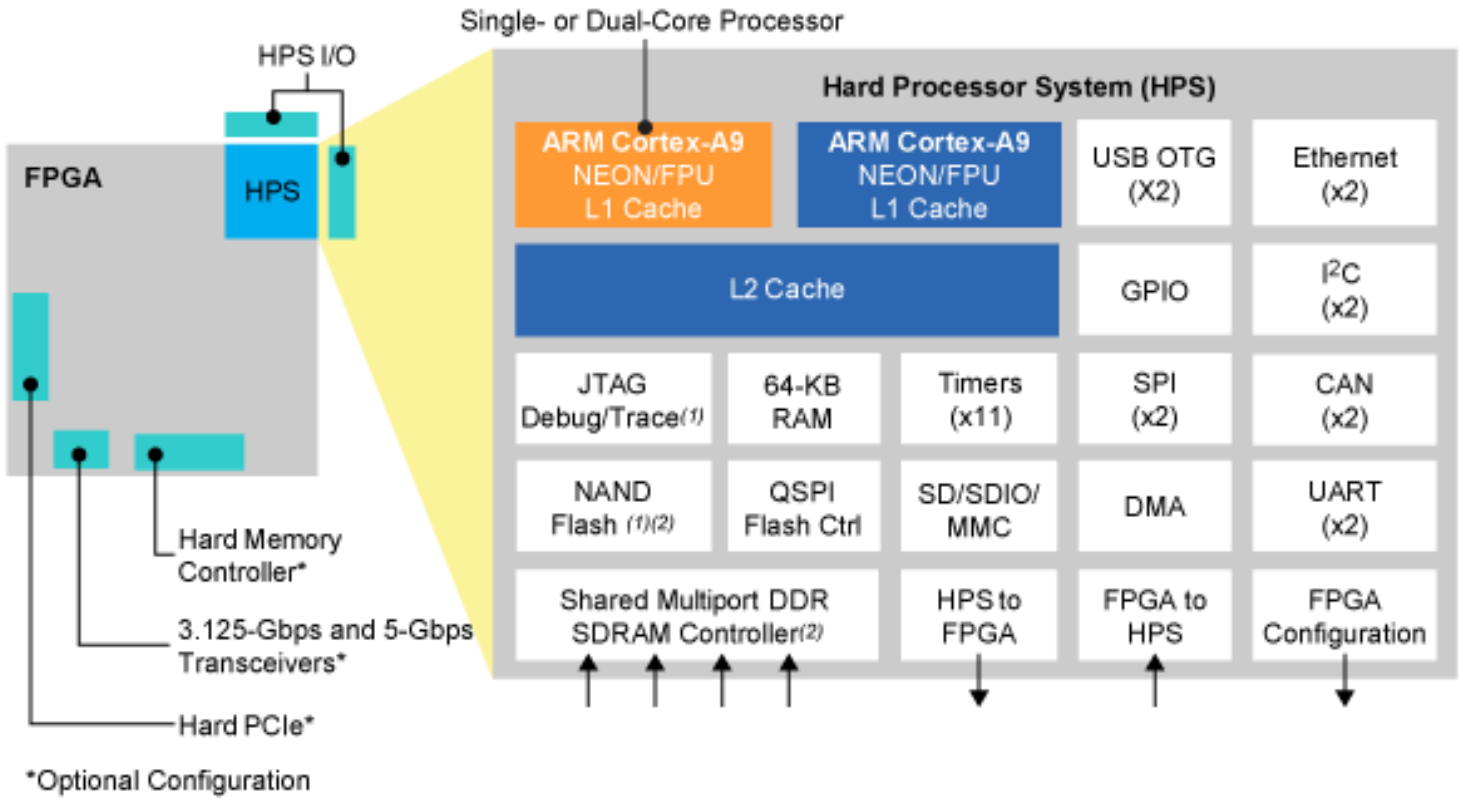
FPGA/HPS Design with Cyclone V

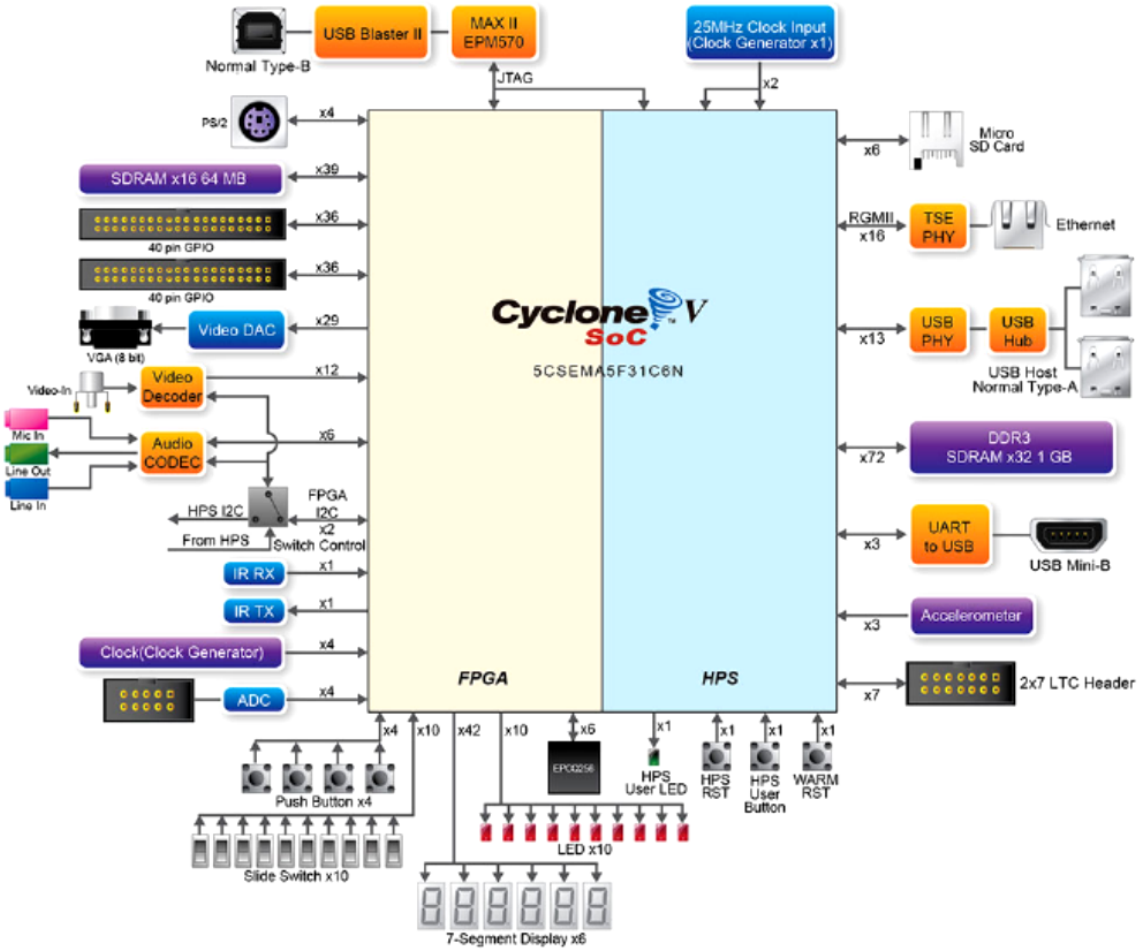
COE838: Lab4 & MD5 Project

General Overview - FPGA/HPS



Source: Altera

General Overview - FPGA/HPS



Source: Altera

FPGA - Avalon I/F & Interconnect

•Hard vs Soft IPs

Examples:

