COE838: System-on-Chip Design

HPS/FPGA Interconnection

Lab 3 and 4, Project manual
DE1-SoC Datasheets [online]
Flynn and Luk book – Chapter 3
- Open Quartus, include top level files which represent your SoC system.
- Use Qsys to integrate an HPS, IPs, and their respective connections to the lw axi bus
• Generate .h files – obtain addresses for components, to be used for memory mapping in C application
- Port map top level with soc_system created in QSys
- Pin assts, tcl scripts etc for IO and porting HPS and FPGA
- Compile/Synthesize
- Start a C project
- Create C application (memory map) using addresses generated by Qsys and NIOS
- Compile & copy binary
- Start Minicom or DS-5 terminal
- Access Yocto linux, copy .x
- Program .sof on board (i.e. Bitstream)
- Execute the binary using host terminal

------------------- Iteration 27 -------------------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x0000001b * 0x00000001c = 0x000002f4. [Expected] 0x000002f4
[SUCCESSFUL]

------------------- Iteration 28 -------------------
Memory-Mapped IO

```c
void *virtual_base;

#define LWHPS2FPGA_BASE 0xff200000

volatile uint32_t *h2p_lw_led_addr = NULL;

//open the /dev/mem to access the FPGA space for reading and writing
if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
    printf( "ERROR: could not open "/dev/mem"...
" );
    return( 1 );
}

//map the virtual memory space to virtual_base, that is 2MB in size
//(0x00200000), at address LWHPS2FPGA_BASE
virtual_base = mmap( NULL, LW_SIZE, ( PROT_READ | PROT_WRITE ),
                    MAP_SHARED, fd, LWHPS2FPGA_BASE);

// map the address space for the LED and HEX registers into user space so
//we can interact with them.. virtual_base + the offset of your IP component
h2p_lw_led_addr= virtual_base + ((uint32_t)(LEDPIO_BASE));
```
Memory-Mapped IO

```c
volatile uint32_t *h2p_lw_led_addr = NULL;

// map the address space for the LED and HEX registers into user space so we can interact with them. virtual_base + the offset of your IP component
h2p_lw_led_addr = virtual_base + ((uint32_t)(LED_PIO_BASE));

alt_write_word(h2p_lw_led_addr, 0x3FF);
```
HPS/FPGA - ARM Cortex-A9

Source: Altera
Avalon Bus

- Interconnect fabric inside FPGA (Altera)
- Used to connect master-slaves as required
- Generates the necessary “busses” using the fabric to make these connections
- Separates data in from data out
  - Uses multiplexers
Avalon Bus

- Follows a protocol
- You have an IP, however it does not necessarily know how to communicate with the FPGA fabric
- Need an Avalon-MM Slave (or Master) for controlling and w/r data
- Components we used in Lab3 were provided with Avalon-MM interfaces
Avalon Bus

- Follows a protocol
- You have an IP, however it does not necessarily know how to communicate with the FPGA fabric
- Need an Avalon-MM Slave (or Master) for controlling and I/O data
Avalon Bus – Custom IPs

- Need to create an Avalon-MM interface for your IP with the Avalon fabric.
- Can also create a “wrapper” for integrating additional signals, not necessarily provided by your IP.
Custom IP in HPS/FPGA

- Use IP Cores in Quartus to generate a custom multiplier (lab4)
- How do we determine when multiplication is complete from HPS side?
WRAPPERS FOR CUSTOM IPS
Conduits drive signals off-chip

U0 : soc_system
PORT MAP(

mult_data_0_mult_data_m_result => mult_output_result,
mult_control_0_mult_control_m_done => "0000000000000000000000000000000" & done,
mult_data_0_mult_data_m_in1 => in1,
mult_data_0_mult_data_m_in2 => in2,
mult_control_0_mult_control_m_start => mult_input_start,
mult_control_0_mult_control_m_reset => mult_input_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);

