The Huffman decoder Core

publish at www.opencores.org

written by René Doss

(Dossmatik GmbH/ Germany)

Version 0.1/ Date 20.12.2009

Disclaimer:

Dossmatik is not responsible for the end product or the success of the end product in any way. The final sign and acceptance of any products or designs is the sole responsibility of the user. This core is at the moment not full tested. The actual state is alpha phase.

Introduction:

In the follow document a decoder is described for baseline jpeg pictures. The code is written for real time video streaming. Some efforts are made in optimization for speed and dynamic huffman table and dynamic quantization tables load.

Language: VHDL

License:

Generally explanation:

This core analyses jpeg baseline makers is the incoming data stream. A state machine switch in the correct state. The incoming picture header information is analyzed and is applied in the decoding process. All tables are dynamic.

Actual are this:

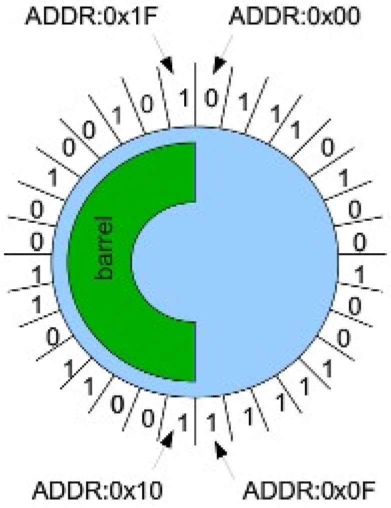
0xFFD8	Start of image
0xFFE0	App0 application segment
0xFFDB	define quantization table
0xFFC0	SOF0 Baseline DCT
0xFFC4	define huffman table
0xFFDA	start of scan
0xFFD9	End of image

Interface:

```
entity huffman_decoder is
port(
clk             :in std_logic;
--interface data input
wr              :in std_logic;                  --write
data_in         : in unsigned (7 downto 0);     --data jpeg stream input
wr_en           : out std_logic:='1';           --write enable
--interface  data out
output_valid    : buffer std_logic;             --use it as write signal in the follow IDCT
data_out        : out signed (15 downto 0);     --decoded and dequantized coefficient
next_eob        : buffer std_logic:='0';        --the next data is the last coefficient of block
                                                --all higher zigzag coeficients are zero
sop             : out std_logic:='0';           --start of picture
eop             : out std_logic:='0';           --end of picture
zrl             : out unsigned (3 downto 0);    -- number of consecutive zeros before the next
                                                        --coefficient
decoder_enable          : in std_logic);
end huffman_decoder;
```

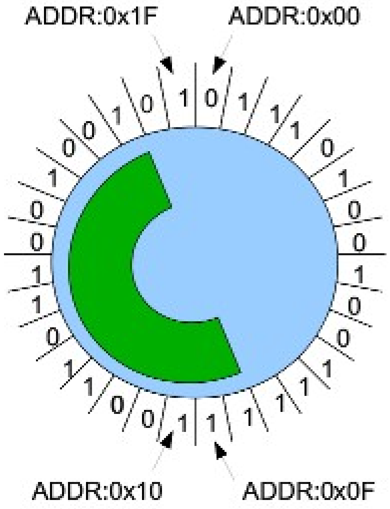
Implementation

The main part is a 32bit circular buffer (*signal rot_buffer: unsigned(31 downto 0);*) and inside this buffer is a shifted barrel. At the initialization the first value is stored at the highest address in the circular buffer. The next datas are stored in the lower addesses. The Barrel is at the highest address and decode the huffcode. The FSM sos_state control the process.



First sos_state decode:

The decoder decode the huffcode and shift the barrel as long the huffcode is. DC huffman table of the first component have to applicate. In this example it is 10 the huffcode and the corresponding code 6. The code is the length of the mantissa bits.



Second sos_state catch:

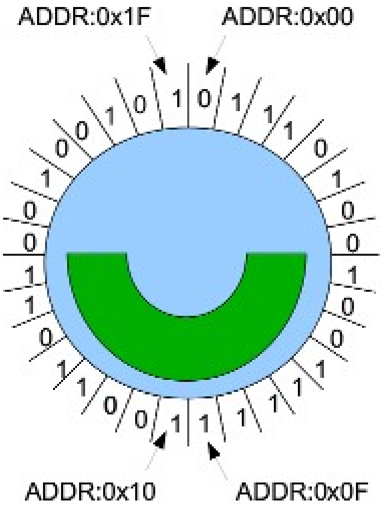
The amplitude value is token from barrel. 100100 is the value from data stream. Also the barrel is shifted to the next position.

Third sos_state post_catch:

Only a state that all value are valid.