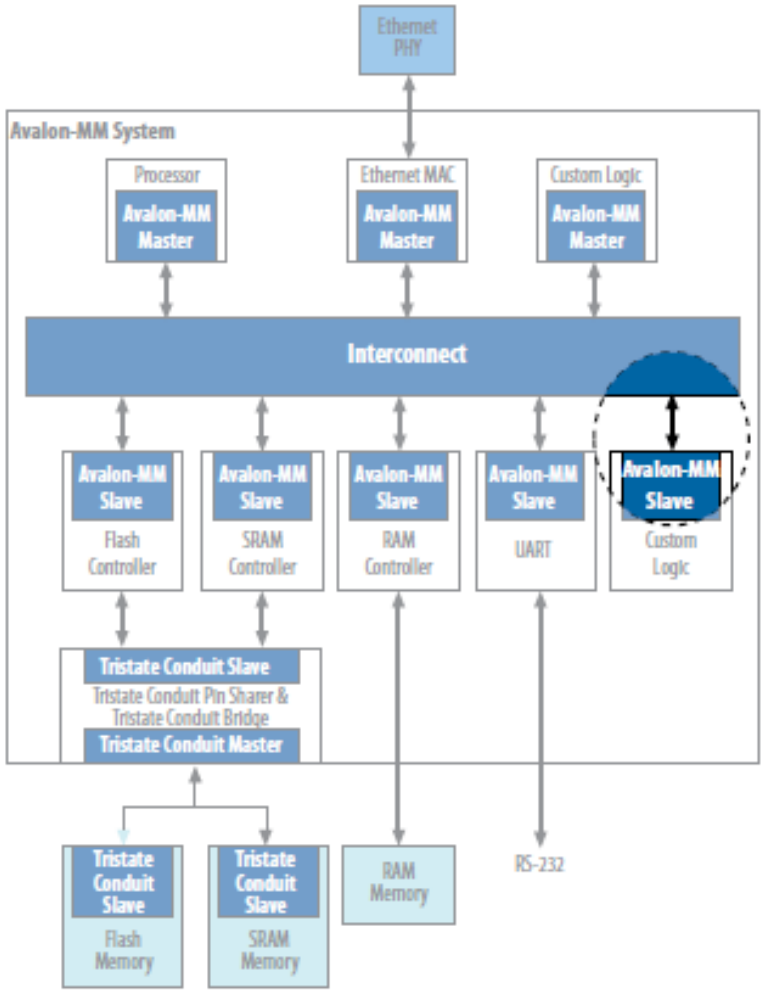
CPUs : NIOS and ARM

Memory Controllers: DDR3 SDRAM, SRAM etc

I/Os: PCI, Ethernet etc

Clock & signal generation : PLLs, DLLs, etc

Arithmetic Logic : MAC units, multipliers etc

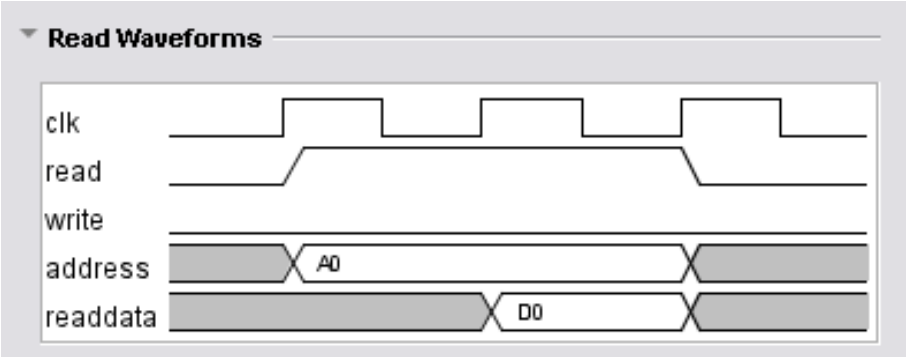


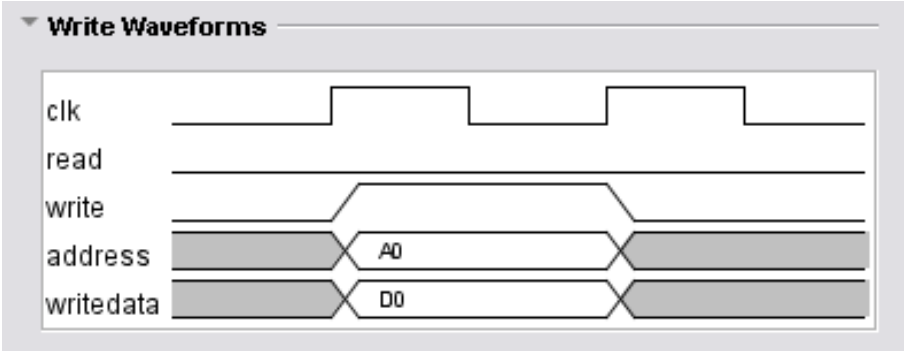
Source: Altera

Avalon MM Slave Interface

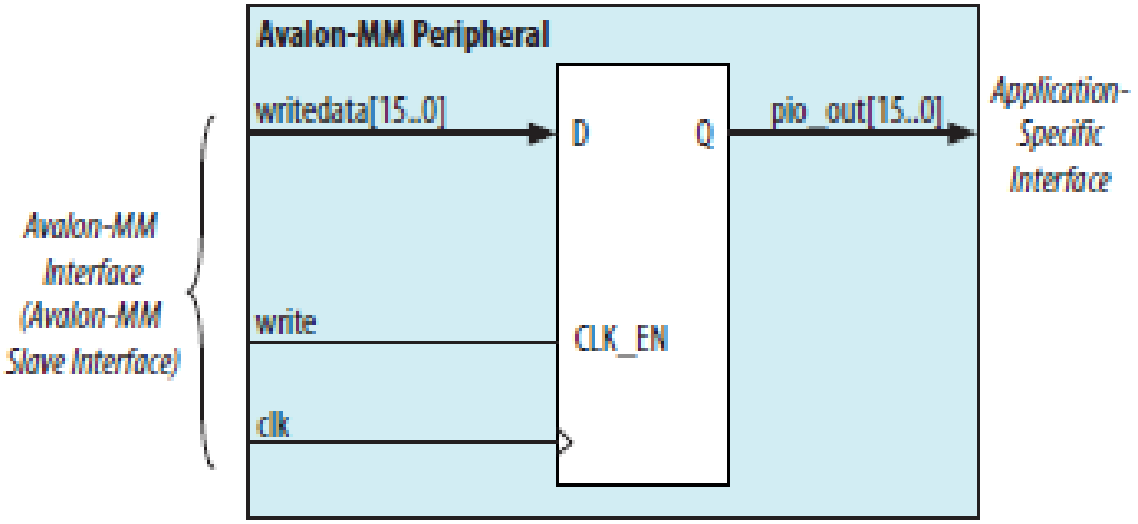
Table I: Common Avalon Bus Signals

Name	Width	Direction	Comments
avs_address	<= 64 bits	Input	Address of slave being accessed
avs_read	1 bit	Input	Read operation requested
avs_write	1 bit	Input	Write operation requested
avs_readdata	8, 16, 32 or 64 bits	Output	Data read from slave
avs_writedata	8, 16, 32 or 64 bits	Input	Data to be written to slave