Let's convert lab2a (multiplier) ->

HPS/FPGA CUSTOM IPS
Table I: Common Avalon Bus Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Direction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>avs_address</td>
<td>&lt;= 64 bits</td>
<td>Input</td>
<td>Address of slave being accessed</td>
</tr>
<tr>
<td>avs_read</td>
<td>1 bit</td>
<td>Input</td>
<td>Read operation requested</td>
</tr>
<tr>
<td>avs_write</td>
<td>1 bit</td>
<td>Input</td>
<td>Write operation requested</td>
</tr>
<tr>
<td>avs_readdata</td>
<td>8, 16, 32 or 64 bits</td>
<td>Output</td>
<td>Data read from slave</td>
</tr>
<tr>
<td>avs_writedata</td>
<td>8, 16, 32 or 64 bits</td>
<td>Input</td>
<td>Data to be written to slave</td>
</tr>
</tbody>
</table>
Lab4 - Avalon MM Slave I/F
Common Timing Diagrams

READING & WRITING DATA
Avalon MM Slave Interface
Avalon MM Slave Interface

Table I: Common Avalon Bus Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Direction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>avs_address</td>
<td>&lt;= 64 bits</td>
<td>Input</td>
<td>Address of slave being accessed</td>
</tr>
<tr>
<td>avs_read</td>
<td>1 bit</td>
<td>Input</td>
<td>Read operation requested</td>
</tr>
<tr>
<td>avs_write</td>
<td>1 bit</td>
<td>Input</td>
<td>Write operation requested</td>
</tr>
<tr>
<td>avs_readdata</td>
<td>8, 16, 32 or 64 bits</td>
<td>Output</td>
<td>Data read from slave</td>
</tr>
<tr>
<td>avs_writedata</td>
<td>8, 16, 32 or 64 bits</td>
<td>Input</td>
<td>Data to be written to slave</td>
</tr>
</tbody>
</table>

Read Waveforms

<table>
<thead>
<tr>
<th>Write Waveforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>write</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>readdata</td>
</tr>
<tr>
<td>clk</td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>write</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>writedata</td>
</tr>
</tbody>
</table>
Qsys: Create Interfaces

Conduits drive signals off-chip
Use Qsys to create the Avalon-MM i/f so that your IP core may communicate through the FPGA fabric to the HPS (i.e. IP – MM I/F – Avalon bus - bridge – axi protocol ) and vice versa
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;

    end entity mult_data;

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);
        mult_in2 : out std_logic_vector(31 downto 0);
        mult_result : in std_logic_vector(31 downto 0) := (others => '0');
    );
end entity mult_data;

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface

Avalon MM Slave Interface
On the SW Side...

```c
// 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\n" if(word == (op1*op2)) {
        printf("[SUCCESSFUL]\n");
        success++;
    }else{
        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 <= "00000000000000000000000000000000" & avs_s0_writedata(15 DOWNTO 0);
                    WHEN "0001" => in2 <= "00000000000000000000000000000000" & avs_s0_writedata(15 DOWNTO 0);
                    WHEN OTHERS =>
                        END CASE;
    }
```
HPS: ARM Cortex-A9
HPS - FPGA
HPS – FPGA Bridges

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

- Connect your slaves to the required bus.

- Qsys will make the necessary connections to the bridges for Avalon/AXI compatibility.