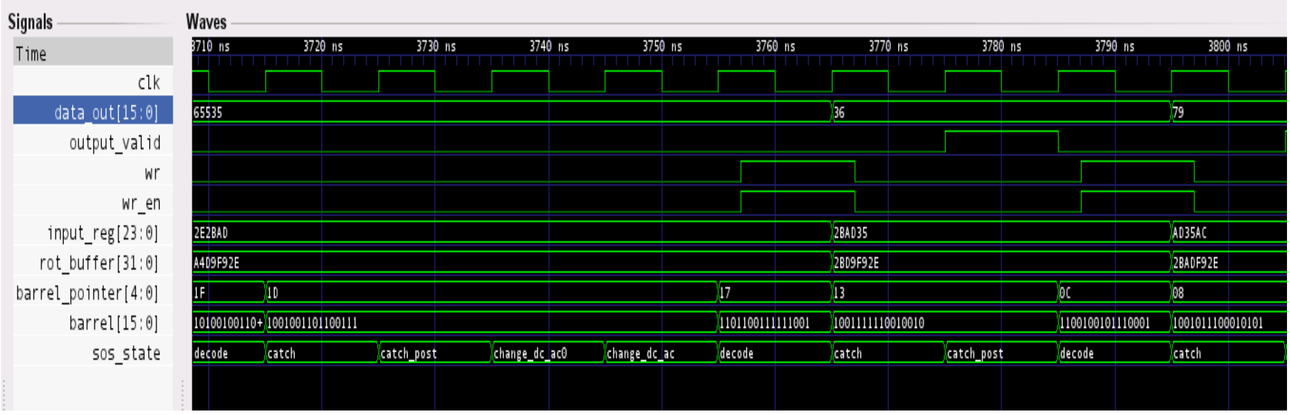
Fourth and fifth sos_state:

The huff table are two change two clocks are needed. For table change.



sixth sos_state:

The decoder decode the huffman code now with the AC huff table of the first component. The code is 1101 and has the length 4. Also in this step the values at the buffer addresss 0x1F..0x18 are invalid. Also in this step the next values from the input stream are loaded into the buffer.



In this timing diagram you can see only some clocks are needed in this implementation. This is important for high value streaming like video applications.

first picture:

header information

- quantization tables
- huffman tables

Pixel information in the stream:

sop (start of picture) goes high

switch to the valid table
decode and dequantizered
output the value in zigzag order

eop (end of picture) goes high

second picture:

header information

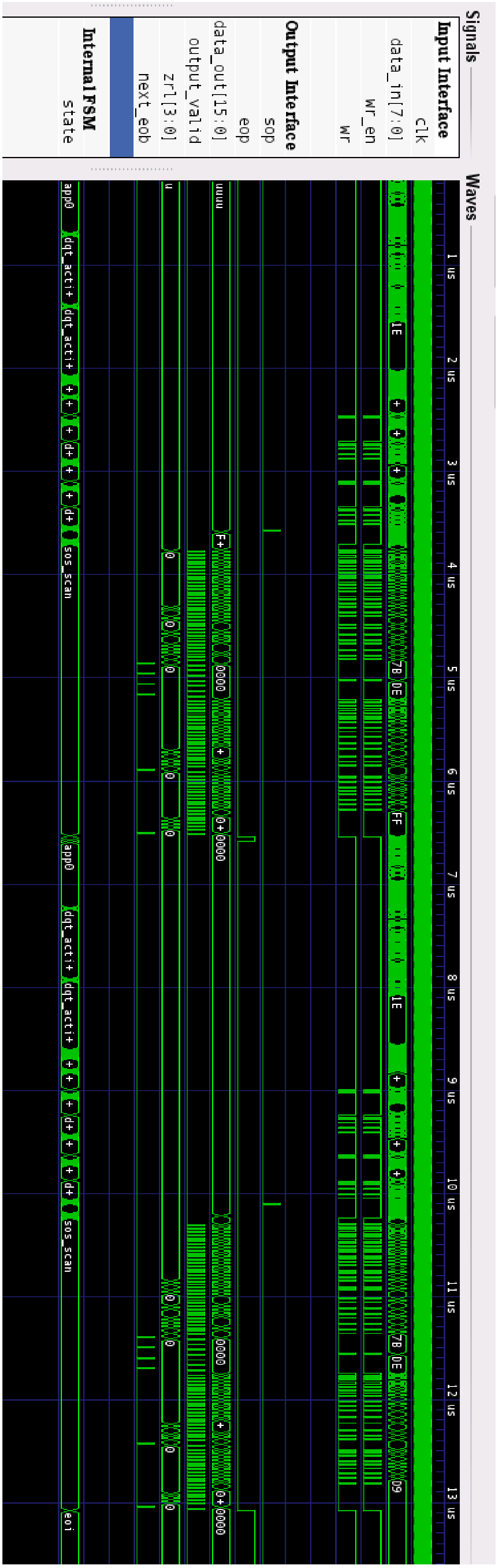
- quantization tables
- huffman tables

Pixel information in the stream:

sop (start of picture) goes high

switch to the valid table
decode and dequantizered
output the value in zigzag order

eop (end of picture) goes high


Signals
Waves
Input Interface
clk
data_in[7:0]
wr_en
wr
Output Interface
sop
eop
data_out[15:0]
output_valid
zrl[3:0]
next_eob
Internal FSM
state
1 us
2 us
3 us
4 us
5 us
6 us
7 us
8 us
9 us
10 us
11 us
12 us
13 us
1E
7B
DE
FF
D9
uuuu
F+
0000
0+
u
0
app0
dqt_acti+
sos_scan
eoi