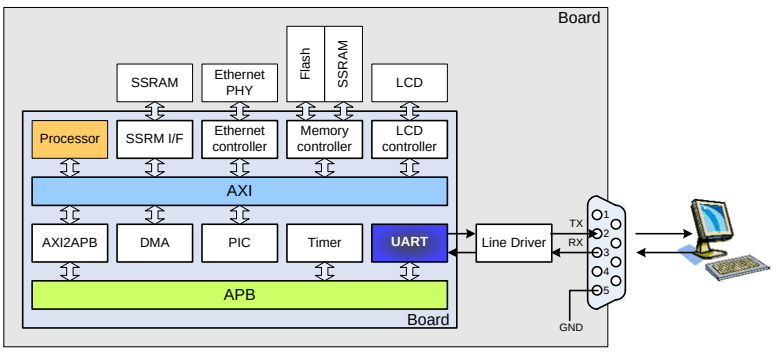
AMBA AXI BFM Based UART Verification

2014 – 2016

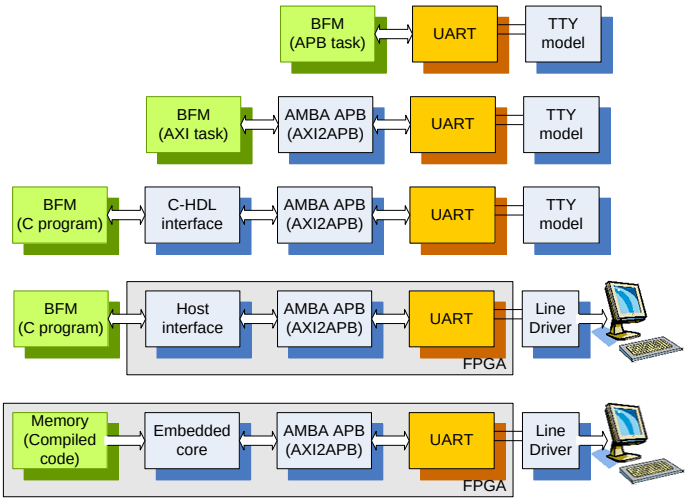
Ando Ki, Ph.D.
(adki@future-ds.com)

What to do

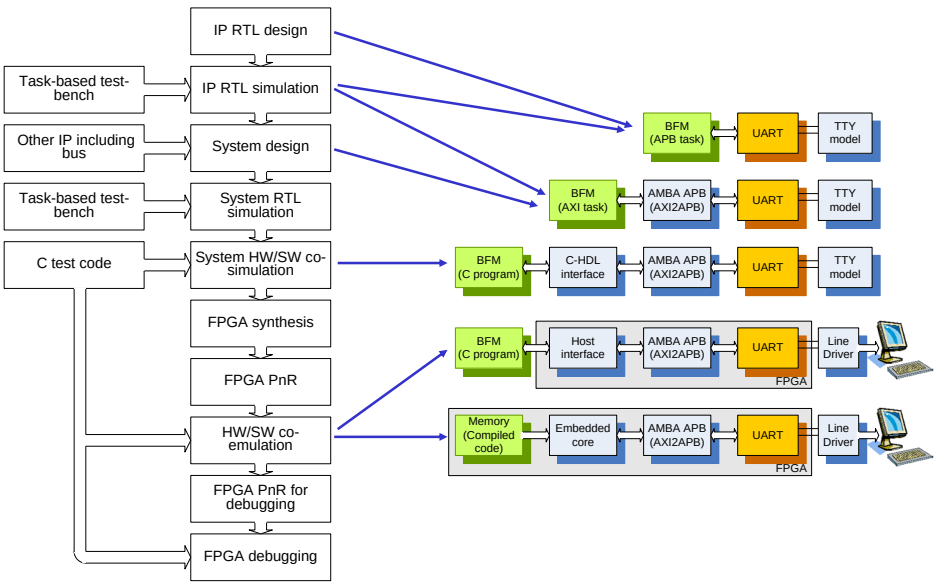
- Verify UART (Universal Asynchronous Receiver and Transmitter) through BFM (Bus Functional Model)
- UART (Universal Asynchronous Receiver and Transmitter)
  - a semiconductor chip providing the RS-232-C asynchronous serial communication protocol. One side of UART is an interface to processor and the other side is the serial port.



Verification plan



Design flow point of view

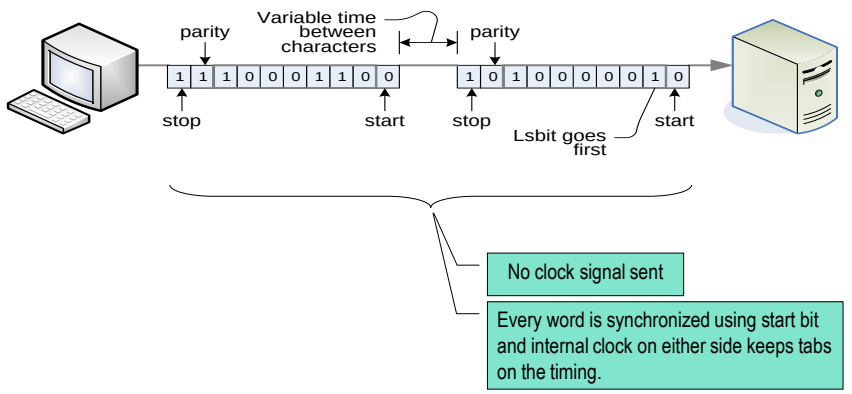


Agenda

- 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

RS-232C protocol

Asynchronous serial data transmission: RS-232C protocol

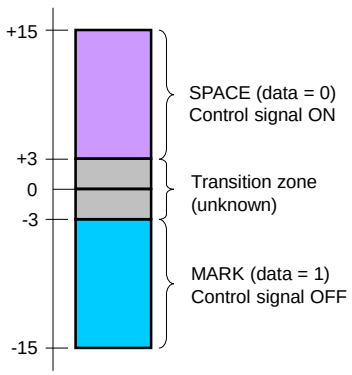


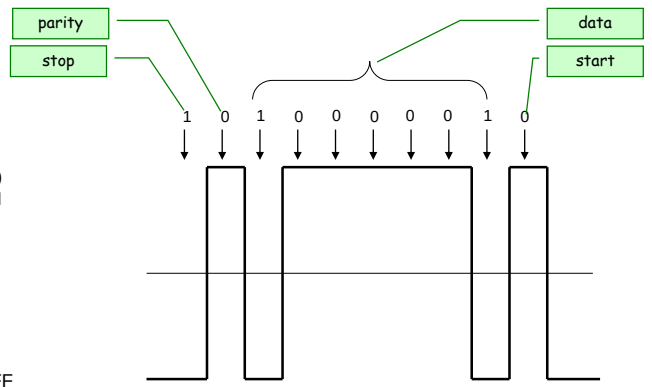
RS-232C: Recommended Standards

RS-232-C and UART

- 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 voltage level

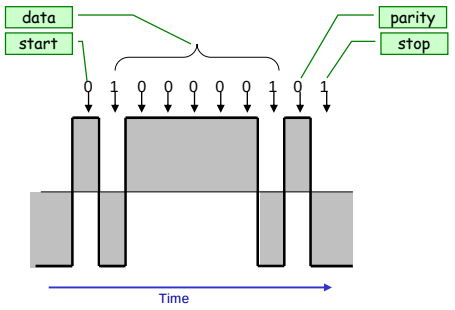


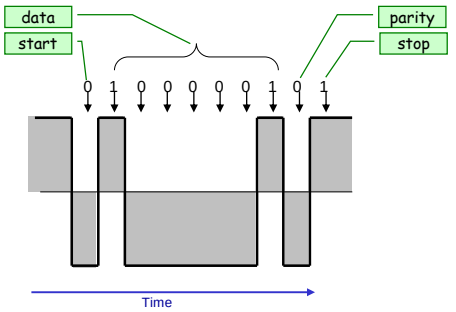


An asynchronous RS-232-C terminal must first send (starting from the left) the start bit, a 0, or +15 volts, followed by a series of zeroes and/or ones, and finally the stop bit, a 1, or -15 volts. After sending the stop bit, the transmitter remains at the negative voltage until the next character.

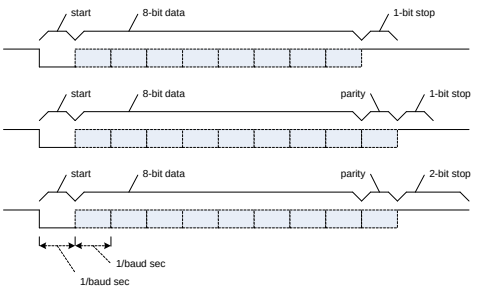
RS-232-C and UART signals

- 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.