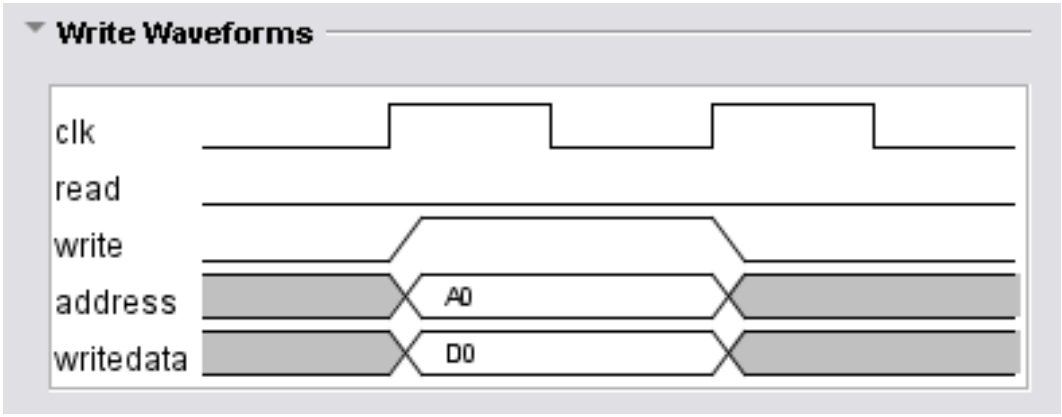




Avalon MM Slave Interface

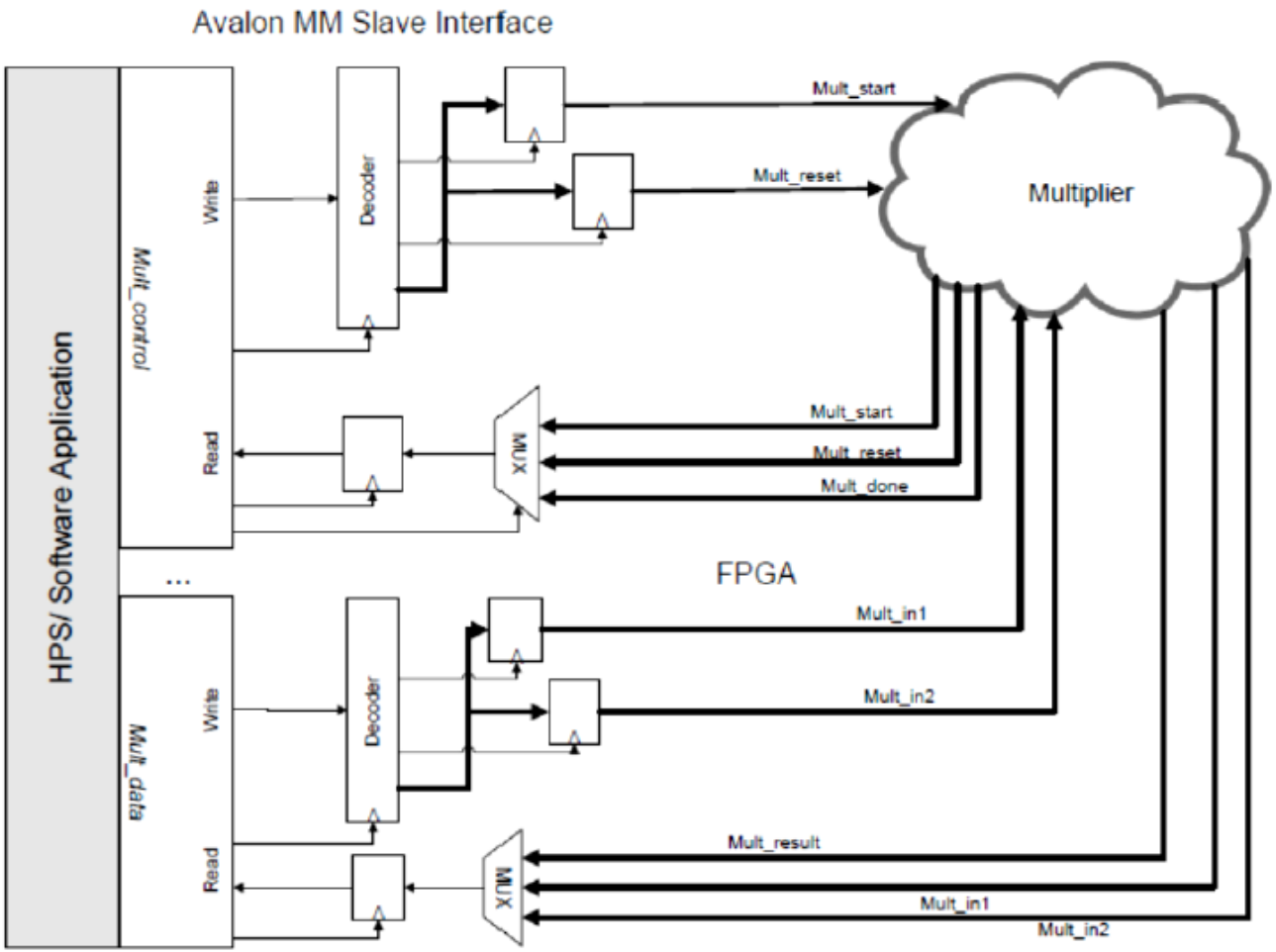


-Most simple Avalon MM Slave example
-The case for lab 3



Source: Altera

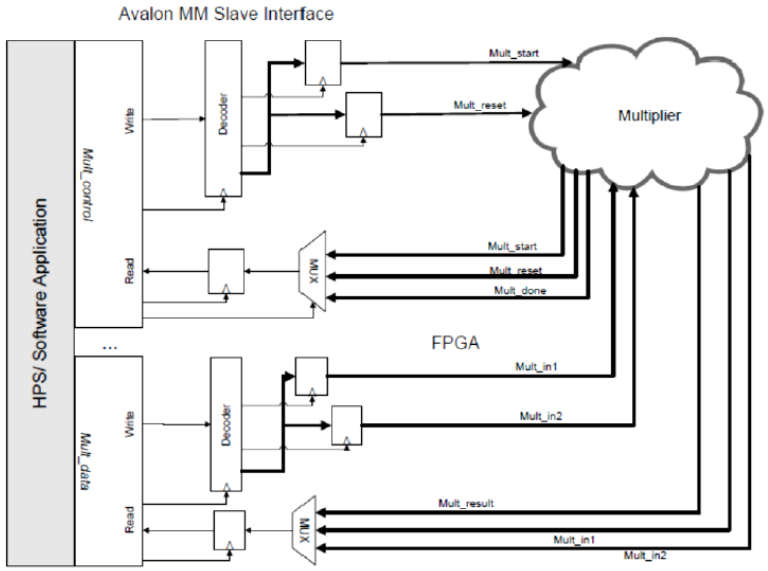
Avalon MM Slave Interface



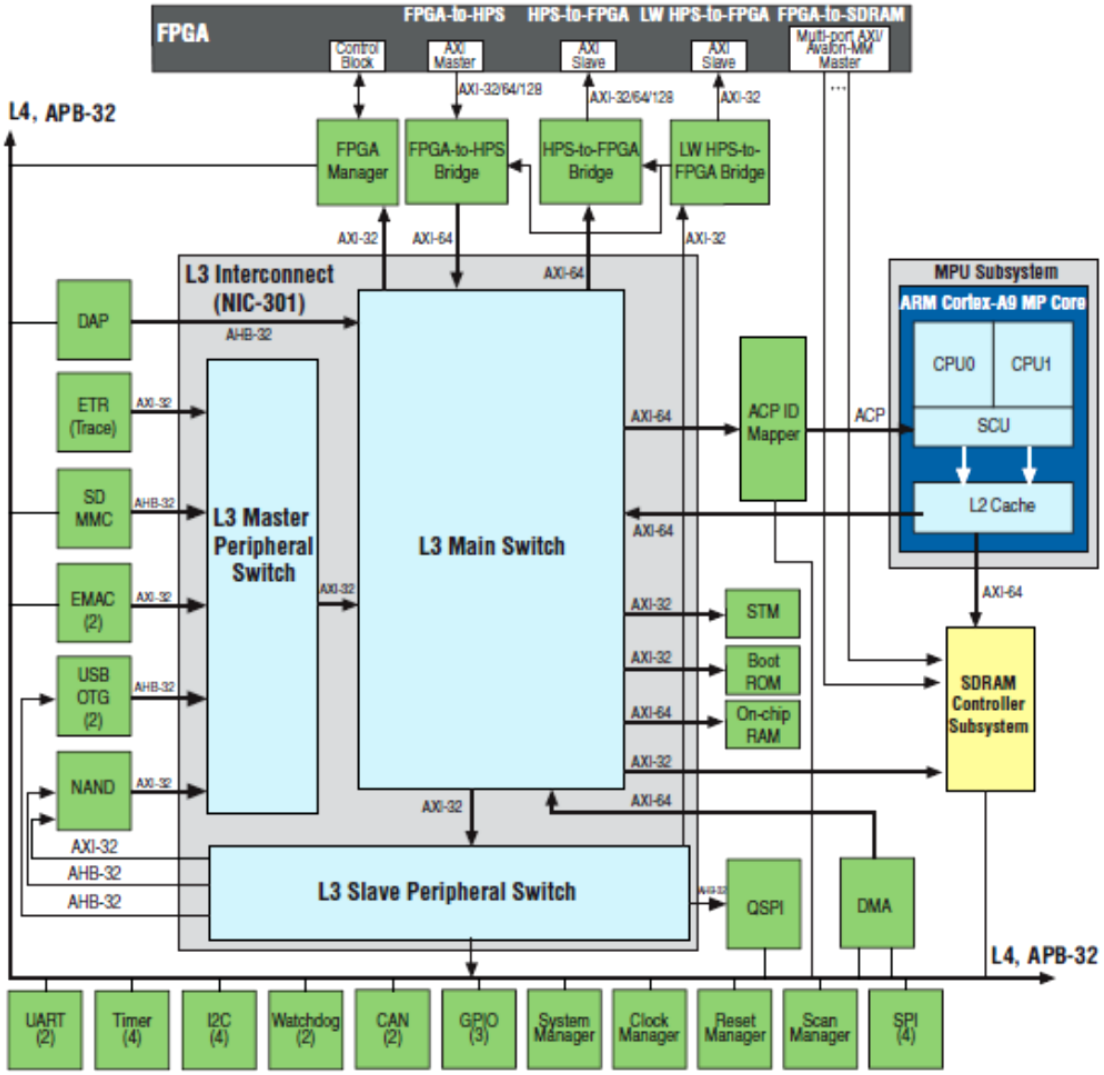
Avalon MM Slave Interface

```
IF(reset = '1')THEN
   avs_s0_readdata <= (OTHERS => '0');
   in1 <= (OTHERS => '0');
   in2 <= (OTHERS => '0');

ELSIF(rising_edge(clk))THEN
   IF(avs_s0_read = '1')THEN
      CASE avs_s0_address IS
         WHEN "0000" =>
            avs_s0_readdata <= mult_result;
         WHEN "0001" =>
            avs_s0_readdata <= in1;
         WHEN "0010" =>
            avs_s0_readdata <= in2;
         WHEN OTHERS =>
            avs_s0_readdata <= (OTHERS => '0');
      END CASE;
   ELSIF(avs_s0_write = '1')THEN
      CASE avs_s0_address IS
         WHEN "0000" =>
            in1 <= "0000000000000000" & avs_s0_writedata(15 DOWNTO 0);
         WHEN "0001" =>
            in2 <= "0000000000000000" & avs_s0_writedata(15 DOWNTO 0);
         WHEN OTHERS =>

         END CASE;
```

```
entity mult_data is
   port (
      avs_s0_address   : in  std_logic_vector(3 downto 0)  := (others => '0'); --
      avs_s0_read      : in  std_logic                     := '0';             --
      avs_s0_write     : in  std_logic                     := '0';             --
      avs_s0_readdata  : out std_logic_vector(31 downto 0);                    --
      avs_s0_writedata : in  std_logic_vector(31 downto 0) := (others => '0'); --
      clk              : in  std_logic                     := '0';             --
      reset            : in  std_logic                     := '0';             --
      mult_in1         : out std_logic_vector(31 downto 0);                    -- mu
      mult_in2         : out std_logic_vector(31 downto 0);                    -- mu
      mult_result      : in  std_logic_vector(31 downto 0) := (others => '0') -- mul
   );
end entity mult_data;
```



