LAB-5

Part 1: 1 week long, individual, submit hardcopy to A-206
due: May 16th, 2016, Monday, 17:30

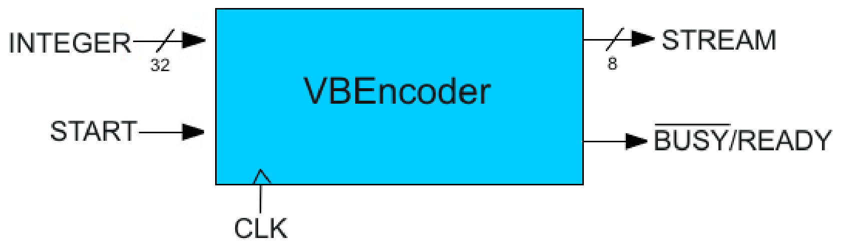
Part 2: 2 weeks long, groupwork, submit via COW
due: Sunday, May 22 , 2016, 23:55hrs

Introduction

Nowadays most of the new technology is focusing on dealing with big data. One of the common application areas is information retrieval. In information retrieval compression has a vital importance because there is a huge data that has to be processed and we need to be quick. So, main motivation in information retrieval is to compress data so that dictionary is small enough to keep in main memory. With this way each look up operation takes much less time. One approach to do this is variable length encoding.

Variable length encoding is used by many commercial/research systems. It is used to save space by representing small numbers with a few bits. Unsigned integer values are stored in 4 bytes in computer systems. Using this encoding, we will be able to store small integer values with less than 4 bytes.

In this assignment, you are going to implement this specific type of encoding system. You will take a 32-bit integer number and convert it to a byte stream with smaller length. Then list the new stream 1 byte per clock. Since listing the byte stream would require multiple clock cycles, there will be a READY output and a START input.



Problem Definition

In this encoding scheme, each output STREAM is 8 bits. Left most bit of the STREAM is continuation bit(c). c=1 means that the current stream is the end of byte stream and c=0 means it is not the end of the stream.

STREAM							
c	d_6	d_5	d_4	d_3	d_2	d_1	d_0

Assume that the INTEGER value we want to convert is G. We begin with one byte STREAM to store G and dedicate left most bit of the STREAM as a continuation bit (c).

- If binary representation of *G* fits within 7 bits, then binary-encode it in the 7 available bits and set *c* = 1.
- If binary representation of *G* does not fit within 7 bits then set c=0 for each higher order 7-bits using the same algorithm.
- At the end set the continuation bit of the last byte STREAM to 1 (*c* = 1) and set the continuation bit of the other STREAM bytes to 0 (*c* = 0).

As an example, assume that we have given the following 4-byte unsigned integer value (125) as input (the punctuation is only for readability):

G=00000000, 00000000, 00000000, 01111101

Since G can be represented in 7 bits, we produce the output STREAM as **1**1111101 where left most bit is the continuation bit c.

As another example, assume that we have given the following 4-byte unsigned integer value (824) as input:

G= 00000000, 00000000, 00000011, 00111000

By using this compression technique G now can be represented in 14 bits and the compressed version will be as the following, **0**0000110 **1**0111000 two bytes stream. Please note that the output STREAM should be ordered from left to right. Also note that 824 is bigger than 127, so it can **not** fit in one byte. We need 2 bytes to encode it.

Part -1: Decoder (Written)

In this part of the assignment, you are expected to draw an ASM chart showing the behavior of the VBEncoder module.
You can read Part-2 for the details of the VBEncoder module.
You can use any input-output definitions or abbreviations of them (by stating the exact names of it in your assignment) in VBEncoder. Also you can use VARIABLE[a:b] in order to refer a variable while implementing VBEncoder.
Please make sure that you cover all the details of VBEncoder module and conform to the ASM notation.

Deliverables (Part 1)

- You will submit your ASM charts as hardcopy. You may use any computer program, or prefer to draw a hand-written chart. If you won't use a computer program to draw your chart, make sure your chart is easily readable.
- Submit your ASM chart to A-206 on Monday, until 17:30 (1 week).
- This part is individual; any kind of group work will be regarded as cheating.
- Use the newsgroup metu.ceng.course.232 for any questions.

Part -2: VBEncoder Implementation

You will implement the provided VBEncoder in FPGA boards.

Use the module definition below:

```
module VBEncoder(input CLK, input [7:0] INT4,input [7:0] INT3, input
[7:0] INT2, input [7:0] INT1, input START, output reg READY, output
reg [7:0] STREAM);
```

4 Byte input is divided into 4 inputs INT4 to INT1. INT4 is the most significant byte and the INT1 is the least significant byte of the given unsigned integer.