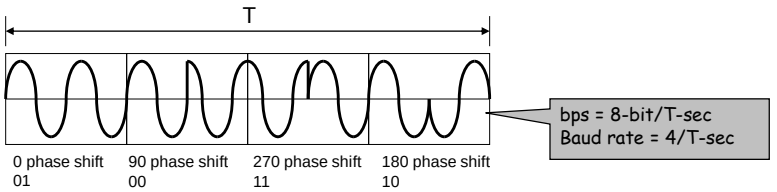
Baud rate, data width, parity and stop



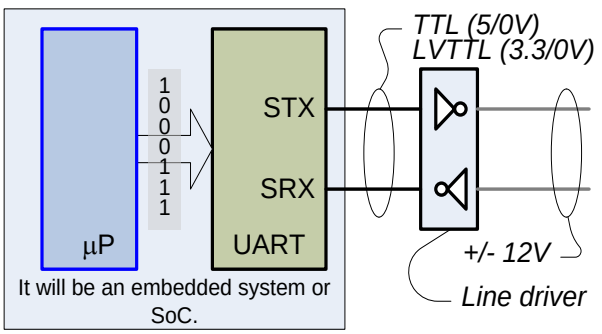
- Parameters
  - Baud rate
    - Possible signal changes in a second.
    - Duration of each bit
  - Data width
    - 8-bit or 7-bit
  - Parity
    - enable or disable
    - even or odd when enable
  - Stop
    - 1-bit or 2-bit

Bps and baud rate

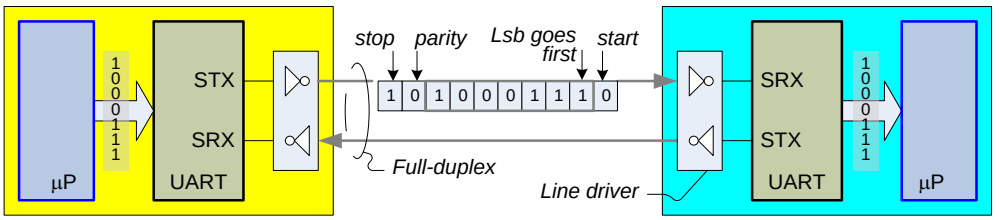
- Bps: bits per second
  - Possible bits transmitted on a uni-directional way in a second.
- Baud rate
  - Possible signal changes in a second.
- When only one bit is sent with each signal change, bps and baud are equivalent.
  - So RS-232-C can use bps and baud rate interchangeably.
- When multi-bit information is sent with each signal change using modulation (multi-frequency, multi-level, multi-phase), bps is higher than baud rate.



UART and line driver



UART hardware specification

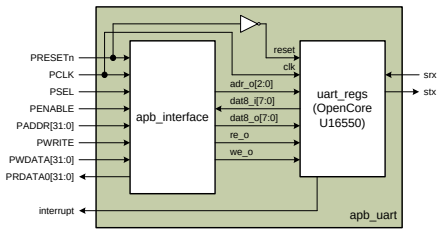


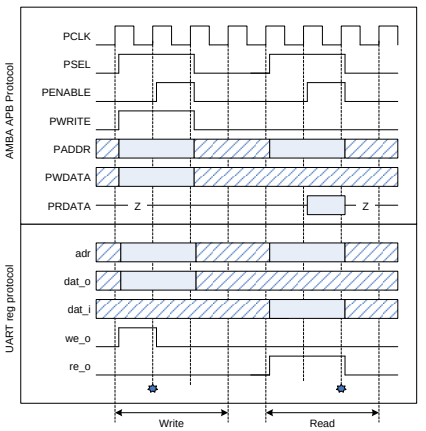
- UART (Universal Asynchronous Receiver and Transmit) is a semiconductor chip
  - It implements RS-232-C asynchronous serial communication protocol.
  - One side of it is a parallel interface to the processor and the other side is the RS-232-C serial port.
  - UART has CSR (Control and Status Register)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

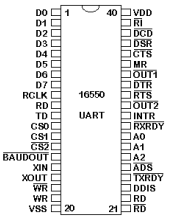
UART16550 hardware specification

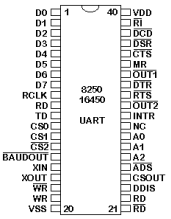




Types of UARTS

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.





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

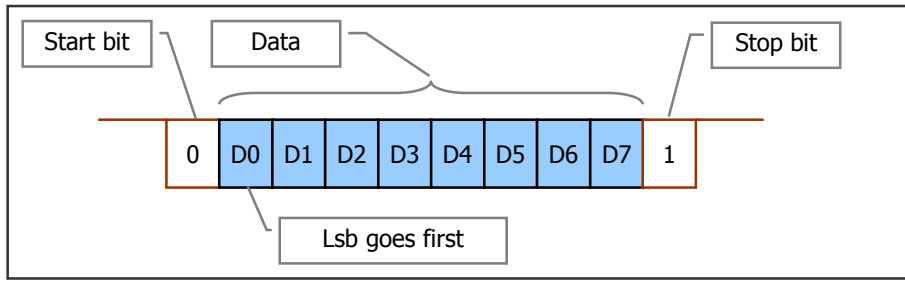
LSR: line status register (2/3)

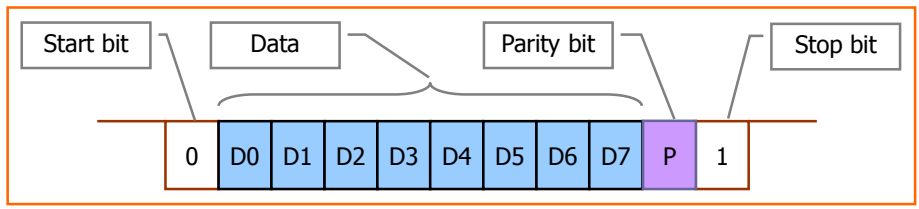
Bit #	Access	Description
3	R	Framing Error (FE) indicator '1' – The received character at the top of the FIFO did not have a valid stop bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. '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

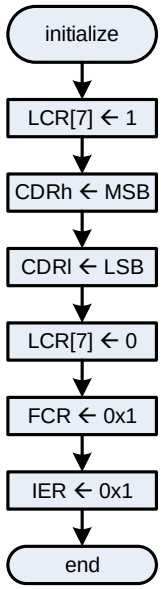
BR is baud rate.

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

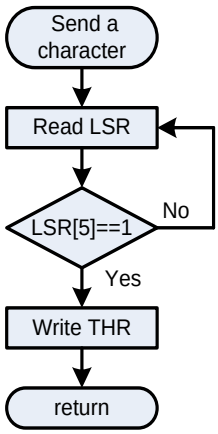
Calculate the value of divisor latches (DL[15:8] and DL[7:0]) for 9,600 baud rate with 33MHz input clock.

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.

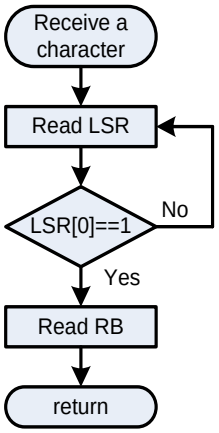


How to transmit a character




```
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



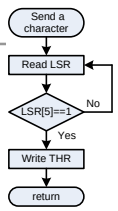

```
unsigned int
uart_get_char(void) {
  while (!(_UART->LSR&0x1));
  // wait until a character has been received
  return (unsigned int)_UART->RB_THR;
}
```


How to control HW through SW

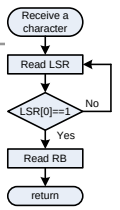
```
static struct uart16550 {
  unsigned char RB_THR;
  unsigned char IER;
  unsigned char IIR_FCR;
  unsigned char LCR;
  unsigned char MCR;
  unsigned char LSR;
  unsigned char MSR;
} *_UART;
```

