Verification of UART using APB BFM

2013 - 2017

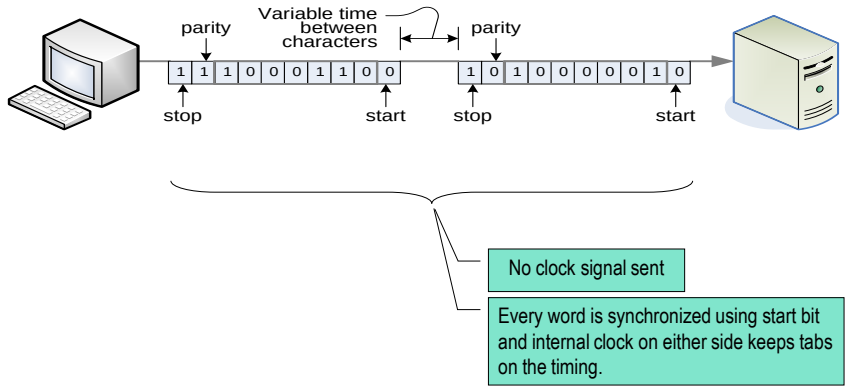
Ando Ki
(adki@future-ds.com)

Agenda

- RS-232C protocol
- RS-232C and UART
- UART and line driver
- Type of UARTS
- OpenCores UART 16550 core
- Frame format
- Baud rate control
- Initialize
- How to transmit a character
- How to receive a character
- UART HW spec.
- How to control HW through SW
- Verification plan
- APB BFM and APB tasks
- TTY model
- Simulation

RS-232C protocol

Asynchronous serial data transmission: RS-232C protocol



No clock signal sent

Every word is synchronized using start bit and internal clock on either side keeps tabs on the timing.

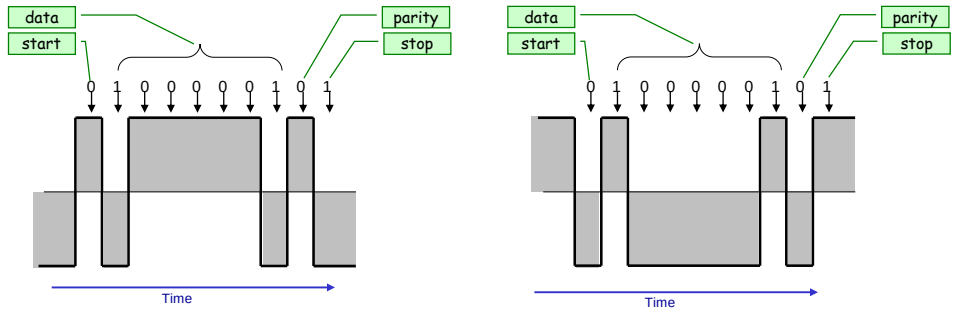
RS-232C: Recommended Standards

RS-232-C and UART (1/2)

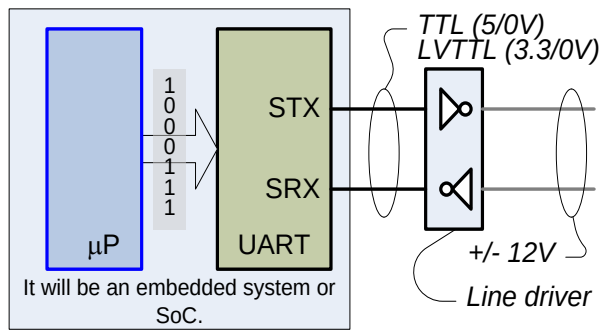
- RS-232-C
  - An EIA standard that defines a commonly used serial communications scheme.
  - It is widely used to transfer data between computers or other devices using an asynchronous serial link at speeds ranging from 110 to 115,200 baud.
  - It uses 25 or 9 pin connector.
  - Signal voltages between +3 to +15 volts are considered ON, Spacing, or Binary 0.
  - Signal voltages between -15 to -3 volts are considered OFF, Marking or Binary 1, which also representing idle state.
  - In the context RS-232-C, the computer is DTE (Data Terminal Equipment) and the modem is DCE (Data Communication Equipment).
- UART (Universal Asynchronous Receiver and Transmitter)
  - a semiconductor chip providing the RS-232-C asynchronous serial communication protocol. One side of UART is an interface to processor and the other side is the serial port.

RS-232-C and UART (2/2)

- RS-232-C uses active low signaling for data.
- RS-232-C uses active high signaling for control.
- UART uses active high signaling for data.
- UART uses active low signaling for control.