HPS: ARM Cortex-A9

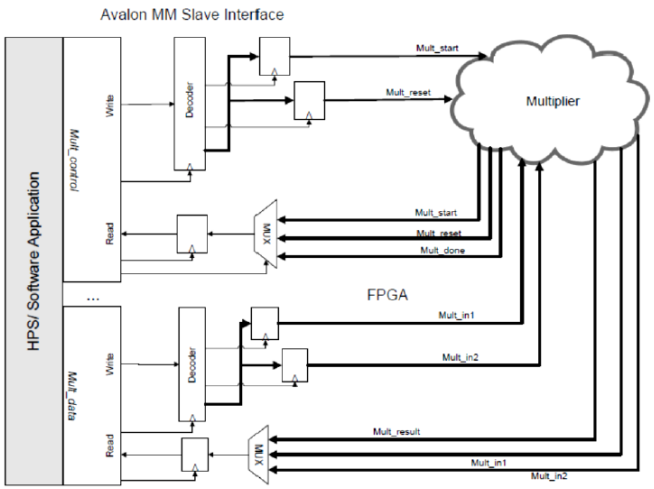


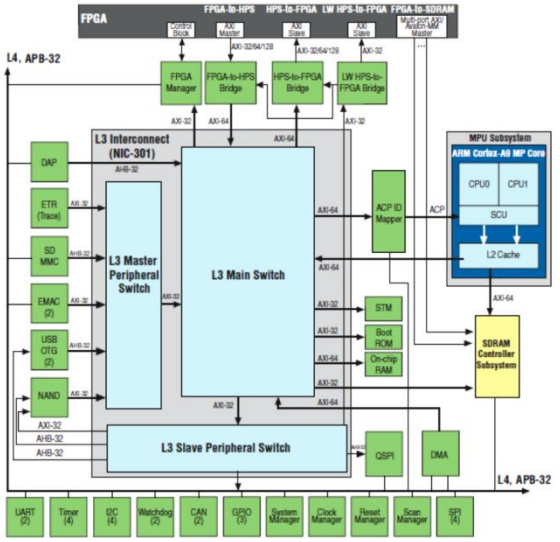
Source: Altera

HPS: ARM Cortex-A9

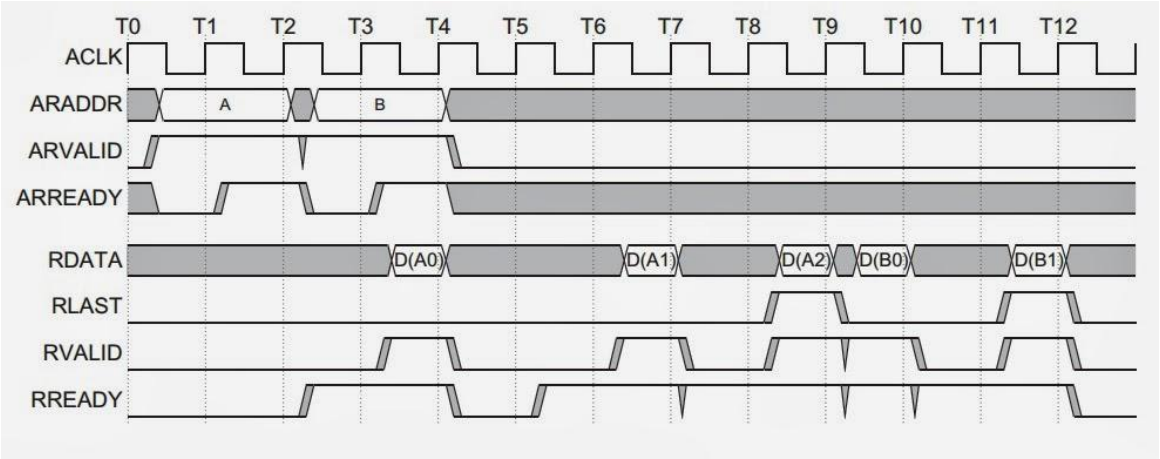
```
//initialize the addresses
mult_control = virtual_base + ((uint32_t)( MULT_CONTROL_0_BASE)
mult_data = virtual_base + ((uint32_t)(MULT_DATA_0_BASE));
```

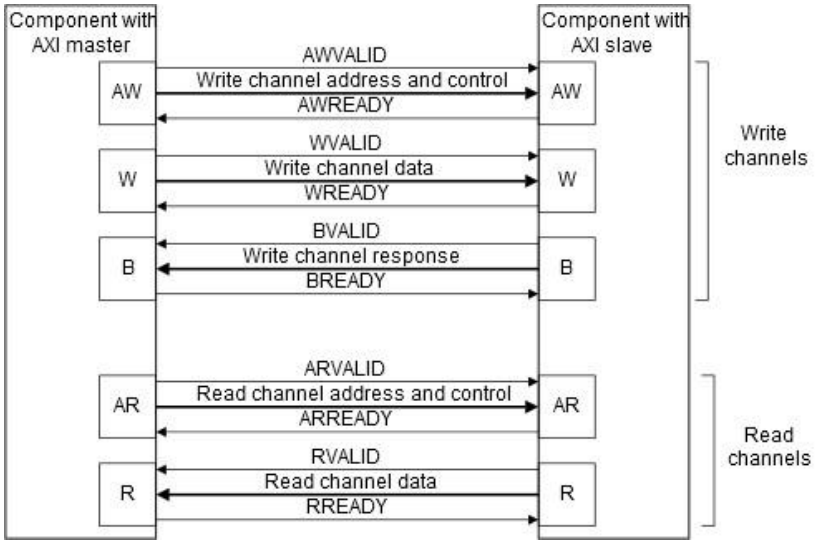
```
void copy_output(){
    uint32_t word, op1, op2;
    //wait for done
    printf("waiting for done\n");
    while(!(alt_read_word(mult_control+2) & 0x1));

    printf("conversion done\n");
    word = alt_read_word(mult_data+0);
    op1 = alt_read_word(mult_data+1);
    op2 = alt_read_word(mult_data+2);
    printf("0x%08x * 0x%08x = 0x%08x. [Expected] 0x%08x\n",
    if(word == (op1*op2)){
        printf("[SUCCESSFUL]\n");
        success++;
    }else{
```