```
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;
}
```

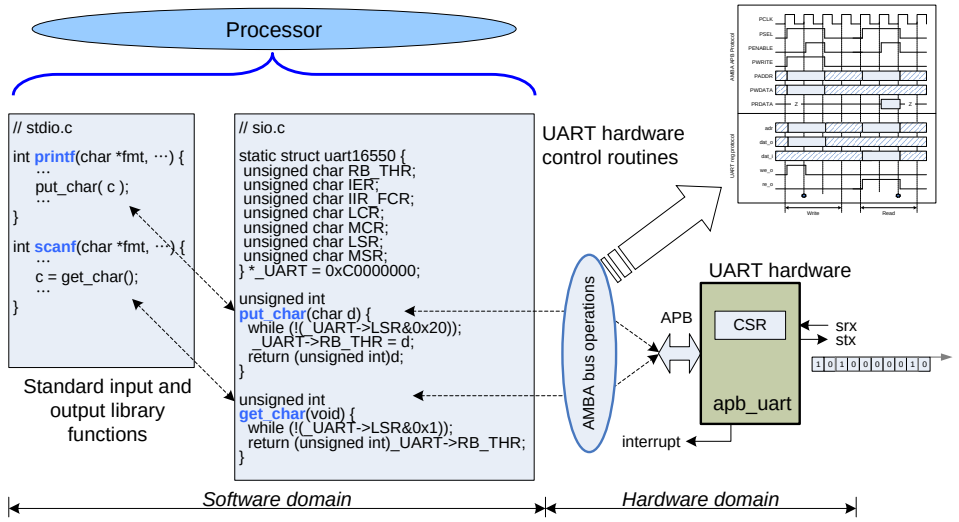
```
unsigned int
put_char(char d) {
  while (!(_UART->LSR&0x20));
   // wait until transmitter FIFO is empty
  _UART->RB_THR = d;
  return (unsigned int)d;
}
```




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



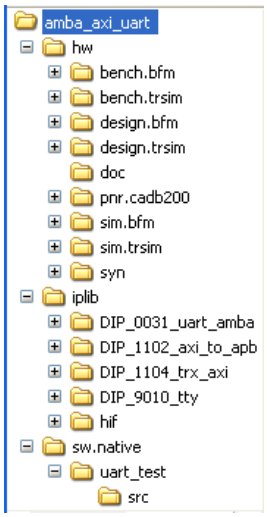
How to control HW through SW



Agenda

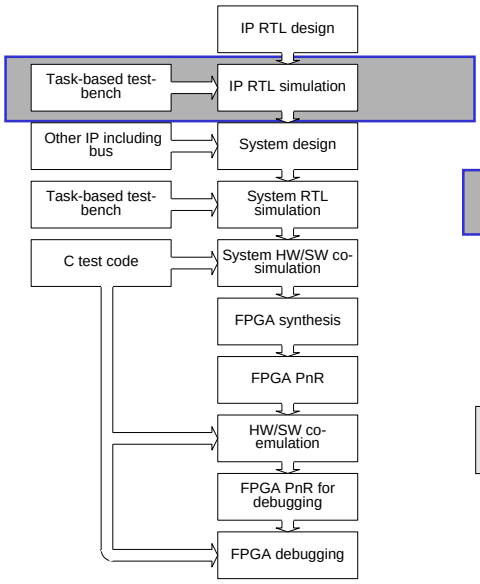
- Directory structure
- Task-based BFM verification
- C-based BFM verification
  - Transaction-level simulation
  - C-FPGA emulation

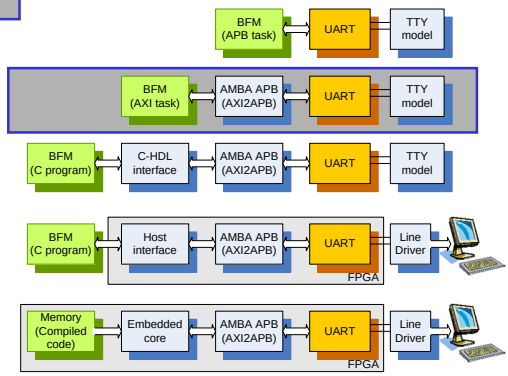
Directory structure



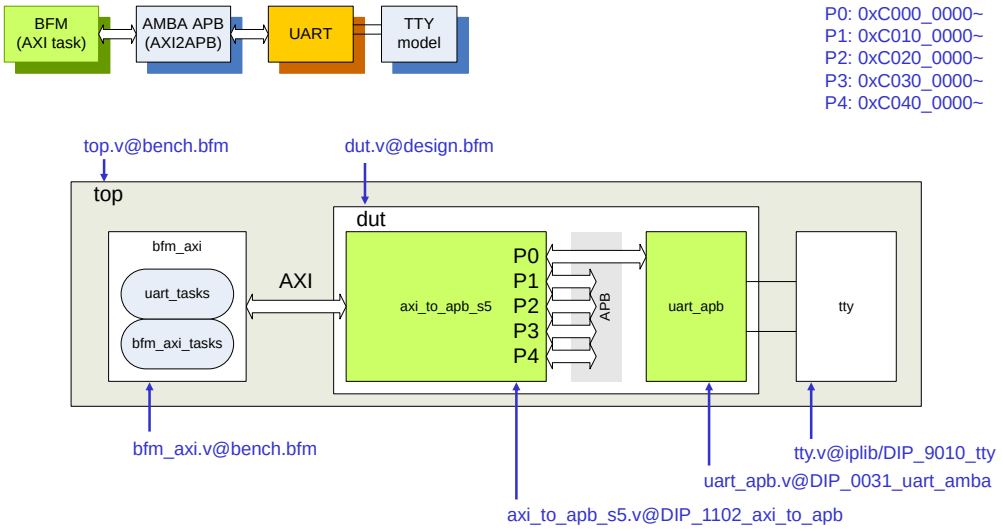
- Task-based BFM verification
  - hw/bench.bfm
  - hw/design.bfm
  - hw/sim.bfm
- C-based BFM verification (C-HDL)
  - hw/bench.trsim
  - hw/design.trsim
  - hw/sim.trsim
  - sw/uart_test
- C-based BFM verification (C-FPGA)
  - hw/design.trsim
  - hw/syn
  - hw/pnr.cadb200
  - sw/uart_test
- Additional blocks
  - TTY (text terminal model)
  - HIF (host interface)

Design flow point of view

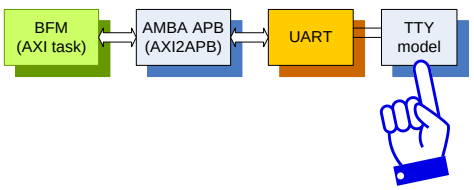




Task-based BFM verification



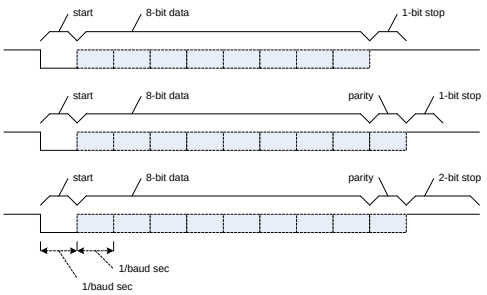
TTY behavioral model (1/2)




```
`timescale 1ns/1ns

module tty #(parameter BAUD_RATE=119200
                , WIDTH=0 // 0=8-bit, 1=7-bit
                , PARITY=0 // 0=no-parity
                , PARITY_EVEN=0 // valid when PARITY==1
                , STOP=0 // 0=1-bit, 1=2-bit
                )
(
  output  reg  STX,
  input   wire SRX
);
  //-----------------------------------------
  localparam INTERVAL = (1_000_000_000/BAUD_RATE); // nsec
  //-----------------------------------------
  reg [3:0] width;
  reg [7:0] data;
  reg [7:0] x;
  reg       pebit; // even parity
  //-----------------------------------------
  initial begin STX = 1'b1; data = 0; x = 0; end
  //-----------------------------------------
  always @ (negedge SRX) begin
      width = (WIDTH) ? 7 : 8;
      pebit = 1'b0;
      x = 0;
      #(INTERVAL*1.5);
      repeat (width) begin // LSB comes first
          data[x] = SRX;
          pebit = pebit ^ SRX;
          x = x + 1;
          #(INTERVAL);
      end
```

- RTL (Register Transfer Level) model means synthesizable code.
- Behavioral model means a kind of functional model not for synthesis, but for simulation.

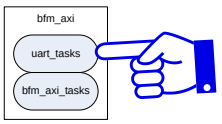
TTY behavioral model (2/2)




```
`        if (PARITY) begin
            pebit = pebit ^ SRX;
            if (PARITY_EVEN) begin // even-parity
               if (pebit) $display($time,,"%m even parity error");
            end else begin
               if (!pebit) $display($time,,"%m odd parity error");
            end
            #(INTERVAL);
         end
         if (SRX!=1'b1) $display($time,,"%m stop-bit error first");
         if (STOP) begin
            #(INTERVAL);
            if (SRX!=1'b1) $display($time,,"%m stop-bit error second");
         end
         //#(INTERVAL*0.5);
        `ifdef DEBUG
        if ((data>=8'h20)&&(data<=8'h7E))
            $display($time," 0x%x(%c) received!", data, data);
        else
            $display($time," 0x%x  received!", data);
        $fflush();
        `else
        $write("%c", data);
        $fflush();
        `endif
    end

endmodule
```

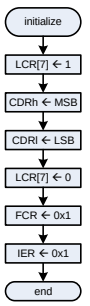

UART tasks (1/2)

- Look at 'uart_tasks.v' in '.../hw/bench.bfm/verilog' directory



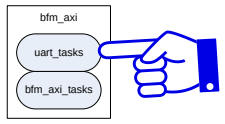