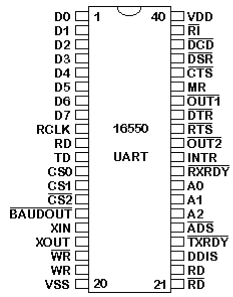
UART and line driver

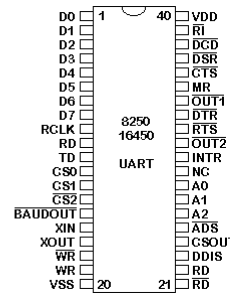


Types of UARTS

Type	remarks
8250	The first UART in this line. It doesn't contain any scratch registers. 8250A is a modernized version of 8250, its bus operating speed is very fast.
8250A	The bus operating speed of this UART is greater than 8250's. It is used in the same way as 16450 in the sphere of software.
8250B	Very similar to that of the 8250 UART.
16450	Used in AT's (Improved bus speed over 8250's). Works stable at 38.4KBPS. Widespread today.
16550	This line is the first generation of buffered UART. This line has 16-byte buffer, however it doesn't work and is replaced with the 16550A.
16550A	This line is the most widespread UART version used for high-speed connection of modems with 14.4KBPS and 28.8KBPS rates. They made sure the FIFO buffers worked on this UART.
16650	New generation of UART. Contains 32 bytes of FIFO, programmed register of X-On/X-Off characters and supports power management.
16750	Produced by Texas Instruments. Contains 64-byte FIFO buffer.

16550 and 8250





OpenCores UART 16550 core

- http://www.opencores.org/projects.cgi/web/uart16550/overview
- The UART (Universal Asynchronous Receiver/Transmitter) core provides serial communication capabilities, which allow communication with modem or other external devices, like another computer using a serial cable and RS232 protocol. This core is designed to be maximally compatible with the industry standard National Semiconductors' 16550A device.
- Features
  - WISHBONE interface in 32-bit or 8-bit data bus modes (selectable)
  - FIFO only operation
  - Register level and functionality compatibility with NS16550A (but not 16450).
  - Debug Interface in 32-bit data bus mode.
- APB interface has been adopted for this project

OpenCores UART CSR (1/2)

Name		Addr	W	Access	Description
Receiver Buffer	RB	0	8	R	Receiver FIFO output
Transmitter Holding Register	THR	0	8	W	Transmit FIFO input
Interrupt Enable	IER	1	8	RW	Enable/Mask interrupts generated by the UART
Interrupt Identification	IIR	2	8	R	Get interrupt information
FIFO Control	FCR	2	8	W	Control FIFO options
Line Control Register	LCR	3	8	RW	Control connection
Modem Control	MCR	4	8	W	Controls modem
Line Status	LSR	5	8	R	Status information
Modem Status	MSR	6	8	R	Modem Status

OpenCores UART CSR (2/2)

Name		Addr	W	Access	Description
Divisor Latch Byte 1 (LSB)	CDRl	0	8	RW	The LSB of the divisor latch
Divisor Latch Byte 2	CDRh	1	8	RW	The MSB of the divisor latch

Two clock divisor registers (CDR) together forming one 16-bit.
The CDR is accessed when 7th (DLAB) bit of LCR is set to 1.

IER: interrupt enable register

Bit #	Access	Description
0	RW	Received Data available interrupt '0' – disabled '1' – enabled
1	RW	Transmitter Holding Register empty interrupt '0' – disabled '1' – enabled
2	RW	Receiver Line Status Interrupt '0' – disabled '1' – enabled
3	RW	Modem Status Interrupt '0' – disabled '1' – enabled
7-4	RW	Reserved. Should be logic '0'.

Reset value: 00h

IIR: interrupt identification register

bit			pri	Interrupt Type	Interrupt Source	Interrupt Reset Control
3	2	1				
0	1	1	1	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register
0	1	0	2	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level
1	1	0	2	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 Char times.	Reading from the FIFO (Receiver Buffer Register)
0	0	1	3	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading IIR.
0	0	0	4	Modem Status	CTS, DSR, RI or DCD.	Reading the Modem status register.

Reset value: C1h

FCR: FIFO control register

Bit #	Access	Description
0	W	Ignored (Used to enable FIFOs in NS16550D). Since this UART only supports FIFO mode, this bit is ignored.
1	W	Writing a '1' to bit 1 clears the Receiver FIFO and resets its logic. But it doesn't clear the shift register, i.e. receiving of the current character continues.
2	W	Writing a '1' to bit 2 clears the Transmitter FIFO and resets its logic. The shift register is not cleared, i.e. transmitting of the current character continues.
5-3	W	Ignored
7-6	W	Define the Receiver FIFO Interrupt trigger level '00' – 1 byte '01' – 4 bytes '10' – 8 bytes '11' – 14 bytes

Reset value: C0h

LCR: line control register (1/2)

Bit #	Access	Description
1-0	RW	Select number of bits in each character ‘00’ – 5 bits; ‘01’ – 6 bits; ‘10’ – 7 bits; ‘11’ – 8 bits
2	RW	Specify the number of generated stop bits ‘0’ – 1 stop bit ‘1’ – 1.5 stop bits when 5-bit character length selected and 2 bits otherwise Note that the receiver always checks the first stop bit only.
3	RW	Parity Enable ‘0’ – No parity ‘1’ – Parity bit is generated on each outgoing character and is checked on each incoming one.
4	RW	Even Parity select ‘0’ – Odd number of ‘1’ is transmitted and checked in each word (data and parity combined). In other words, if the data has an even number of ‘1’ in it, then the parity bit is ‘1’. ‘1’ – Even number of ‘1’ is transmitted in each word.