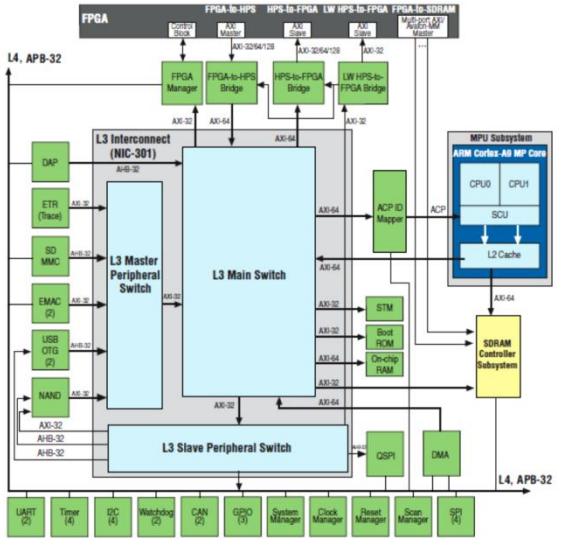




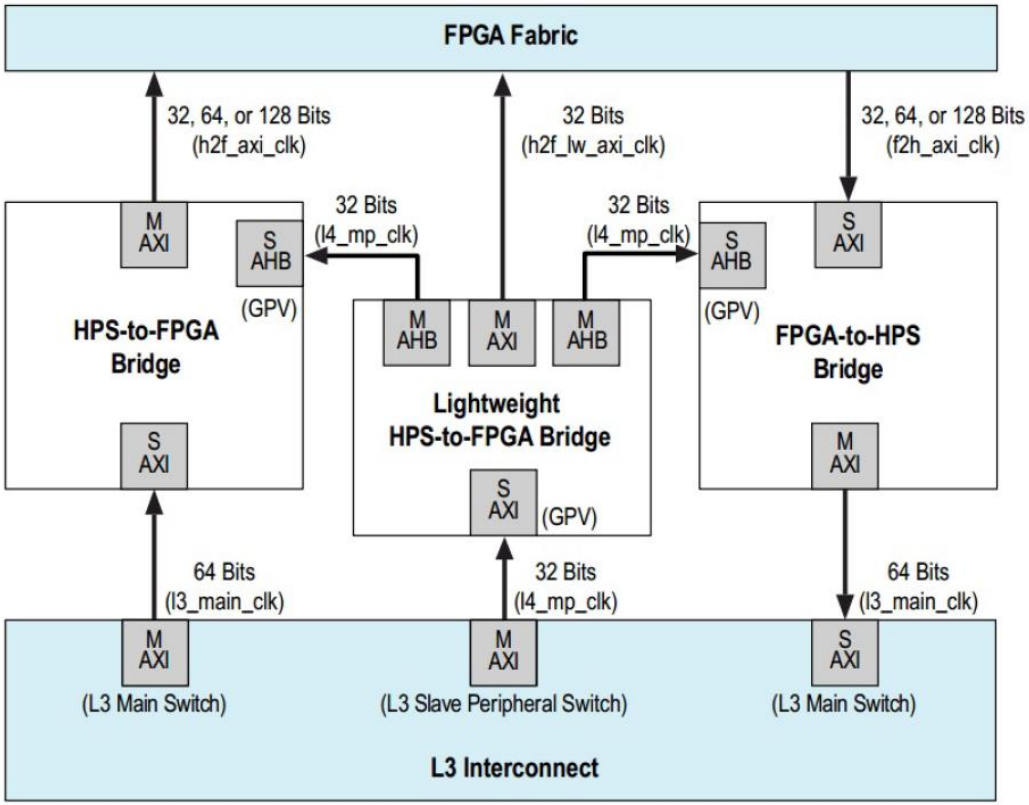
HPS: ARM Cortex-A9







HPS – FPGA Bridges



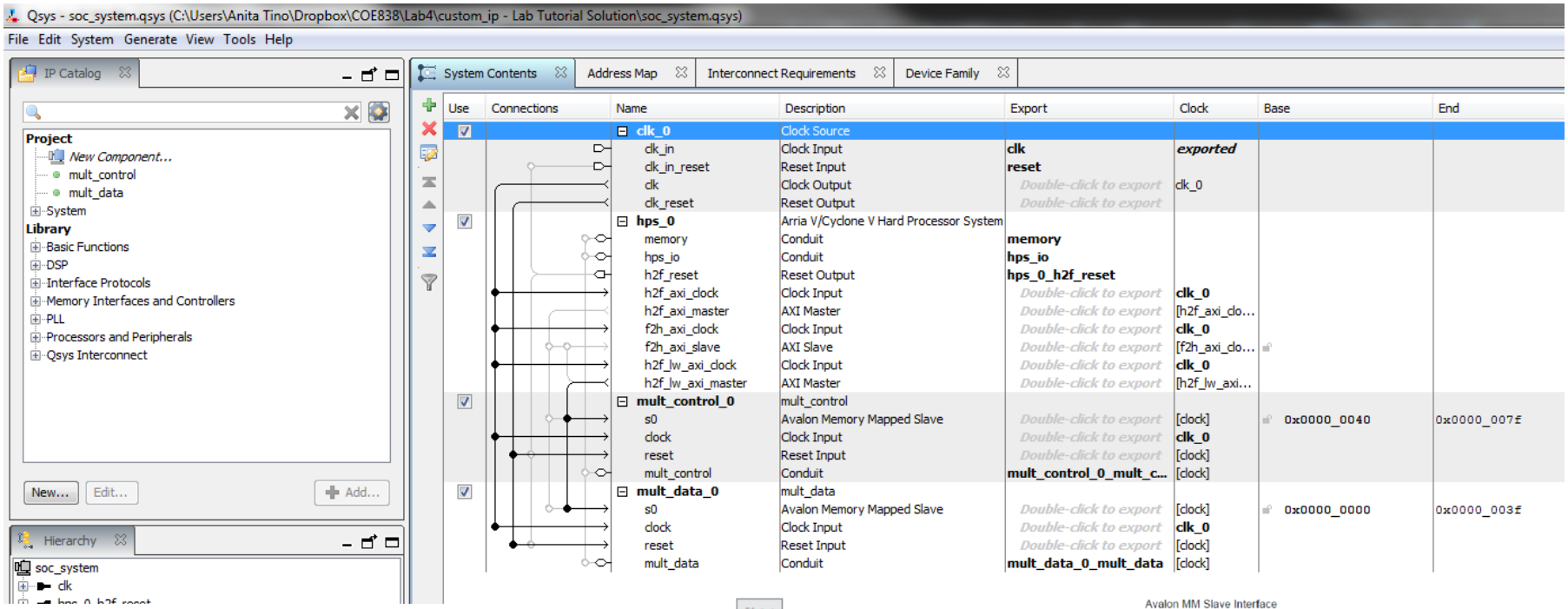
Source: Altera

-Modifies data and clock signals to support transportation (protocols, clocking etc) between components

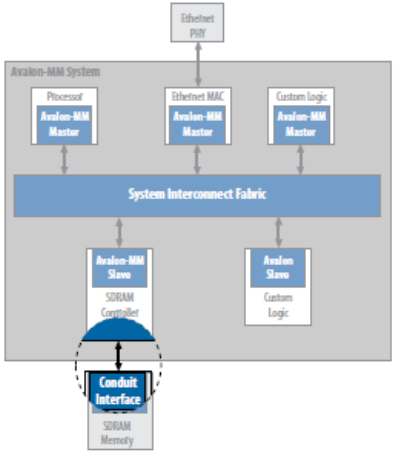
-Qsys: Use this to design your system consisting of IPs and Avalon slaves, specify bridges connections etc.

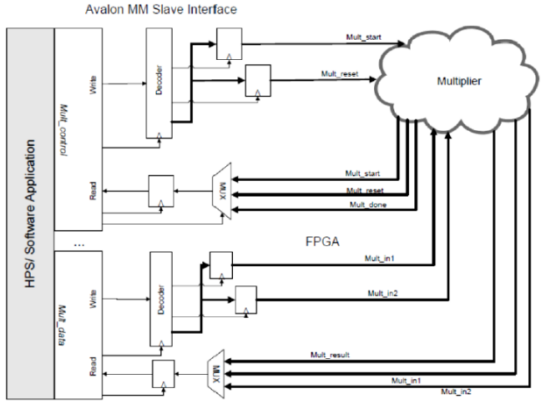
-Qsys will generate system interconnections, adaptation requirements between components etc