```
task apb_read;
    input  [31:0] addr;
    output [31:0] data;
begin
    read_task(MST_ID,addr,4,1,2'b01);
    data = dataRB[0];
end
endtask
//-------------------------------------------------------
task apb_write;
    input  [31:0] addr;
    input  [31:0] data;
begin
    dataWB[0] = data;
    write_task(MST_ID,addr,4,1,2'b01);
end
endtask
```

```
 // U16550 CSR address
  localparam ADDR_UART=32'hC000_0000;
  localparam RB_THR  = ADDR_UART + 0
           , IER     = ADDR_UART + 4
           , IIR_FCR = ADDR_UART + 8
           , LCR     = ADDR_UART + 12
           , MCR     = ADDR_UART + 16
           , LSR     = ADDR_UART + 20
           , MSR     = ADDR_UART + 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);
      apb_write(IER,    {24'h0,dl[15:8]});
      apb_write(RB_THR, {24'h0,dl[ 7:0]});
      apb_write(LCR,    32'h03);
      apb_write(IIR_FCR,32'h01);
      apb_write(IER,    32'h01);
  end
  endtask
```

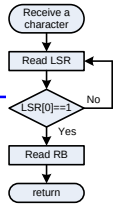


UART tasks (2/2)

- Look at 'uart_tasks.v' in '.../hw/bench.bfm/verilog' directory



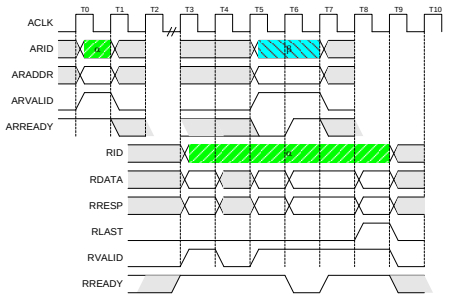

```
task send_a_character;
    input [ 7:0] dat;
    reg   [31:0] tmp;
 begin
      apb_read(LSR, tmp);
      // wait until buffer empty
      while (!(tmp&32'h20)) apb_read(LSR, tmp);
      apb_write(RB_THR, {24'h0,dat});
 end
 endtask
```

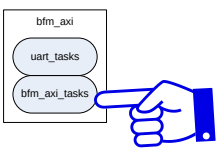



```
task receive_a_character;
    output [ 7:0] dat;
    reg    [31:0] tmp;
 begin
      apb_read(LSR, tmp);
      // wait until a character recived
      while (!(tmp&32'h01)) apb_read(LSR, tmp);
      apb_read(RB_THR, tmp);
      dat = tmp[7:0];
 end
 endtask
```


AXI BFM: read task (1/3)

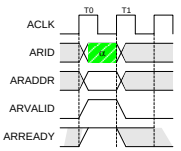
```
task read_task;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
begin
    fork
    read_address_channel(id,addr,size,leng,type);
    read_data_channel(id,addr,size,leng,type);
    join
end
endtask
```



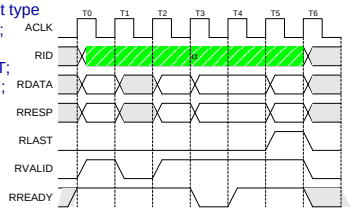


AXI BFM: read task (2/3)

```
task read_address_channel;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
begin
    @ (posedge ACLK);
    ARID    <= #1 id;
    ARADDR  <= #1 addr;
    ARLEN   <= #1 leng-1;
    ARLOCK  <= #1 'b0;
    ARSIZE  <= #1 get_size(size);
    ARBURST <= #1  type[1:0];
    `ifdef AMBA_AXI_PROT
    ARPROT  <= #1 'h0; // data, secure, normal
    `endif
    ARVALID <= #1 'b1;
    @ (posedge ACLK);
    while (ARREADY==1'b0) @ (posedge ACLK);
    ARVALID <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```

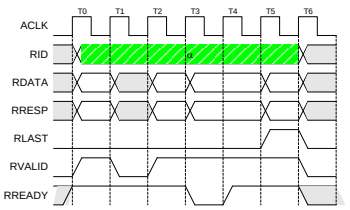



```
task read_data_channel;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
    reg   [WIDTH_AD-1:0] naddr;
    reg   [WIDTH_DS-1:0] strb;
    reg   [WIDTH_DA-1:0] maskT;
    reg   [WIDTH_DA-1:0] dataR;
    integer idx, idy, idz;
begin
    idz = 0;
    naddr  = addr;
    @ (posedge ACLK);
    RREADY <= #1 1'b1;
    for (idx=0; idx<leng; idx=idx+1) begin
        @ (posedge ACLK);
        while (RVALID==1'b0) @ (posedge ACLK);
        strb = get_strb(naddr, size);
        dataR = RDATA;
        for (idy=0; idy<WIDTH_DS; idy=idy+1) begin
            if (strb[idy]) begin
                dataRB[idz] = dataR&8'hFF; // justified
                idz = idz + 1;
            end
            dataR = dataR>>8;
        end
```



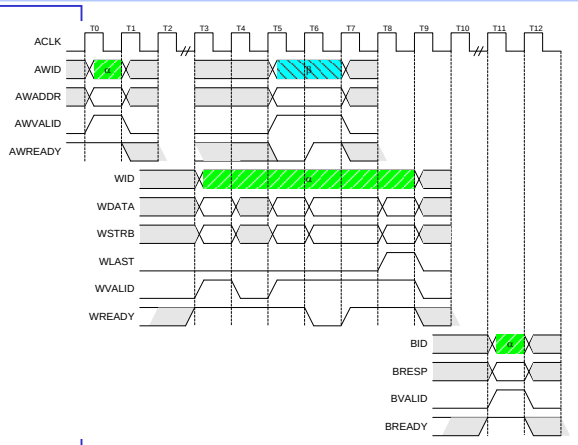
AXI BFM: read task (3/3)

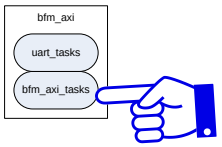
```
        if (id!=RID) begin
          $display($time,,"%m Error id/RID mis-match for read-data-
channel", id, RID);
        end
        if (idx==leng-1) begin
          if (RLAST==1'b0) begin
            $display($time,,"%m Error RLAST expected for read-
data-channel");
          end
        end else begin
          @ (negedge ACLK);
          naddr = get_next_addr( naddr  // current address
                                 , size  // num of bytes in a beat
                                 , type);// type of burst
        end
    end
    RREADY <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```



AXI BFM: write task (1/3)

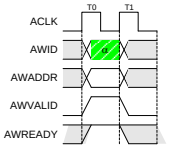
```
task write_task;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
begin
    fork
    write_address_channel(id,addr,size,leng,type);
    write_data_channel(id,addr,size,leng,type);
    write_resp_channel(id);
    join
end
endtask
```



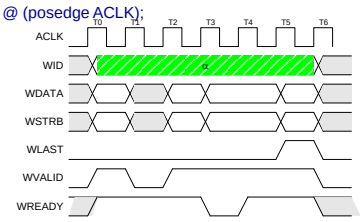


AXI BFM: write task (2/3)

```
task write_address_channel;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
begin
    @ (posedge ACLK);
    AWID    <= #1 id;
    AWADDR  <= #1 addr;
    AWLEN   <= #1 leng-1;
    AWLOCK  <= #1 'b0;
    AWSIZE  <= #1 get_size(size);
    AWBURST <= #1  type[1:0];
    `ifdef AMBA_AXI_PROT
    AWPROT  <= #1 'h0; // data, secure, normal
    `endif
    AWVALID <= #1 'b1;
    @ (posedge ACLK);
    while (AWREADY==1'b0) @ (posedge ACLK);
    AWVALID <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```