LCR: line control register (1/2)

Bit #	Access	Description
5	RW	Stick Parity bit. ‘0’ – Stick Parity disabled ‘1’ - If bits 3 and 4 are logic ‘1’, the parity bit is transmitted and checked as logic ‘0’. If bit 3 is ‘1’ and bit 4 is ‘0’ then the parity bit is transmitted and checked as ‘1’.
6	RW	Break Control bit ‘1’ – the serial out is forced into logic ‘0’ (break state). ‘0’ – break is disabled
7	RW	Divisor Latch Access bit. (DLAB) ‘1’ – The divisor latches can be accessed ‘0’ – The normal registers are accessed

Reset value: 03h

LSR: line status register (1/3)

Bit #	Access	Description
0	R	Data Ready (DR) indicator. '0' – No characters in the FIFO '1' – At least one character has been received and is in the FIFO.
1	R	Overrun Error (OE) indicator '1' – If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No overrun state
2	R	Parity Error (PE) indicator '1' – The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No parity error in the current character

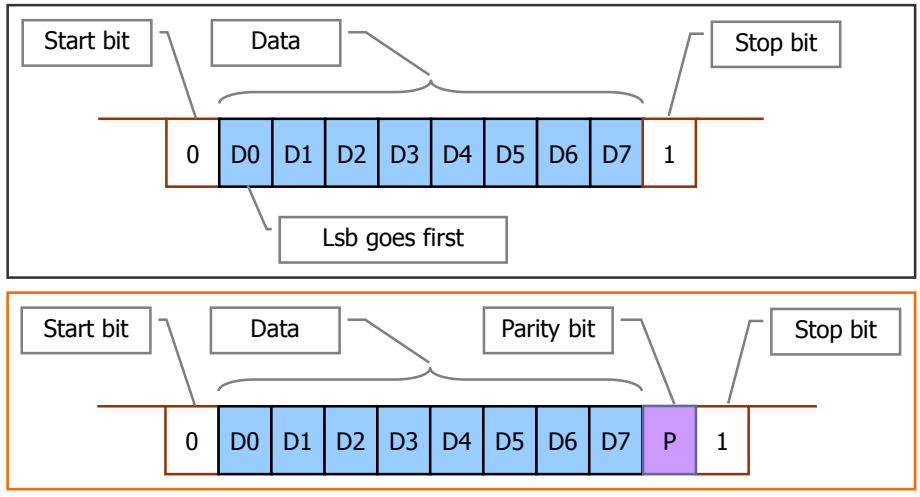
LSR: line status register (2/3)

Bit #	Access	Description
3	R	Framing Error (FE) indicator '1' – The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No framing error in the current character
4	R	Break Interrupt (BI) indicator '1' –A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '0' – No break condition in the current character

LSR: line status register (3/3)

Bit #	Access	Description
5	R	Transmit FIFO is empty. '1' – The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared when data is being been written to the transmitter FIFO. '0' – Otherwise
6	R	Transmitter Empty indicator. '1' – Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared when data is being been written to the transmitter FIFO. '0' – Otherwise
7	R	'1' – At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. '0' – Otherwise.

Frame format



Baud rate control

$$\frac{F_I}{DR} = 16 \times BR$$

where F_I is input clock speed, DR is divisor latch value

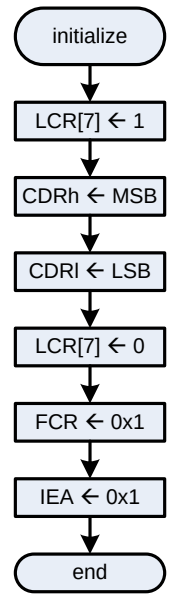
BR is baud rate.

$$DR = \frac{F_I}{16 \times BR} + 0.5$$

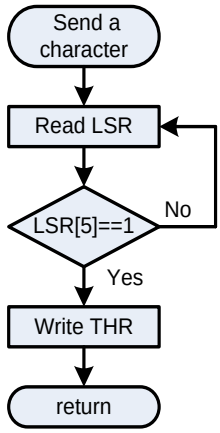
Calculate the value of divisor latches (DL[15:8] and DL[7:0]) for 9,600 baud rate with 33MHz input clock.

