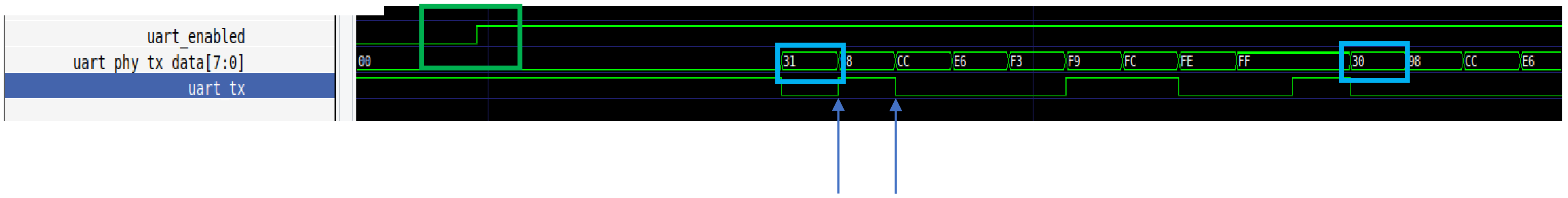
```c
        reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;

        // Set UART clock to 64 kbaud (enable before I/O configuration)
        // reg_uart_clkdiv = 625;
        reg_uart_enable = 1;

    // Now, apply the configuration
    reg_mprj_xfer = 1;
    while (reg_mprj_xfer == 1);

        putchar('1');   //hurry
        putchar('0');   //hurry

    // Configure LA probes [31:0], [127:64] as inputs to the cpu
        // Configure LA probes [63:32] as outputs from the cpu
        reg_la0_oenb = reg_la0_iena = 0x00000000;    // [31:0]
        reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
        reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
        reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]

        // Flag start of the test
        reg_mprj_datal = 0xAB400000;
```

- firmware\stub.c



```c
void putchar(char c)
{
        if (c == '\n')
            putchar('\r');
        while (reg_uart_txfull == 1);
        reg_uart_data = c;
}
```

- Gtkwave



104.2us