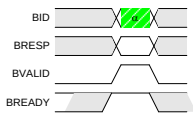



```
task write_data_channel;
    input [31:0]        id;
    input [WIDTH_AD-1:0] addr;
    input [31:0]        size; // 1 ~ 128 byte in a beat
    input [31:0]        leng; // 1 ~ 16  beats in a burst
    input [31:0]        type; // burst type
    reg   [WIDTH_AD-1:0] naddr;
    integer idx;
begin
    naddr  <= addr;
    @ (posedge ACLK);
    WID    <= #1 id;
    WVALID <= #1 1'b1;
    for (idx=0; idx<leng; idx=idx+1) begin
        WDATA <= #1 get_data(addr, naddr, size);
        WSTRB <= #1 get_strb(naddr, size);
        WLAST <= #1 (idx==(leng-1));
        naddr <= get_next_addr(naddr, size, type);
        @ (posedge ACLK);
        while (WREADY==1'b0) @ (posedge ACLK);
    end
    WLAST  <= #1 'b0;
    WVALID <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```



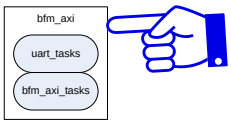
AXI BFM: write task (3/3)

```
task write_resp_channel;
    input [31:0] id;
begin
    BREADY <= #1 'b1;
    @ (posedge ACLK);
    while (BVALID==1'b0) @ (posedge ACLK);
    if (id!=BID) begin
      $display($time,,"%m Error id mis-match for write-resp-
channel 0x%x/0x%x", id, BID);
    end else begin
       case (BRESP)
       2'b00: begin
            `ifdef DEBUG
            $display($time,,"%m OK response for write-resp-channel:
OKAY");
            `endif
            end
       2'b01: $display($time,,"%m OK response for write-resp-
channel: EXOKAY");
       2'b10: $display($time,,"%m Error response for write-resp-
channel: SLVERR");
       2'b11: $display($time,,"%m Error response for write-resp-
channel: DECERR");
       endcase
    end
    BREADY <= #1 'b0;
    @ (negedge ACLK);
end
endtask
```



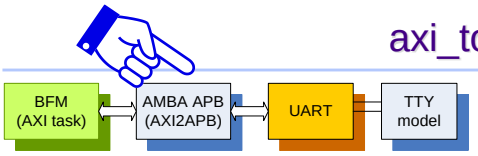
bfm_axi.v

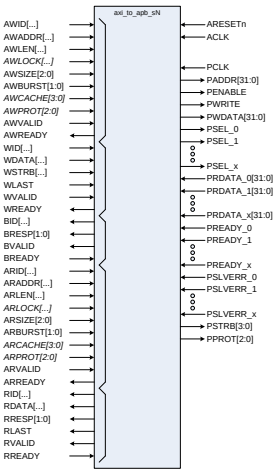
- Look at 'bfm_axi.v' in '.../hw/bench.bfm/verilog' directory

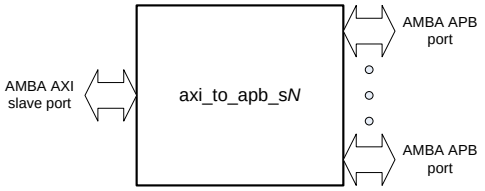


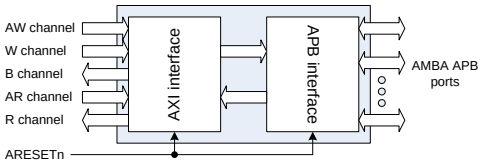

```
module bfm_axi #(parameter ... )
(
     input  wire              ARESETn
    , input  wire              ACLK
    , output reg  [WIDTH_CID-1:0] MID=MST_ID
    ...
);
    ...
    //-----------------------------------------------------
    initial begin
        wait (ARESETn==1'b0);
        wait (ARESETn==1'b1);
        repeat (2) @ (posedge ACLK);
        //-----------------------------------------------------
        @ (posedge ACLK); stamp = $realtime;
        @ (posedge ACLK); delta = $realtime - stamp;
        freq_r = 1000000000.0/delta;
        freq = $rtoi(freq_r);
        repeat (2) @ (posedge ACLK);
        //-----------------------------------------------------
        if (EN) begin
          init_uart(freq,115200);
          for (char="A"; char<="Z"; char=char+1) begin
              send_a_character(char);
          end
          send_a_character(8'h0A); // \v (line feed)
          send_a_character(8'h0C); // \r (carriage return)
          wait_empty_send_buff;
        end
        //-----------------------------------------------------
        repeat (10) @ (posedge ACLK);
        DONE = 1'b1;
    end
    //-----------------------------------------------------------
    `include "bfm_axi_tasks.v"
    `include "uart_tasks.v"
....
endmodule
```

axi_to_apb.v

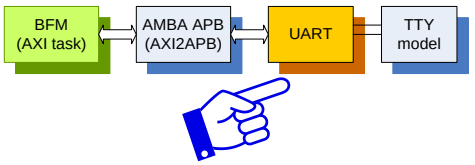


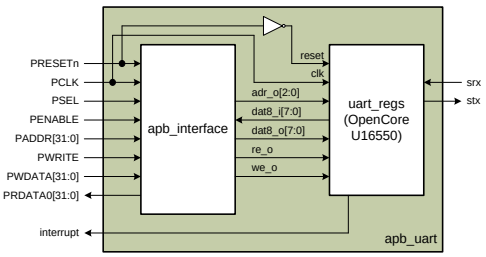






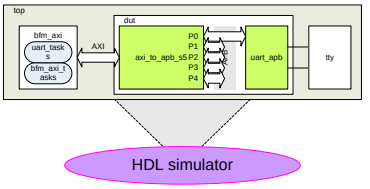
UART





Example: AXI 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/amba_axi_uart/hw
  - Step 2: see the codes
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/design.bfm/verilog
  - Step 3: compile and run
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/sim.bfm/modelsim
    - [user@host] make
  - Step 4: waveform view
    - [user@host] gtkwave wave.vcd &

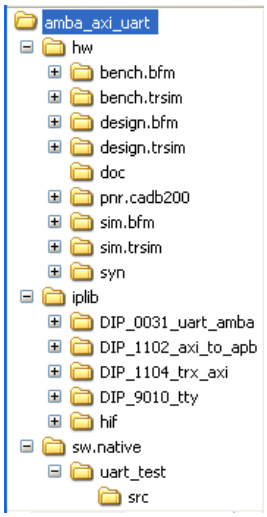



```
[user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/sim.bfm/modelsim
[user@host] make
[user@host] gtkwave wave.vcd &
```


Agenda

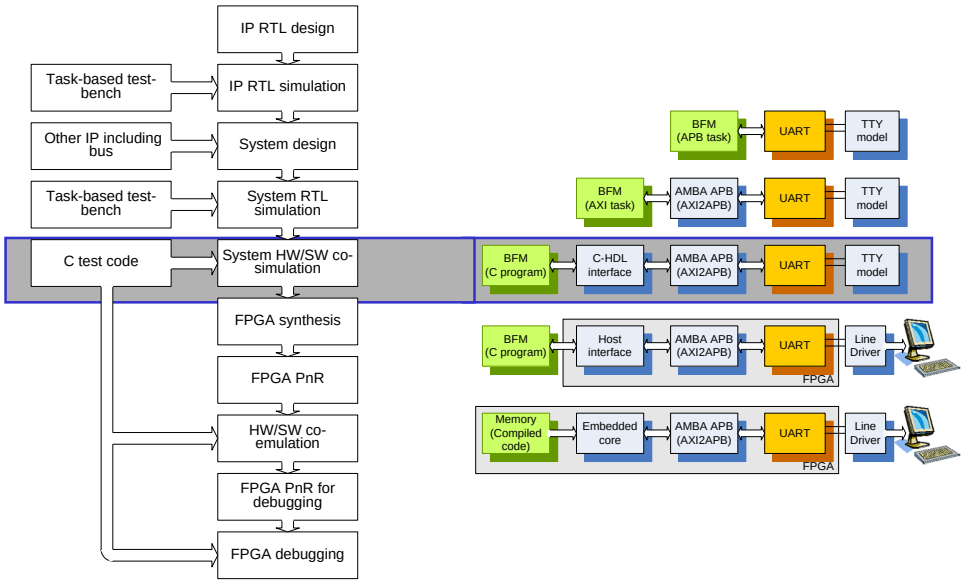
- Directory structure
- Task-based BFM verification
- C-based BFM verification
  - Transaction-level simulation
  - C-FPGA emulation

Directory structure

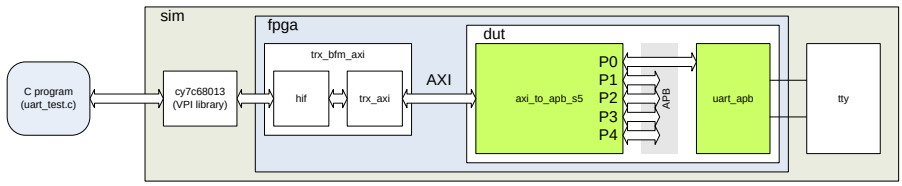


- Task-based BFM verification
  - hw/bench.bfm
  - hw/design.bfm
  - hw/sim.bfm
- C-based BFM verification (C-HDL)
  - hw/bench.trsim
  - hw/design.trsim
  - hw/sim.trsim
  - sw/uart_test
- C-based BFM verification (C-FPGA)
  - hw/design.trsim
  - hw/syn
  - hw/pnr.cadb200
  - sw/uart_test
- Additional blocks
  - TTY (text terminal model)
  - HIF (host interface)

Design flow point of view

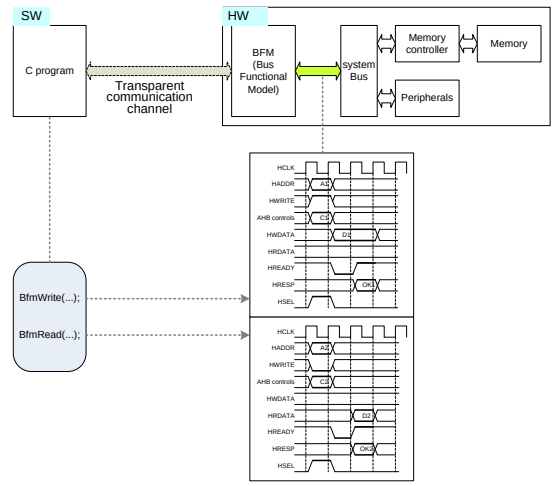


C-based BFM verification: C-HDL



C-based BFM Concept

- BFM API (Bus Functional Model Application Programming Interface) generates system bus transactions.
  - System bus can be AMBA AXI, AHB, or APB depending of BFM.



BFM C API

Generate bus write transaction