Initialize

- Upon reset the followings are done by hardware (UART core)
  - The receiver and transmitter FIFOs are cleared including shift registers.
  - The divisor latch register is set to 0.
  - The line control register is set to 8-bit data, no parity, 1 stop bit.
  - All interrupt are disabled in the interrupt enable register.
- Perform the following during UART initialization phase
  - Set bit 7 of LCR to 1 to access divisor latches.
  - Set the divisor latches, MSB first, LSB next.
    - See baud rate control
  - Set bit 7 of LCR to 0.
  - Set the FIFO trigger level of FCR.
  - Enable desired interrupt by setting IER.

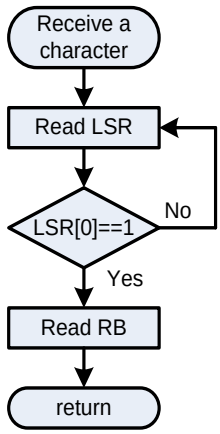


How to transmit a character



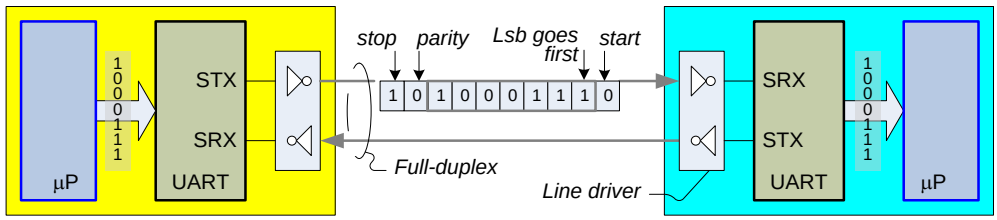

```
unsigned int
uart_put_char(char d) {
  while (!(_UART->LSR&0x20));
  // wait until transmitter FIFO is empty
  _UART->RB_THR = d;
  return (unsigned int)d;
}
```


How to receive a character




```
unsigned int
uart_get_char(void) {
  while (!(_UART->LSR&0x1));
  // wait until a character has been received
  return (unsigned int)_UART->RB_THR;
}
```

UART hardware specification

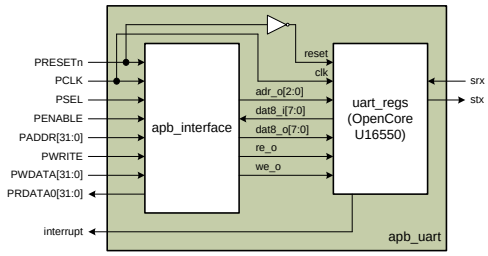


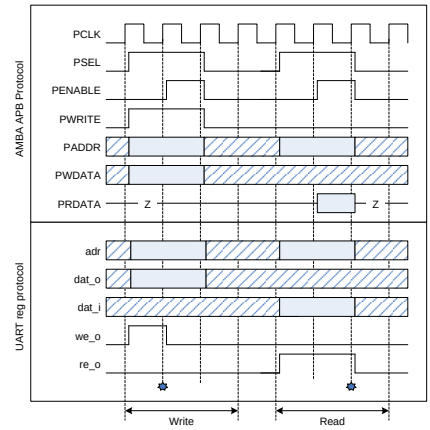
- UART (Universal Asynchronous Receiver and Transmit) is a semiconductor chip
  - It implements RS-232-C asynchronous serial communication protocol.
  - One side of it is a parallel interface to the processor and the other side is the RS-232-C serial port.
  - UART has CSR (Control and Status Register)
    - The right-hand side shows UART16550.

CSR name		Add	W	Access
Receiver Buffer	RB	0	8	R
Transmitter Holding	THR	0	8	W
Interrupt Enable	IER	1	8	RW
Interrupt Identification	IIR	2	8	R
FIFO Control	FCR	2	8	W
Line Control Register	LCR	3	8	RW
Modem Control	MCR	4	8	W
Line Status	LSR	5	8	R
Modem Status	MSR	6	8	R

UART hardware specification

UART16550 hardware specification



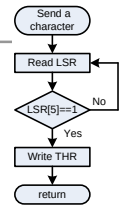


How to control HW through SW

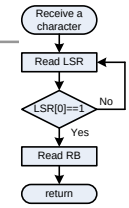
```
static struct uart16550 {
  unsigned char RB_THR;
  unsigned char IER;
  unsigned char IIR_FCR;
  unsigned char LCR;
  unsigned char MCR;
  unsigned char LSR;
  unsigned char MSR;
} *_UART;
```

```
void
uart_init(void* UartStart) {
    extern void uart_set_baud(unsigned int);
    _UART = (struct uart16550*)UartStart;
    uart_set_baud(19200);
    _UART->IIR_FCR = 0x01;
    _UART->IER     = 0x01;
}
```