Qsys: Putting it all together



Conduits drive signals off-chip





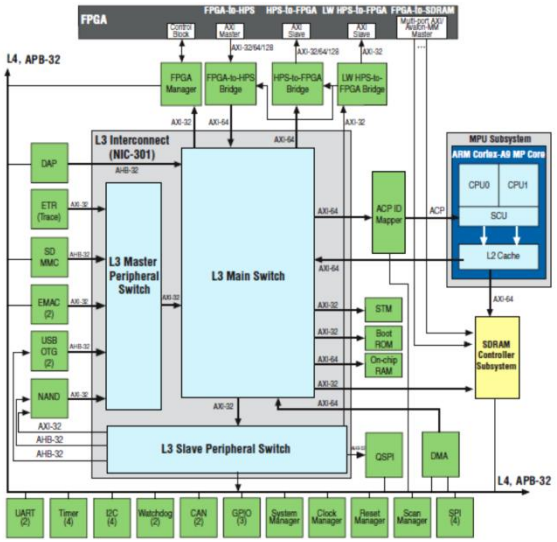
Top-Level VHDL: Putting it all together

```
            hps_io_hps_io_emac1_inst_TX_CTL    => HPS_ENET_TX_EN,    --                          .hps_io_emac1_inst_TX_CTL
            hps_io_hps_io_emac1_inst_RX_CLK    => HPS_ENET_RX_CLK,    --                          .hps_io_emac1_inst_RX_CLK
            hps_io_hps_io_emac1_inst_RXD1      => HPS_ENET_RX_DATA(1),      --                          .hps_io_emac1_inst_RXD1
            hps_io_hps_io_emac1_inst_RXD2      => HPS_ENET_RX_DATA(2),      --                          .hps_io_emac1_inst_RXD2
            hps_io_hps_io_emac1_inst_RXD3      => HPS_ENET_RX_DATA(3),      --                          .hps_io_emac1_inst_RXD3
            hps_io_hps_io_sdio_inst_CMD        => HPS_SD_CMD,        --                          .hps_io_sdio_inst_CMD
            hps_io_hps_io_sdio_inst_D0         => HPS_SD_DATA(0),         --                          .hps_io_sdio_inst_D0
            hps_io_hps_io_sdio_inst_D1         => HPS_SD_DATA(1),         --                          .hps_io_sdio_inst_D1
            hps_io_hps_io_sdio_inst_CLK        => HPS_SD_CLK,        --                          .hps_io_sdio_inst_CLK
            hps_io_hps_io_sdio_inst_D2         => HPS_SD_DATA(2),         --                          .hps_io_sdio_inst_D2
            hps_io_hps_io_sdio_inst_D3         => HPS_SD_DATA(3),         --                          .hps_io_sdio_inst_D3
            hps_io_hps_io_usb1_inst_D0         => HPS_USB_DATA(0),         --                          .hps_io_usb1_inst_D0
            hps_io_hps_io_usb1_inst_D1         => HPS_USB_DATA(1),         --                          .hps_io_usb1_inst_D1
            hps_io_hps_io_usb1_inst_D2         => HPS_USB_DATA(2),         --                          .hps_io_usb1_inst_D2
            hps_io_hps_io_usb1_inst_D3         => HPS_USB_DATA(3),         --                          .hps_io_usb1_inst_D3
            hps_io_hps_io_usb1_inst_D4         => HPS_USB_DATA(4),         --                          .hps_io_usb1_inst_D4
            hps_io_hps_io_usb1_inst_D5         => HPS_USB_DATA(5),         --                          .hps_io_usb1_inst_D5
            hps_io_hps_io_usb1_inst_D6         => HPS_USB_DATA(6),         --                          .hps_io_usb1_inst_D6
            hps_io_hps_io_usb1_inst_D7         => HPS_USB_DATA(7),         --                          .hps_io_usb1_inst_D7
            hps_io_hps_io_usb1_inst_CLK        => HPS_USB_CLKOUT,        --                          .hps_io_usb1_inst_CLK
            hps_io_hps_io_usb1_inst_STP        => HPS_USB_STP,        --                          .hps_io_usb1_inst_STP
            hps_io_hps_io_usb1_inst_DIR        => HPS_USB_DIR,        --                          .hps_io_usb1_inst_DIR
            hps_io_hps_io_usb1_inst_NXT        => HPS_USB_NXT,        --
            hps_0_h2f_reset_reset_n           => reset_reset_n,
            mult_data_0_mult_data_m_result     => mult_output_result, -- mult_output_0_mult_output.result
            mult_control_0_mult_control_m_done => "00000000000000000000000000000000" & done,    --<<<<<                          .done
            mult_data_0_mult_data_m_in1        => in1,      --   mult_input_0_mult_input.in1
             mult_data_0_mult_data_m_in2       => in2,      --                           .in2
            mult_control_0_mult_control_m_start => mult_input_start,    --                        .start
            mult_control_0_mult_control_m_reset => mult_input_reset     --                         .reset
        );

        m0 : mult_unit
    PORT MAP( clk => CLOCK_50, reset => mult_input_reset(0), enable => mult_input_start(0), mult_a => in1(15 DOWNTO 0), mult_b => in2(15 DOWNTO 0),
            mult_done => done, mult_result => mult_output_result);


END Behaviour;
```

HPS Software: Putting it all together

```
void copy_output(){
    uint32_t word, op1, op2;
    //wait for done
    printf("waiting for done\n");
    while(!(alt_read_word(mult_control+2) & 0x1));

    printf("conversion done\n");
    word = alt_read_word(mult_data+0);
    op1 = alt_read_word(mult_data+1);
    op2 = alt_read_word(mult_data+2);
    printf("0x%08x * 0x%08x = 0x%08x. [Expected] 0x%08x\n",  op1, op2, word, (op1*op2));
    if(word == (op1*op2)){
       printf("[SUCCESSFUL]\n");
       success++;
    }else{
```

```
---------------- Iteration 27 ------------------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x0000001b * 0x0000001c = 0x000002f4. [Expected] 0x000002f4
[SUCCESSFUL]
------------------------------------------------
---------------- Iteration 28 ------------------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x0000001c * 0x0000001d = 0x0000032c. [Expected] 0x0000032c
[SUCCESSFUL]
------------------------------------------------
---------------- Iteration 29 ------------------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x0000001d * 0x0000001e = 0x00000366. [Expected] 0x00000366
[SUCCESSFUL]
------------------------------------------------
---------------- Iteration 30 ------------------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x0000001e * 0x0000001f = 0x000003a2. [Expected] 0x000003a2
[SUCCESSFUL]
------------------------------------------------
[TEST PASSED] 30/30
```