```
void BfmWrite( unsigned int  addr
             , unsigned int *data
             , unsigned int  size
             , unsigned int  length);
```

- 'addr': address to access
  - It should be a multiple of 'size'.
- 'data': pointer to buffer containing data to be written
  - Data of each 'data' element should be justified.
- 'size': num of bytes that each 'data' element contains, it can be 1, 2, or 4.
- 'length': num of 'data' elements to be written.

Generate bus read transaction

```
void BfmRead ( unsigned int  addr
             , unsigned int *data
             , unsigned int  size
             , unsigned int  length);
```

- 'addr': address to access
  - It should be a multiple of 'size'.
- 'data': pointer to buffer containing data to be return
  - Data of each 'data' element should be justified.
- 'size': num of bytes that each 'data' element contains, it can be 1, 2, or 4.
- 'length': num of 'data' elements to be written.

An example program

- In addition to BFM API, there are a number of host interface C API including FPGA configuration.
  - Use 'iProveInitiailze()' function, which takes care of all details.
    - It uses two global strings: 'FILE_EIF' and 'INST_TRX'
    - It specifies an FPGA board using 'card_id' variable, which is Card Identification Num.

```
// main.c
int main(int argc, char *argv[ ]) {
  ...
  if (iProveInitialize()) exit(1); // see init_fpga.c
  test_bench(); // see test_bench.c
  ...
}
```

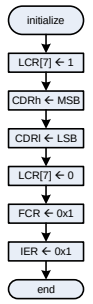
```
// init_fpga.c
#include "iprove.h" // Dynalith specific header
char  FILE_EIF[128]    ="../../inspire/par/fpga0.eif";
char  INST_TRX[64]    ="dut";
int iProveInitialize (void) {
    iProveLoadEmulationInfoFile(FILE_EIF);
    iProveGetModuleHandle(INST_TRX, &handle_trx);
    iProveStart(card_id);
   return(0);
}
```

```
// test_bench.c
#include "trx_axi_api.h" // bfm specific header
void test_bench() {
      unsigned int addr;
      unsigned int dataW[128], dataR[128];
      for (addr=0x0; addr<0x100; addr+=4) {
            dataW[addr] = addr+1;
      }
      for (addr=0x0; addr<0x100; addr+=4) {
            BfmWrite(addr, dataW, 4, 1);
            BfmRead(addr, dataR, 4, 1);
      }
      for (addr=0x0; addr<0x100; addr+=4) {
            if (dataR[addr]!=(addr+1)) {
                  printf("Read-After-Write error at 0x%x", addr);
            }
      }
}
```

UART API (1/2)

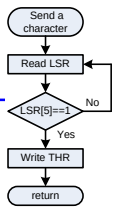
- Look at 'uart_api.c' in '.../sw.native/src' directory

```
#ifndef ADDR_UART_START
#define ADDR_UART_START 0xC0000000
#endif

#define UART_RB_THR  (ADDR_UART_START+0x00) // read for RB, write for THR
#define UART_IER     (ADDR_UART_START+0x04) // read/write
#define UART_IIR_FCR (ADDR_UART_START+0x08) // read for IIR, write for FCR
#define UART_LCR     (ADDR_UART_START+0x0C) // read/write (0x03) -- line_control;
#define UART_MCR     (ADDR_UART_START+0x10) // write only (0x00) -- modem_control;
#define UART_LSR     (ADDR_UART_START+0x14) // read only -- line_status;
#define UART_MSR     (ADDR_UART_START+0x18) // read only -- modem_status;
//-----------------------------------------------------------
void uart_init(uint32_t baud, uint32_t freq) {
  uint32_t value, divisor, divisorH, divisorL;
  divisor   = freq/16/(baud/10)+5;
  divisor   = divisor/10;
  divisorH  = (divisor>>8)&0xFF;
  divisorL  = divisor&0xFF;

  value = 0x83;
  REGW(UART_LCR    , value);     // DLAB (7th bit of LCR) = 1
  REGW(UART_IER    , divisorH); // Divisor latch MSB byte, when DLAB==1
  REGW(UART_RB_THR , divisorL); // Divisor latch LSB byte, when DLAB==1
  value = 0x03;
  REGW(UART_LCR    , value);    // DLAB (7th bit of LCR) = 0
  value = 0x00;
  REGW(UART_IIR_FCR, value); // set receiver FIFO trigger level to 1-byte
  value = 0x01;
  REGW(UART_IER    , value); // activate receive-data available interrupt
}
```

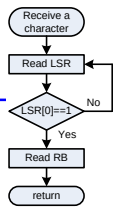


UART API (2/2)

- Look at 'uart_api.c' in '.../sw.native/src' directory



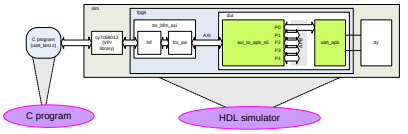

```
uint32_t uart_put_char(uint32_t c)
{
  volatile uint32_t lsr;
  do {
    REGR(UART_LSR, lsr);
  } while ((lsr&0x20)!=0x20);
  REGW(UART_RB_THR, c);
  return(c);
}
```




```
uint32_t uart_get_char(void)
{
  volatile uint32_t lsr;
  do {
    REGR(UART_LSR, lsr);
  } while ((lsr&0x1)!=0x1);
  REGR(UART_RB_THR, lsr);
  return lsr;
}
```

```
//-----------------------------------------------------------
// Register access macros
#ifdef TRX_BFM
  #include "bfm_api.h"
  #define REGR(A,B)  BfmRead((unsigned int)(A), (unsigned int*)&(B), 4, 1)
  #define REGW(A,B)  BfmWrite((unsigned int)(A), (unsigned int*)&(B), 4, 1)
#else
  ... ...
#endif
```

Example: AXI BFM task-based case



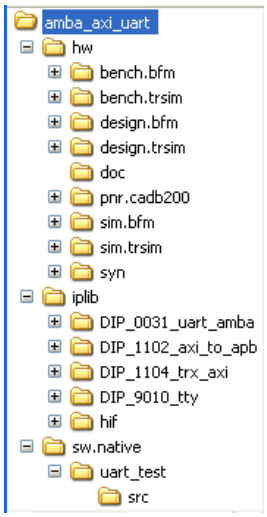
- This example shows how to use BFM with tasks
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw
  - Step 2: see the verilog codes
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/design.trsim/verilog
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/bench.trsim/verilog
  - Step 3: see the C code
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/sw.native/uart_test
  - Step 4: compile and run
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/sw.native/uart_test
    - [user@host] make cleanup; make trsim_run
  - Step 4: waveform view (go to simulation directory)
    - [user@host] cd $(PROJECT)/codes/amba_axi_uart/hw/sim.trsim/modelsim
    - [user@host] gtkwave wave.vcd &