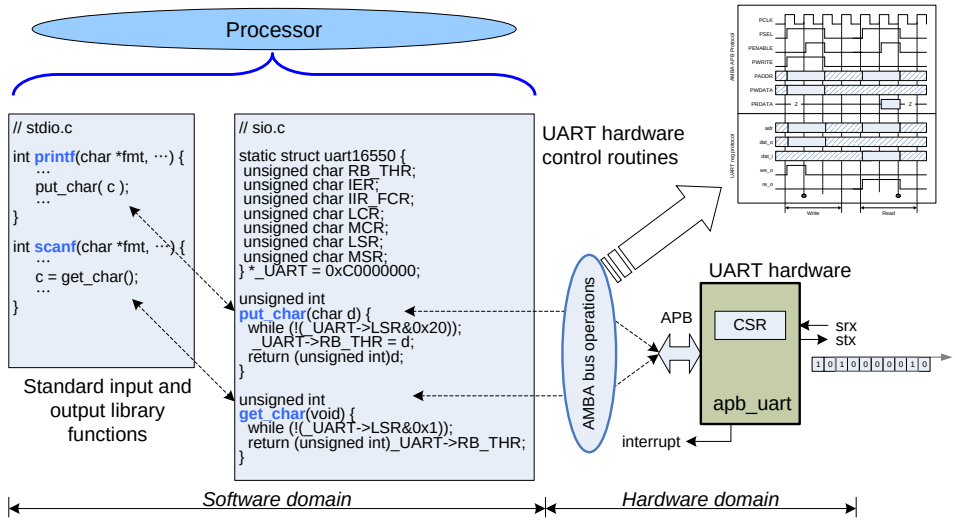
```
unsigned int
put_char(char d) {
  while (!(_UART->LSR&0x20));
   // wait until transmitter FIFO is empty
  _UART->RB_THR = d;
  return (unsigned int)d;
}
```




```
unsigned int
get_char(void) {
  while (!(_UART->LSR&0x1));
  // wait until a character has been received
  return (unsigned int)_UART->RB_THR;
}
```

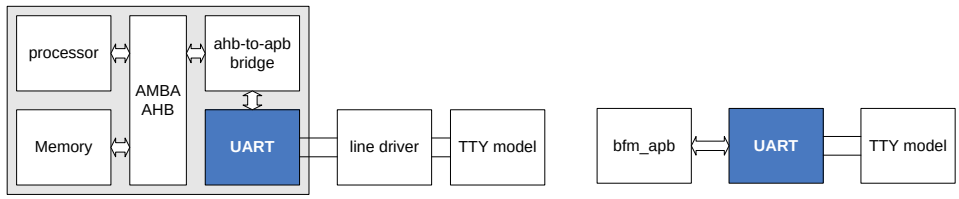


How to control HW through SW



Verification plan

- The best way
- A novel way



APB BFM: module

```
`timescale 1ns/1ns

module bfm_apb_s1
    #(parameter P_ADDR_START0 = 16'h0000, P_ADDR_SIZE0 = 16'h0010)
(
     input  wire        PRESETn
    , input  wire        PCLK
    , output reg         PSEL
    , output reg  [31:0] PADDR
    , output reg         PENABLE
    , output reg         PWRITE
    , output reg  [31:0] PWDATA
    , input  wire [31:0] PRDATA0
);
   reg [31:0] freq;
   real stamp_x, stamp_y, delta;
   initial begin
      PSEL    =  1'b0;
      PADDR   = ~32'h0;
      PENABLE =  1'b0;
      PWRITE  =  1'b0;
      PWDATA  = ~32'h0;
      PPROT   =  3'h0;
      PSTRB   =  4'h0;
      wait (PRESETn==1'b0);
      wait (PRESETn==1'b1);
      @ (posedge PCLK);
      @ (posedge PCLK); stamp_x = $time;
      @ (posedge PCLK); stamp_y = $time; delta = stamp_y - stamp_x;
      @ (negedge PCLK); $display("%m PCLK %f nsec %f Mhz", delta, 1000.0/delta);
      freq = 1000000000/delta;
      repeat (3) @ (posedge PCLK);
      uart_test(freq, 115200);
      repeat (5) @ (posedge PCLK);
      $finish(2);
   end
```