You will place the encoding output to the STREAM register. You start listing from the **most** significant byte and at each clock cycle you will output the next part of the compressed data.

Clarifications

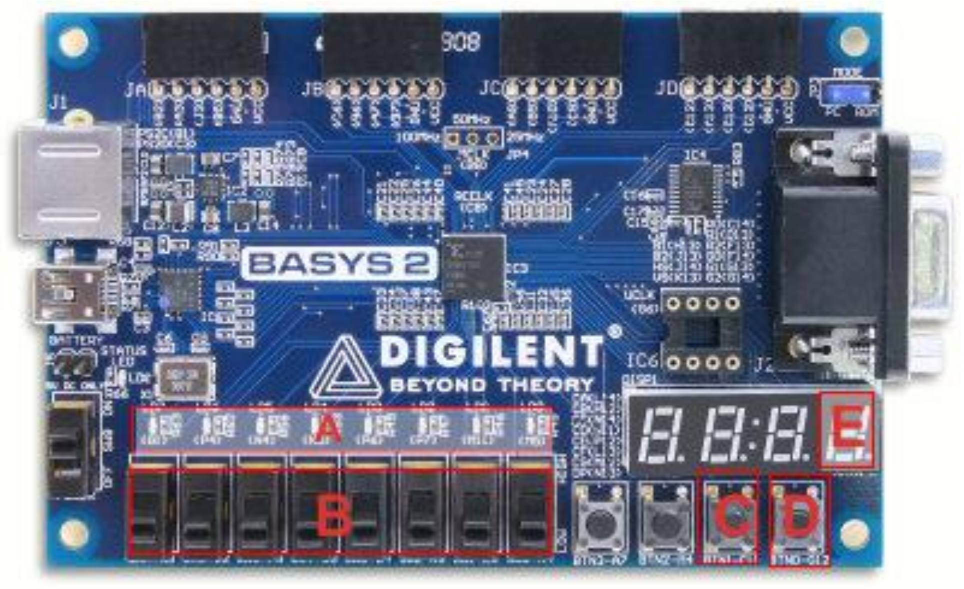
1. You should assume that the system is in IDLE state before calculation starts and after listing ends.
2. Calculation will begin when START is equal to 1, and calculation/listing won't stop even if START becomes 0. But after the listing ends, for a new calculation, you should wait until START becomes 1 again.
3. While the module is in the IDLE state at each posedge CLK, you will control START. If zero you will let the CLK go on without changing state. If start is 1 then you will change the state and in the next CLK you will start calculating. And afterwards you will list all bytes of the STREAM per one CLK. In the next CLK after listing is finished you will turn back to IDLE and change READY to 1.

4. START is only checked while in the IDLE state. When the circuit passes to another state, value of START becomes irrevelevant until arriving at IDLE again (i.e For the circuit that uses VBEncoder, there is no way to cancel a calculation/listing once it starts, calculation/listing has to finish on its own).
5. You must press&release CLK button while START is pressed.
6. READY becomes 0 only when "a posedge clock arrives while START=1".
7. You can examine pages 39-40-41 from Recitation-3(ceng232-20152-rec3.pdf) for how to handle edge-based clock and level-based inputs together properly. As in those examples, having a "next_xxxx" for your local registers (state, current calculation values, etc.) will make synchronizing things easier.
8. In demo, we will change the values of INT4-INT2 manually in Board232.v file and change INT1 by 8 switches.
9. Please note that, since 1 bit of each byte is reserved for the continuation bit, the total STREAM might occupy more than 4 bytes for large numbers.

FPGA Implementation

You will be provided with a Board232.v file (and a ready-to-use Xilinx project), which will bind inputs and outputs of the FPGA board with your Verilog module. You are required to test your Verilog module on the FPGA boards. After the submission date, you will make a demo to course assistants.

Name	**Type**	**Size**
INT4,INT3,INT2		Will be given as input in Board232.v
INT1	SW7 to SW0	All switches(B)
START	BTN1 + BTN0	Right-most two buttons (C +D)
Clock (CLK)	BTN0	Right-most button (D)
READY	7-segment display	Right most 7 segment display (E)
STREAM	LD7-LD0	All 8 LEDs(A)



Deliverables:

- Implement both modules in a single Verilog file: lab5_2.v. Do NOT submit your testbenches, or project file (.xise).
- You can share your testbenches on the newsgroup.
- Submit the file through the COW system before the given deadline.
- Use the newsgroup metu.ceng.course.232 for any questions regarding to the homework.
- This part is supposed to be done with your group partner. Make sure both of you take roles in implementation of the project. Any kind of inter-group cheating is not allowed.