```
[user@host] cd $(PROJECT)/codes/amba_axi_uart/sw.native/uart_test
[user@host] make cleanup
[user@host] make trsim.run
[user@host] gtkwave wave.vcd &
```


Agenda

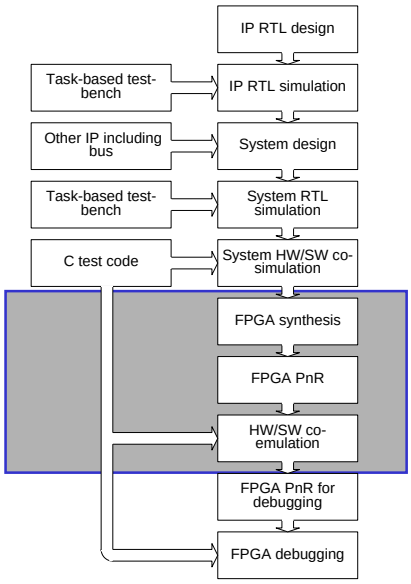
- Directory structure
- Task-based BFM verification
- C-based BFM verification
  - Transaction-level simulation
  - C-FPGA emulation

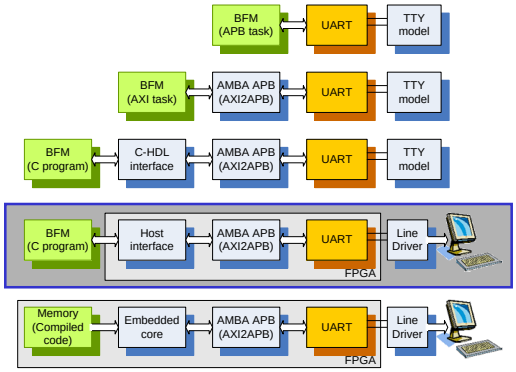
Directory structure



- Task-based BFM verification
  - hw/bench.bfm
  - hw/design.bfm
  - hw/sim.bfm
- C-based BFM verification (C-HDL)
  - hw/bench.trsim
  - hw/design.trsim
  - hw/sim.trsim
  - sw/uart_test
- C-based BFM verification (C-FPGA)
  - hw/design.trsim
  - hw/syn
  - hw/pnr.cadb200
  - sw/uart_test
- Additional blocks
  - TTY (text terminal model)
  - HIF (host interface)

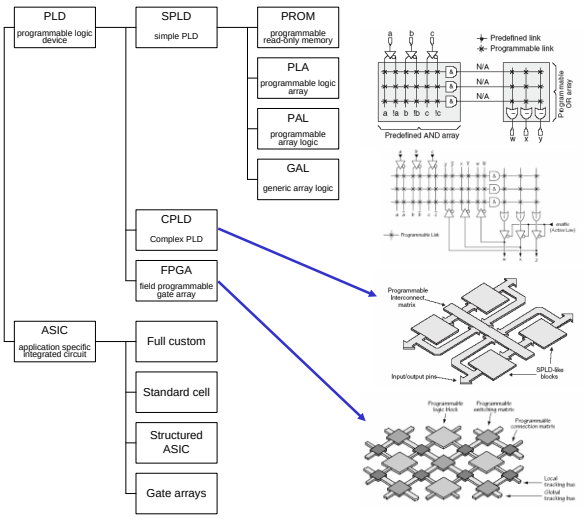
Design flow point of view



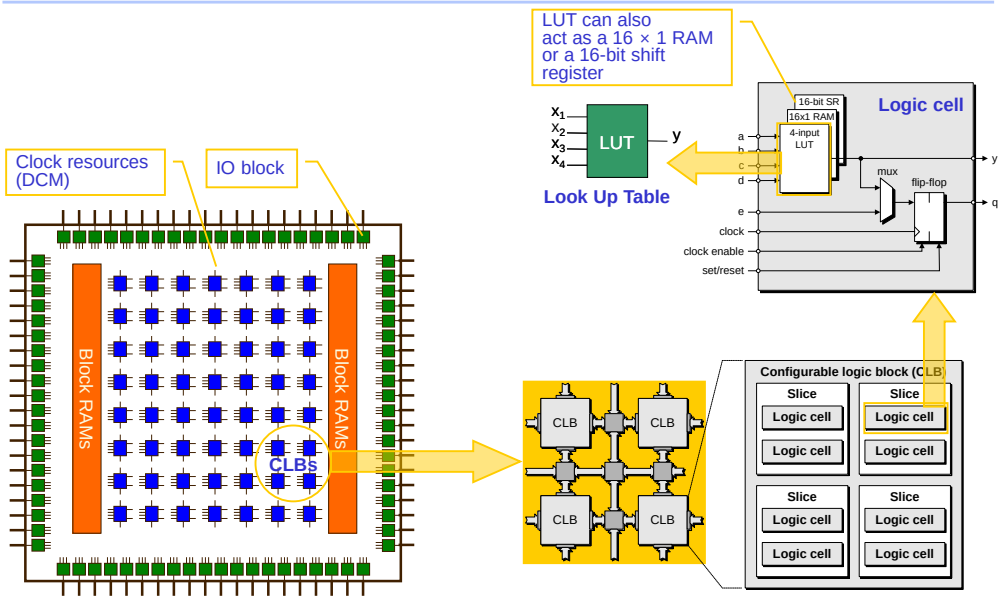


PLD and ASIC

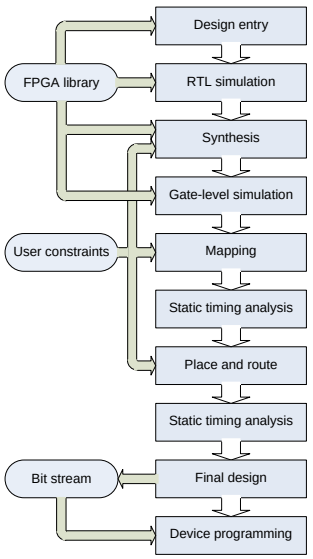
The Design Warrior's Guide to FPGAs Devices, Tools, and Flows. ISBN 0750676043

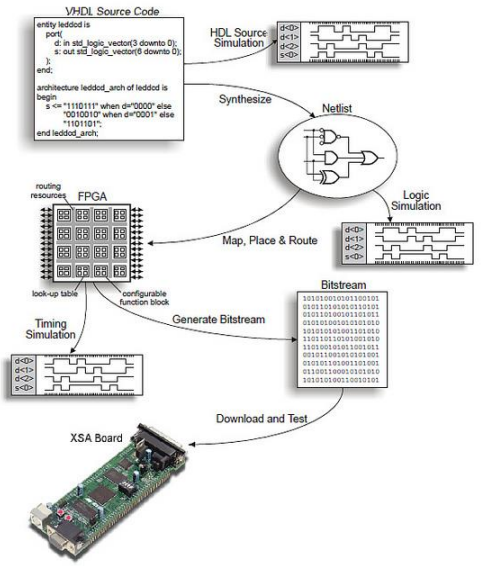


Xilinx FPGA example

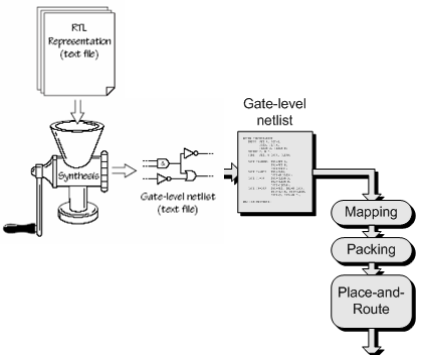


FPGA design flow

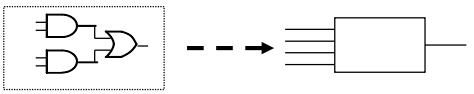




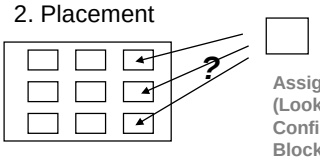
Circuit compilation for FPGA



1. Technology Mapping

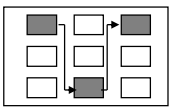


2. Placement



Assign a logical LUT (LookUp Table) / Configurable Logic Block) to a physical location.

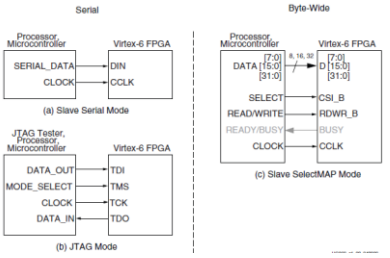
3. Routing



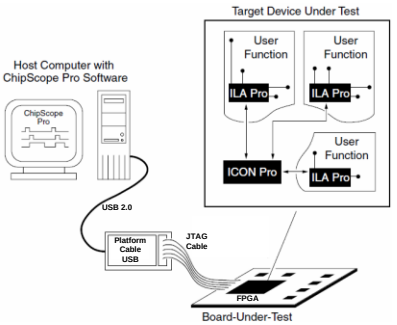
Select wire segments And switches for Interconnection.

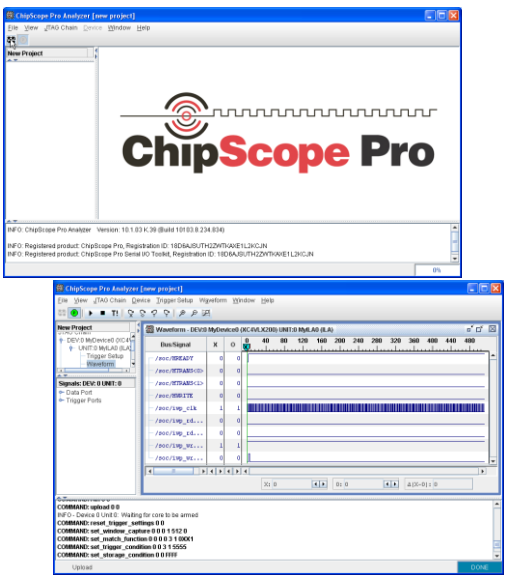
FPGA configuration

- Xilinx FPGA configuration path
  - JTAG/Boundary-scan configuration
  - Serial configuration
    - Master-serial
    - Slave-serial
  - SelectMAP configuration
    - Master SelectMAP (parallel)
    - Slave SelectMAP (parallel)
  - Master SPI (Serial Peripheral Interface) configuration
    - NOR FLASH
  - Master BPI (Byte Peripheral Interface) configuration
    - SPI serial FLASH

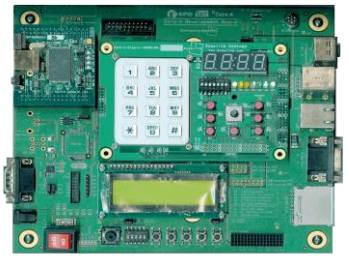


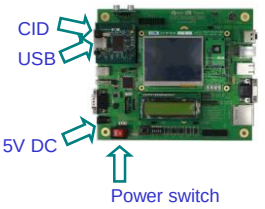