Calculate frequency

Call UART testing scenario

APB BFM: module

```
integer  err;
//-------------------------------------------------------
task uart_test;
    input [31:0] freq;
    input [31:0] baud;
    reg [7:0] dat;
    integer idx;
begin
    err = 0;
    //-------------------------------------------------------
    init_uart( freq // input [31:0] frea;
             , baud // input [31:0] baud
    );
    //-------------------------------------------------------
    for (idx="A"; idx<="Z"; idx = idx + 1) begin
        send_a_character(idx[7:0]);
        receive_a_character(dat);
        if (dat>=8'h20&&dat<=8'h7E)
            $display($time,,"%m 0x%x(%c) received!", dat, dat);
        else
            $display($time,,"%m 0x%x  received!", dat);
        if (idx[7:0]!==dat) begin
          err = err + 1;
          $display($time,,"%m ERROR 0x%x received, but 0x%x expected",
                   dat, idx[7:0]);
        end
    end
    //-------------------------------------------------------
    if (err==0) $display($time,,"%m test OK");
end
endtask
//-------------------------------------------------------
`include "bfm_apb_tasks_s1.v"
`include "uart_apb_tasks.v"
//-------------------------------------------------------
endmodule
```

initialize UART

Send a character

Receive a character

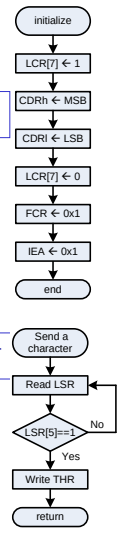
APB bus tasks

UART handling tasks

UART APB tasks

```
// U16550 CSR address
localparam RB_THR    =0
         , IER       =4
         , IIR_FCR   =8
         , LCR       =12
         , MCR       =16
         , LSR       =20
         , MSR       =24;
//-------------------------------------------------------
task init_uart;
    input [31:0] freq;
    input [31:0] baud;
    reg   [31:0] dl;
begin
    dl = (freq/(16*baud))+0.5;
    apb_write(LCR,    32'h83, 4);
    apb_write(IER,    {24'h0,dl[15:8]}, 4);
    apb_write(RB_THR, {24'h0,dl[ 7:0]}, 4);
    apb_write(LCR,    32'h03, 4);
    apb_write(IIR_FCR,32'h01, 4);
    apb_write(IER,    32'h01, 4);
end
endtask
//-------------------------------------------------------
task send_a_character;
    input [ 7:0] dat;
    reg   [31:0] tmp;
begin
     apb_read(LSR, tmp, 4);
     // wait until buffer empty
     while (!(tmp&32'h20)) apb_read(LSR, tmp, 4);
     apb_write(0, {24'h0,dat}, 4);
end
endtask
//-------------------------------------------------------
```

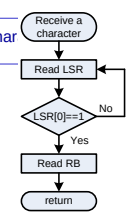
initialize UART



Send a character


```
//-------------------------------------------------------
task receive_a_character;
    output [ 7:0] dat;
    reg    [31:0] tmp;
begin
     apb_read(LSR, tmp, 4);
     // wait until a character recived
     while (!(tmp&32'h01)) apb_read(LSR, tmp, 4);
     apb_read(RB_THR, tmp, 4);
     dat = tmp[7:0];
end
endtask
//-------------------------------------------------------
```

Receive a char



BFM APB tasks

```
task apb_write;
input [31:0] addr;
input [31:0] data;
input [ 2:0] size;
begin
    @ (posedge PCLK);
    PADDR   <= #1 addr;
    PWRITE  <= #1 1'b1;
    PSEL    <= #1 1'b1; //decoder(addr);
    PWDATA  <= #1 data;
    PSTRB   <= #1 get_pstrob(addr,size);
    @ (posedge PCLK);
    PENABLE <= #1 1'b1;
    @ (posedge PCLK);
    while (get_pready(addr)==1'b0) @ (posedge PCLK);
    `ifndef LOW_POWER
    PADDR   <= #1 32'h0;
    PWRITE  <= #1 1'b0;
    PWDATA  <= #1 ~32'h0;
    `endif
    PSEL    <= #1 1'b0;
    PENABLE <= #1 1'b0;
    if (get_pslverr(addr)==1'b1) $display($time,,"%m PSLVERR");
end
endtask
//-------------------------------------------------------
```

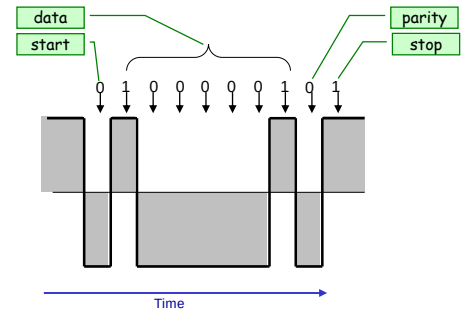
```
//-------------------------------------------------------
task apb_read;
input  [31:0] addr;
output [31:0] data;
input  [ 2:0] size;
begin
    @ (posedge PCLK);
    PADDR   <= #1 addr;
    PWRITE  <= #1 1'b0;
    PSEL    <= #1 1'b1; //decoder(addr);
    PSTRB   <= #1 4'hF;
    @ (posedge PCLK);
    PENABLE <= #1 1'b1;
    @ (posedge PCLK);
    while (get_pready(addr)==1'b0) @ (posedge PCLK);
    `ifndef LOW_POWER
    PADDR   <= #1 32'h0;
    `endif
    PSEL    <= #1 1'b0;
    PENABLE <= #1 1'b0;
    if (get_pslverr(addr)==1'b1) $display($time,,"%m PSLVERR");
    data    = get_prdata(addr); // it should be blocking
end
endtask
//-------------------------------------------------------
... ..
```