Source: Altera
Top-Level VHDL: Putting it all together

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_level is
  port(
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    enable : in STD_LOGIC;
    m_a : in STD_LOGIC_VECTOR(15 downto 0);
    m_b : in STD_LOGIC_VECTOR(15 downto 0);
    m_done : out STD_LOGIC;
    m_result : out STD_LOGIC_VECTOR(15 downto 0)
  );
end top_level;

architecture的行为
begin
  -- 详细代码...
end architecture;
```
HPS Software: Putting it all together

```c
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++;
  }
}
```

---------- Iteration 27 ----------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x00000001b * 0x00000001c = 0x0000002f4. [Expected] 0x0000002f4
[SUCCESSFUL]

---------- Iteration 28 ----------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x00000001c * 0x00000001d = 0x00000032c. [Expected] 0x00000032c
[SUCCESSFUL]

---------- Iteration 29 ----------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x00000001e * 0x00000001e = 0x000000366. [Expected] 0x000000366
[SUCCESSFUL]

---------- Iteration 30 ----------
Reset done. Deasserting signal
Start successful
waiting for done
conversion done
0x00000001f * 0x00000001f = 0x0000003a2. [Expected] 0x0000003a2
[SUCCESSFUL]

[TEST PASSED] 30/30
Lab4 - Avalon MM Slave I/F
Lab 4 – Your Assignment

- Create a custom functioned SoC of your choice
  - Obviously, DO NOT use the multiplier as your design
- Points awarded for creativity
- Divide custom IP is obviously not that creative – just doing the opposite!
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, 12, 17, 22, 7, 12, 17, 22, 7, s[16..31] := { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, s[32..47] := { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, s[48..63] := { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, // K constants
K[ 0.. 3] := { 0x76aa478, 0xe8c7b756, 0x242070db, K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, K[ 8..11] := { 0x8bf1c690, 0xe9b6c764, 0x39b067d1, K[12..15] := { 0x06a09e8e, 0x0cb642b8, 0xe49b69df, K[16..19] := { 0xf3b8b860, 0x5f97937a, 0xf60929e2, K[20..23] := { 0x59df8f49, 0x2431032c, 0xe4a22284, K[24..27] := { 0xe1b178be, 0x35840d5a, 0xe9bca6a5, K[28..31] := { 0x0cb642b8, 0x339b067d, 0x8b709e1f, K[32..35] := { 0xfffa3942, 0x8e9b3779, 0x6b9017d1, K[36..39] := { 0x4d72b8e2, 0x38b070db, 0x0ea1a178, K[40..43] := { 0x9b06c150, 0x2f6b0fba, 0xe4a47060, K[44..47] := { 0x653e3d2d, 0x04e0759e, 0x5d3725f8, K[48..51] := { 0x4e8c3d24, 0x5408f36f, 0x7f880114, K[52..55] := { 0x59b3b8bf, 0x5f167344, 0x4b040309, K[56..59] := { 0x9d2c985e, 0xc9193c0d, 0x8c443690, K[60..63] := { 0xf7537e82, 0xb6f90af1, 0x18b8633f, // Initialize variables:
var int a0 := 0x67452301 // A
var int b0 := 0xefcdbc89 // B
var int c0 := 0x98badcfe // C
var int d0 := 0x01035476 // 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
        endif
        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 Standard Project
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 projects = 2-5% bonus on final COE838 grade
Some Bonus Projects ...

**DE1-SoC MTL**

- Develop an SoC that can be controlled by an Android app running on the DE1-SoC
  - Special report
Some Bonus Projects ...

**Parallella** – making heterogeneous high performance parallel platforms attainable to general public (16 – 64 cores on a board)

- Create a MM application (i.e. video processing, bitcoin etc) for the Parallella, determining statistics and providing comparisons to another platform (i.e. X86)

https://www.youtube.com/watch?v=hFWIC3RF0f8
Some Bonus Projects ...

- **DE1-SoC** – video processing SoC design, inputs a graphic and displays an altered graphic on VGA (or screen)
  - Picture or video filtering using SoCs

- **HLS** – LegUp vs custom design (or IP) to develop same hardware for an SoC on DE1-SoC. Compare various stats of performance, power, area etc
Mini Bonus

+1-2% on project mark - .c application equivalent which performs MD5 decryption. Use pure HPS vs HPS/FPGA vs x86 (compare stats)

+2-5% Implement above + Use h2f bus (64b, 128b) & compare statistics