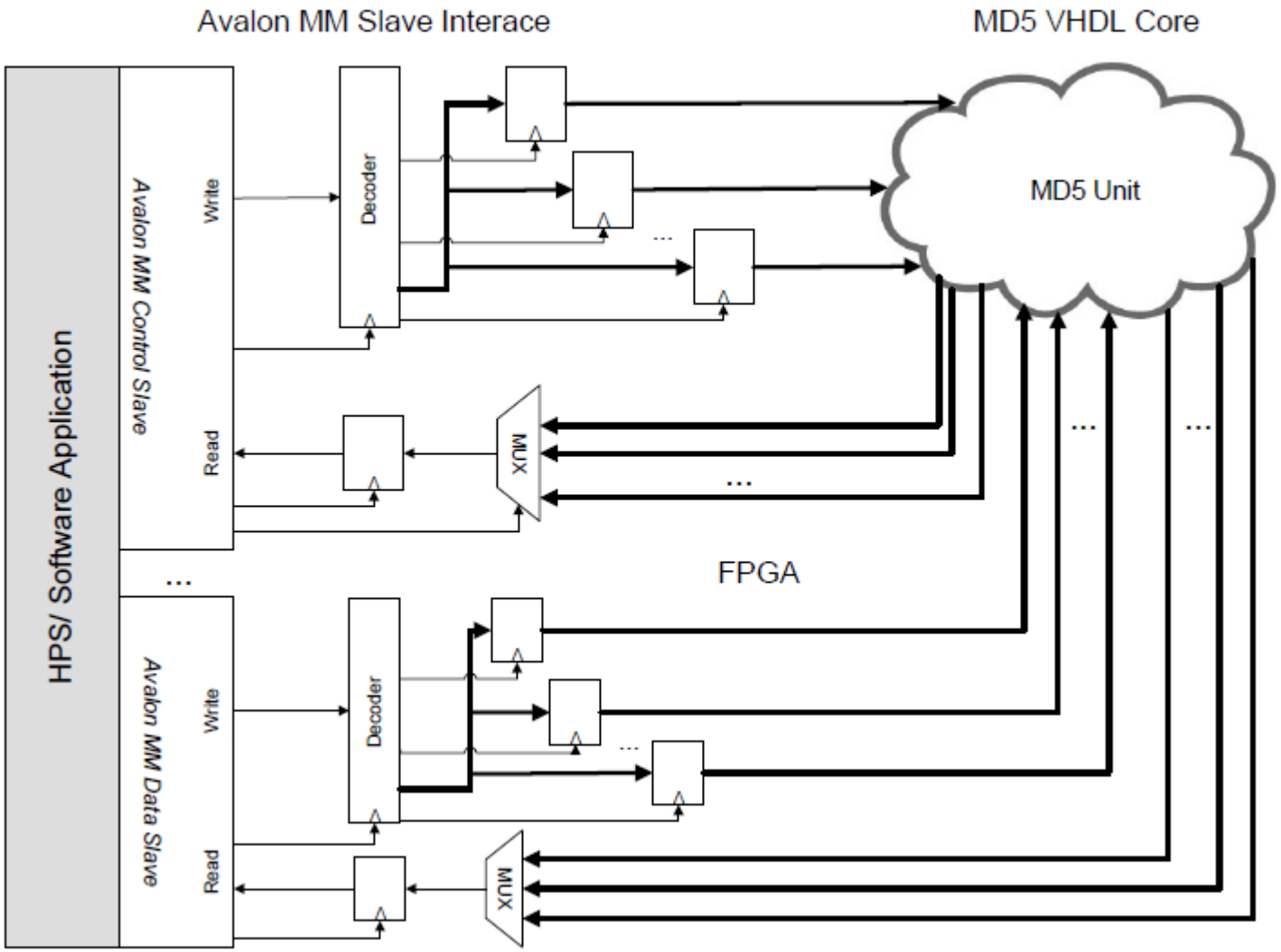
Project :
MD5 Decryption SoC Design

MD5 Algorithm

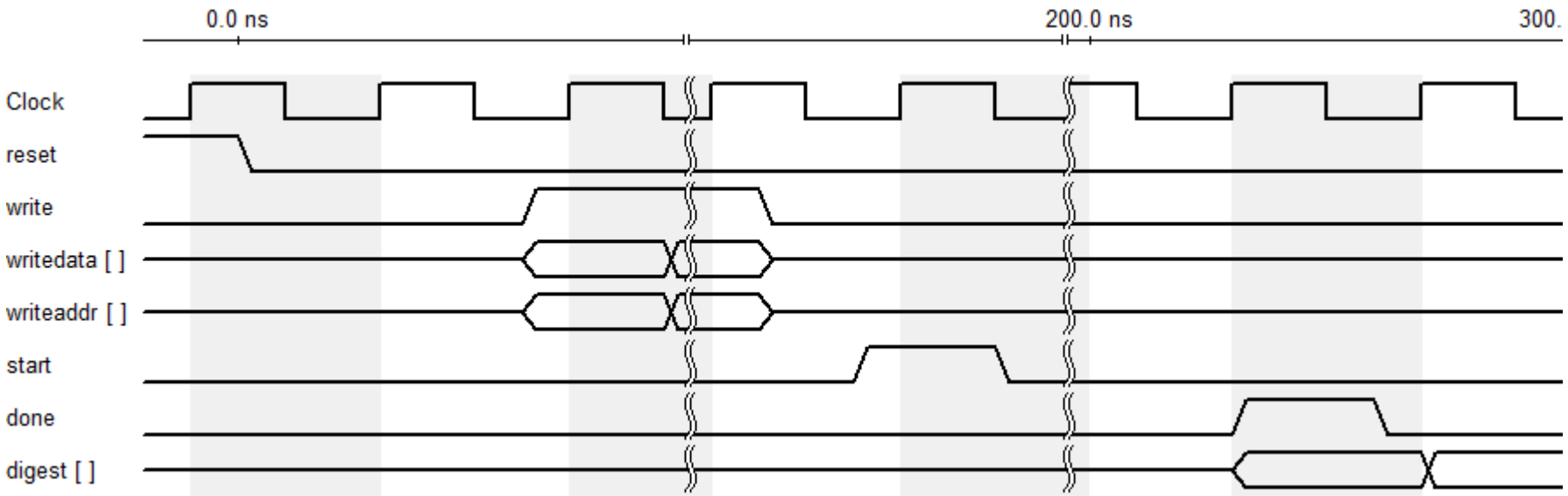
```
//Note: All variables are unsigned 32 bit and wrap
var int[64] s, K
//s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7,
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4,
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6,

//K constants
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db,
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613,
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1,
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e,
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51,
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681,
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87,
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9,
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122,
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60,
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085,
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8,
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7,
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d,
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314,
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb,

//Initialize variables:
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D
```

```
//Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15

//Initialize hash value for this chunk:
    var int A := a0      var int B := b0
    var int C := c0      var int D := d0

//Main loop:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            F := C xor (B or (not D))
            g := (7×i) mod 16
        dTemp := D
        D := C
        C := B
        B := B + leftrotate((A + F + K[i] + M[g]), s[i])
        A := dTemp
    end for
//Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 //(Output is in little-endian)

//leftrotate function definition
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

MD5 Avalon Slave Interface



MD5 Timing Characteristics



MD5 Project

- **Analyze** the MD5 Core – Obtain a thorough understanding
  - VHDL RTL analysis
  - Test bench => 1 core vs 32 cores
  - Timing properties in ModelSim & lab manual
- **Design Avalon MM Interface**
- **Design HPS Software Application**
  - Generate Messages
  - Send *constant* data, send message
  - Receive digest when complete
  - Calculate hash time, # of hashes, hash rate, correct answer etc
- Parallel vs Sequential
- **Formal report** must follow specifications
- **Bonus** – Pure software vs hardware