TTY model

```
`timescale 1ns/1ns

module tty #(parameter BAUD_RATE  = 115200, LOOPBACK=1)
(
  output  reg   STX,
  input   wire  SRX
);
  //-----------------------------------------------
  localparam INTERVAL = (1000000000/BAUD_RATE); // nsec
  //-----------------------------------------------
  reg [7:0] data  = 0;
  //-----------------------------------------------
  initial begin STX = 1'b1; end
  //-----------------------------------------------
  always @ (negedge SRX) begin
      receive(data);
      $write("%c", data); $fflush();
      if (LOOPBACK) send(data);
  end
  //-----------------------------------------------
  task receive;
      output [7:0] value;
      integer      x;
  begin
        value = 0;
        #(INTERVAL*1.5);
        for (x=0; x<8; x=x+1) begin // LSB comes first
              value[x] = SRX;
              #(INTERVAL);
        end
  end
  endtask
```

```
//-----------------------------------------------
  task send;
      input [7:0] value;
      integer     y;
  begin
      STX = 1'b0;
      #(INTERVAL);
      for (y=0; y<8; y=y+1) begin // LSB goes first
        STX = value[y];
        #(INTERVAL);
      end
      STX = 1'b1;
      #(INTERVAL);
  end
  endtask
  //-----------------------------------------------
endmodule
```



Test-bench

```
`timescale 1ns/1ns

`ifndef CLK_FREQ
`define CLK_FREQ      50000000
`endif

module top ;
  //-----------------------------------------------------
  reg             PRESETn    = 1'b0;
  reg             PCLK       = 1'b0;
  wire            PSEL   ;
  ... ...
  //-----------------------------------------------------
  bfm_apb_s1
  u_bfm_apb_s1 (
       .PRESETn  (PRESETn)
     , .PCLK     (PCLK   )
     , .PSEL     (PSEL   )
     , .PADDR    (PADDR  )
     , .PENABLE  (PENABLE)
     , .PWRITE   (PWRITE )
     , .PWDATA   (PWDATA )
     , .PRDATA0  (PRDATA )
    `ifdef AMBA3
     , .PREADY   (PREADY )
     , .PSLAVERR (PSLAVERR)
    `endif
    `ifdef AMBA4
     , .PPROT    (PPROT)
     , .PSTRB    (PSTRB)
    `endif
  );
```

```
//-----------------------------------------------------
  wire srx, stx;
  uart_apb u_uart_apb (
       .PRESETn  (PRESETn  )
     , .PCLK     (PCLK     )
     , .PSEL     (PSEL     )
     , .PENABLE  (PENABLE  )
     , .PADDR    (PADDR    )
     , .PWRITE   (PWRITE   )
     , .PRDATA   (PRDATA   )
     , .PWDATA   (PWDATA   )
     , .interrupt (        ) // interrupt request (active-high)
     , .srx      (srx      ) // serial output
     , .stx      (stx      ) // serial input
  );
  //-----------------------------------------------------
  tty #(.BAUD_RATE(115200), .LOOPBACK(1))
  u_tty (
       .STX (srx)
     , .SRX (stx)
  );
  //-----------------------------------------------------
  localparam CLK_FREQ=`CLK_FREQ;
  localparam CLK_PERIOD_HALF=1000000000/(CLK_FREQ*2);
  //-----------------------------------------------------
  always #CLK_PERIOD_HALF PCLK <= ~PCLK;
  //-----------------------------------------------------
  ... ...
endmodule
```

Simulation with ModelSim (1/4)

```
# Makefile
SHELL                   = /bin/sh
MAKEFILE        = Makefile
#-----------------------------------------------------
VLIB            = $(shell which vlib)
VLOG            = $(shell which vlog)
VSIM            = $(shell which vsim)
WORK            = work
#-----------------------------------------------------
TOP             = top
#-----------------------------------------------------
all: vlib compile simulate

vlib:
                if [ -d $(WORK) ]; then /bin/rm -rf $(WORK); fi
                $(VLIB) $(WORK)

compile:
                $(VLOG) -lint -work $(WORK)  -f modelsim.args

simulate: compile
                $(VSIM) -novopt -c -do "run -all; quit" $(WORK).$(TOP)
```

