Specification:

The EXAM function is specified below using its:
1. Pseudocode
2. Table of input/output ports
3. Timing requirements.

1. Pseudocode:

```plaintext
h0 := iv0 = 0xA0
h1 := iv1 = 0xB0
h2 := iv2 = 0xC0
h3 := iv3 = 0xD0
last_block_stored := 0

while (last_block_stored != 1)
do
    wait until src_ready
    r := m
    last_block_stored := last_block

    a0 := h0
    a1 := h1
    a2 := h2
    a3 := h3

    for i from 0 to 127 do
        ki := k[i mod 16]
        ri := r[i mod 4]
        a5 := (24*ri) mod 2^8
        a4 := (a3*a5+a0*a2) mod 2^8
        if (i<64) then
            a3 := (a2*a4) mod 2^8
        else
            a3 := (a4*a4) mod 2^8
        end if
        a2 := a1 <<< (2*i)
        a1 := a0 >>> 6
        a0 := (a5 + ki) mod 2^8
    end for

    h0 := (h0 + a0) mod 2^8
    h1 := (h1 + a1) mod 2^8
    h2 := (h2 + a2) mod 2^8
    h3 := (h3 + a3) mod 2^8
end while

y := h0 || h1 || h2 || h3
done := 1
```
Notation:

- \( m \): 32-bit message block (input)
- \( r \): 32-bit register
- \( y \): 32-bit circuit output (output)
- \( \text{iv0..iv3} \): 8-bit initialization vectors (constants)
- \( a0..a5, h0..h3, k_i, r_i \): 8-bit intermediate values, treated as 8-bit unsigned integers
- \( k[i] \): 8-bit round constants: \( k[0]..k[15] \). Assume that \( k[i] = i*16+15 \).
- \( r[i] \): bytes of the 32-bit register \( r \), where \( r[0] \) represents the least significant byte of \( r \), and \( r[3] \) represents the most significant byte of \( r \).

Operations:

- \( X <<< Y \): rotation of \( X \) to the left by the number of positions given in \( Y \)
- \( X >>> Y \): rotation of \( X \) to the right by the number of positions given in \( Y \)
- \( X \| Y \): \( X \) concatenated with \( Y \).

2. Table of input/output ports:

<table>
<thead>
<tr>
<th>Port</th>
<th>Mode</th>
<th>Width</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>Input</td>
<td>1</td>
<td>System clock.</td>
</tr>
<tr>
<td>reset</td>
<td>Input</td>
<td>1</td>
<td>Asynchronous system reset.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>32</td>
<td>32-bit message block.</td>
</tr>
<tr>
<td>src_ready</td>
<td>Input</td>
<td>1</td>
<td>Control signal indicating that the source is ready. Must remain active until source is read.</td>
</tr>
<tr>
<td>src_read</td>
<td>Output</td>
<td>1</td>
<td>Control signal confirming that the source was read. Active for one clock cycle.</td>
</tr>
<tr>
<td>last_block</td>
<td>Input</td>
<td>1</td>
<td>Control signal indicating the last block of the message.</td>
</tr>
<tr>
<td>done</td>
<td>Output</td>
<td>1</td>
<td>Control signal indicating that the output is ready.</td>
</tr>
<tr>
<td>y</td>
<td>Output</td>
<td>32</td>
<td>Output ( y = h0 | h1 | h2 | h3 ).</td>
</tr>
</tbody>
</table>

3. Timing Requirements:

Assume that

- one clock cycle is used for the once-per-message initialization: \( h0 = \text{iv0}; h1 = \text{iv1}; h2 = \text{iv2}; h3 = \text{iv3} \);
- one clock cycle is used for the once-per-block initialization: \( a0 = h0; a1 = h1; a2 = h2; a3 = h3 \);
- one round of the main for-loop of the pseudocode executes in one clock cycle; there are a total of 128 rounds.
- one clock cycle is used for the once-per-block finalization: \( h0 = h0 + a0; h1 = h1 + a1; h2 = h2 + a2; h3 = h3 + a3 \);
Tasks:

Task 1

Draw a block diagram of the Datapath of the EXAM circuit, using medium complexity components corresponding to the operations used in the pseudocode. Clearly specify
- names, widths and directions of all buses
- names, widths and directions of all inputs and outputs of the logic components.

Assume that one round of the main for-loop of the pseudocode executes in one clock cycle.

Minimize the number of control signals to be generated by the Control Unit.

Mark the critical path in your circuit.

Task 2

Draw an interface between the Control Unit and the Datapath

Show the names and widths of all signals forming this interface.

Task 3

Draw an ASM chart of a controller capable of performing computations described in the pseudocode. Translate each action and condition in your ASM chart to the values of the corresponding control and status signals.

Task 4

Develop RTL VHDL code for your entire circuit.

Task 5

Write a testbench for your circuit, and debug any possible errors in your RTL code.

Task 6

Synthesize and implement your solution using
- the smallest device of the Spartan 3 family
- the fastest speed grade available to you
- Xilinx XST for synthesis
- default options of tools.

Locate and include in your report the following results:
- FPGA device used
- minimum clock period after synthesis
- maximum clock frequency after synthesis
- minimum clock period after placing & routing
- maximum clock frequency after placing & routing
- number of CLB slices
- number of LUTs
- number of flip-flops
- number of I/O blocks.
Task 7

Verify the operation of your circuit using post-synthesis and timing simulation.

Deliverables:

1. Block diagram of the Datapath (may be hand-drawn and scanned). Please remember to mark-up the critical path!
2. Interface with the division into the Datapath and the Controller (may be hand-drawn and scanned)
3. ASM chart of the Controller (may be hand-drawn and scanned)
4. All synthesizable source codes, including full source codes for the Datapath, the Controller, and the Top-Level Circuit
5. Testbench for the Top-Level Circuit, and optionally testbenches for lower-level units (Datapath and Controller). Waveforms from functional simulation in .awf format and as a single screenshot (PDF or JPG) proving correct operation.
7. Waveforms from the post-synthesis simulation and timing simulation in .awf format and as a single screenshot (PDF or JPG) proving correct operation.
8. Full reports generated by tools after
   • Synthesis
   • Implementation
   • Static timing analysis.