Xilinx FPGA debugging: ChipScope

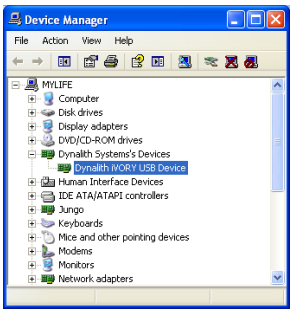




Core-A development board





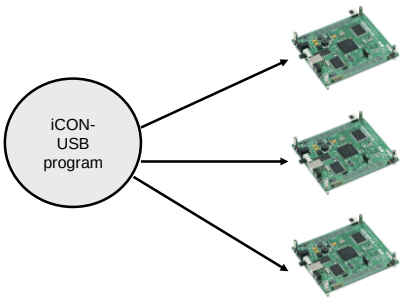


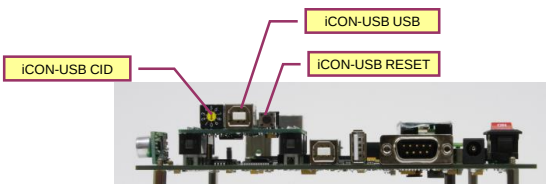
'Dynalith iVORY USB Device' should be listed in your 'Device Manager', when turned on

For the first time, install iVORY USB device driver.

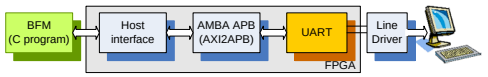
Card identification number

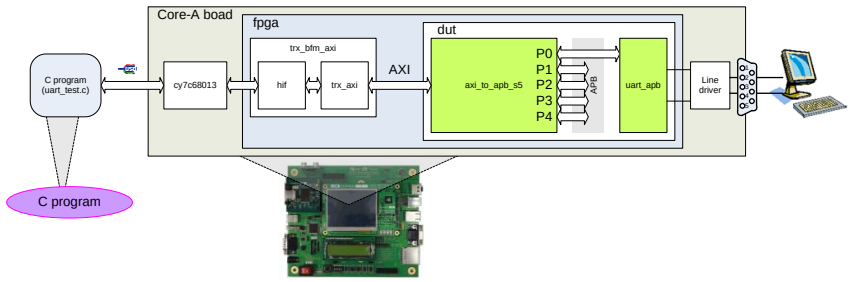
- Each iCON-USB is assigned a unique number from 0 to 7,
  - since more than one iCON-USB can be plugged into USB ports.
- EIF is Dynalith proprietary bit-stream file format
  - It contains Xilinx bit-stream and other information including card identification, FPGA type, and so on.





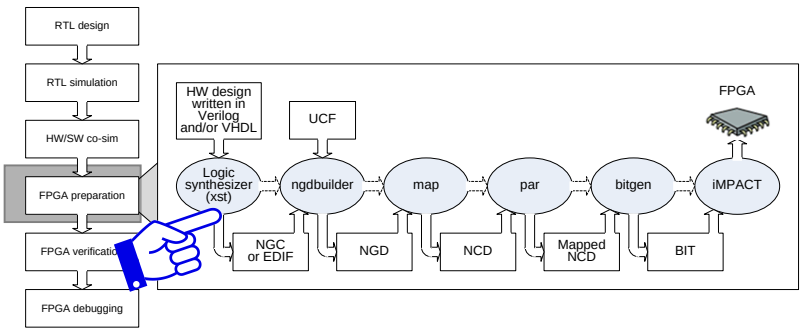
C-based BFM verification: C-HDL





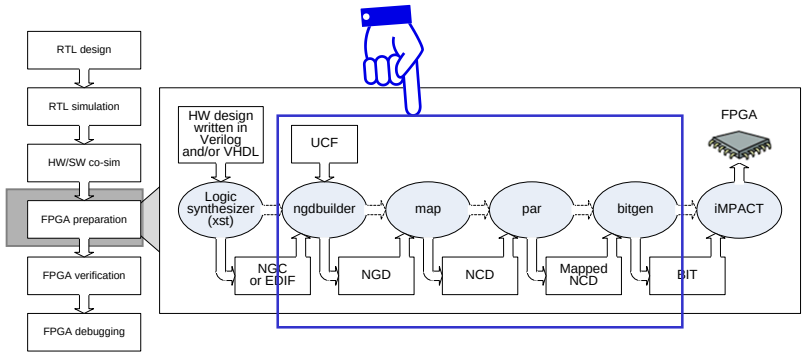
Logic synthesis

- Go to '.../hw/syn/xst.s3' directory
- Run 'make'



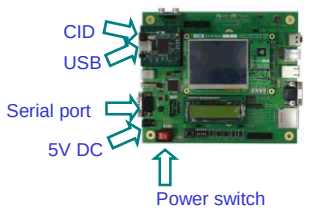
FPGA mapping, place & routing, bit-stream generation

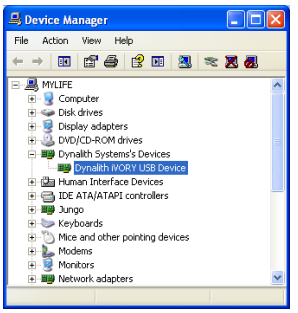
- Go to '.../hw/pnr.cadb' directory
- Run 'make'



Run C-FPGA

- Go to '.../sw.native/uart_test' directry
- Connect USB to the FPGA board (make sure FPGA turned off)
- Connect serial port to your host computer (use USB-to-Serial)
  - It may be requested installation of device driver of USB-to-Serial
- Invoke terminal emulator, e.g., teraterm, hyperterminal, or PuTTY, ...
  - Make sure baud rate, which should be 115200.
- Turn on the FPGA board
  - Install device driver, i.e., iVORY if required
    - iVORY will be found in C:/Dynalith/iNSPIRE/drv/ivory
- Run 'make cleanup',
- Then, run 'make'





Dynalith Systems H.Q.:
2nd Fl., Taejin Bldg, 24 Dolwoomul-Gil (14-2 Yangjae-Dong), Seocho-Ku, Seoul, 137-888, Korea
Tel: +82-2-556-0020; Fax: +82-2-556-2252
E-mail: corea@dynalith.com
Web: www.dynalith.com

Dynalith Systems R&D Center:
335 Gwahang-Ro (373-1 Kusong-Dong), Yusong-Gu, CHiPS B/D, KAIST, Taejon, 305-701, Korea
Tel: +82-42-862-6411; Fax: +82-42-862-6410

Dynalith Systems Co., Ltd.
(주) 다이나릿 시스템
www.dynalith.com