Modelsim commands

Specify where to store compile results

Compilation

Simulation

```
@ECHO OFF
REM RunMe.bat
SET MODELSIMWORK=work
SET MODELSIMVLIB=vlib
SET MODELSIMVSIM=vsim
SET MODELSIMVCOM=vcom
SET MODELSIMVLOG=vlog

SET DESIGNTOP=top

IF EXIST %MODELSIMWORK% RMDIR /S/Q %MODELSIMWORK%

%MODELSIMVLIB% %MODELSIMWORK%
%MODELSIMVLOG% -work %MODELSIMWORK% -lint^
                -f modelsim.args
%MODELSIMVSIM% -novopt -c -do "run -all; quit"^
            %MODELSIMWORK%.%DESIGNTOP%
```

Simulation with ModelSim (2/4)

modelsim.args

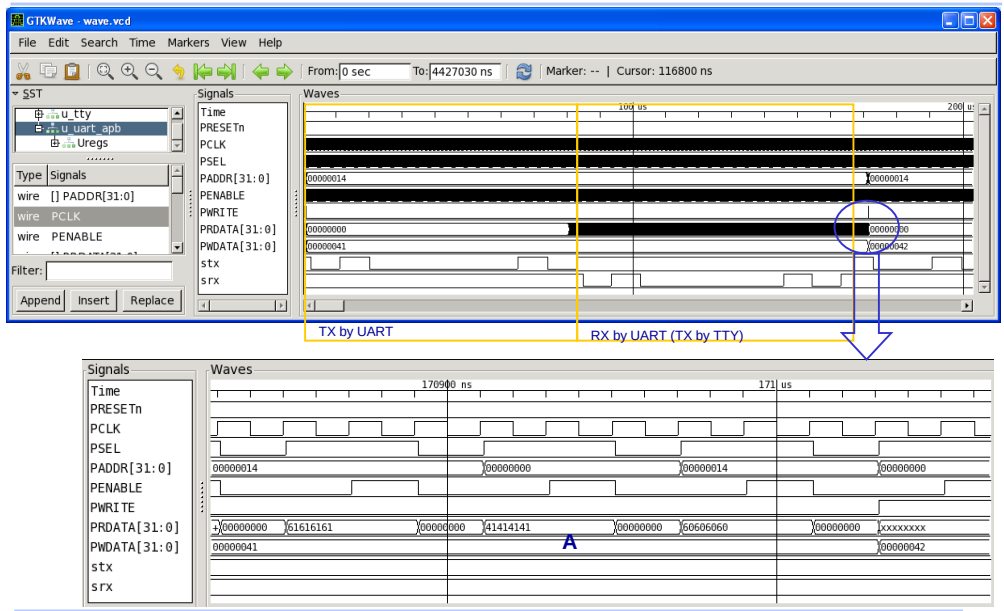
```
//------------------------------------------------------------------
+incdir+../../design/verilog
      ./sim_define.v
      ../../design/verilog/bfm_apb_s1.v
      ../../design/verilog/uart_apb.v
//------------------------------------------------------------------
// Below are test-bench
//------------------------------------------------------------------
+incdir+../../bench/verilog
      ../../bench/verilog/top.v
      ../../bench/verilog/tty.v
//------------------------------------------------------------------
```

sim_define.v

```
`ifndef _SIM_DEFINE_V_
`define _SIM_DEFNE_V_
//------------------------------------------------------------------
`define SIM      // define this for simulation case if you are not sure
`define VCD      // define this for VCD waveform dump
`define DEBUG
`define RIGOR
`define LOW_POWER
//------------------------------------------------------------------
`define CLK_FREQ  50000000
`define MEM_DELAY  0
//------------------------------------------------------------------
`endif
```

Simulation with ModelSim (3/4)

Simulation with ModelSim (4/4)



Example: APB BFM task-based case

- This example shows how to use BFM with tasks
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/p2_uart_bfm_apb
  - Step 2: see the codes
    - [user@host] cd $(PROJECT)/codes/p2_uart_bfm_apb/desing/verilog
  - Step 3: compile and run
    - [user@host] cd $(PROJECT)/codes/p2_uart_bfm_apb/sim/modelsim
    - [user@host] make
  - Step 4: waveform view
    - [user@host] gtkwave wave.vcd &

[user@host] cd $(PROJECT)/codes/p2_uart_bfm_abp/sim/modelsim
[user@host] make
[user@host] gtkwave wave.vcd &

Issues, project and quiz

- How to use interrupt
- How to implement parity in TTY model

References

- AMBA Specification, Rev 2.0, ARM Limited. (AMBA 2.0 APB)
- AMBA™ 3 APB Protocol v1.0, IHI 0024B, ARM, 2004. (AMBA 3.0 APB 1.0)
- AMBA® APB Protocol Version: 2.0, IHI 0024C (ID041610), ARM, 2010. (AMBA 4.0 APB 2.0)
- Jacob Gorban, UART IP Core Specification, Aug. 11, 2002. (www.opencores.org)