jpegenc Documentation

Release 0.0.1

Merkourious and Vikram

October 22, 2016

Contents

CHAPTER 1

Backend Modules Documentation

Backend Modules:

1.1 Quantizer

1.1.1 divider module

This module contains the HDL for divider used for Quantiser

jpegenc.subblocks.quantizer.divider.**divider**
This module contains the HDL implementation

jpegenc.subblocks.quantizer.divider.**divider_ref**(*dividend*, *divisor*)
software implementation of divider

1.1.2 quant_rom module

MyHDL implementation of Quantiser ROM

jpegenc.subblocks.quantizer.quant_rom.**build_huffman_rom_tables**(*csvfile*)
build huffman tables

jpegenc.subblocks.quantizer.quant_rom.**quant_rom**
Build Chrominance ROM for Huffman Tables

1.1.3 quantizer module

The above module is the hardware implementation of quantizer top module

class jpegenc.subblocks.quantizer.quantizer.**QuantCtrl**
Bases: object

Control Signals used for quantizer top module

start : signal used to start the processing of block ready : asserts when block is ready to take next input color_components : select Y1 or Y2 or Cb or Cr component

class jpegenc.subblocks.quantizer.quantizer.**QuantIODataStream**(*width_data=12*, *width_addr=6*)
Bases: object

Input datastream into the Quantizer top module

data_in : send input data into the module read_addr : read the data from the input buffer

jpegenc.subblocks.quantizer.quantizer.**quantizer**
The Quantizer module divides the input data and data in the ROM

Arguments: quanti_datastream : Input datastream to the module quant_ctrl : control signals to the module

Returns: quanto_datastream : Output datastream from the module

1.1.4 quantizer_core module

The above module is the hardware implementation of quantizer core module

class jpegenc.subblocks.quantizer.quantizer_core.**QuantDataStream**(*width_data=12*)
Bases: object

Input interface for core module

data : input data to the quantizer core module valid : asserts when input data is valid

jpegenc.subblocks.quantizer.quantizer_core.**quantizer_core**
This Module is the core of the Quantizer

Arguments: quant_input_stream : Input stream to the core module color_component : used to select specific quantizer tables

Returns: quant_output_stream : Output data stream from the Quantizer

1.1.5 ramz module

This ram is used to store quantization values

jpegenc.subblocks.quantizer.ramz.**ramz**
default addr width 6, data width 12

1.1.6 romr module

This module generates reciprocals for numbers 0-255

jpegenc.subblocks.quantizer.romr.**romr**
Reciprocals of numbers are generated for quantizer core

1.2 RLE Module

1.2.1 doublebuffer module

The above module is a double buffer to store runlength encoded data

jpegenc.subblocks.rle.doublebuffer.**doublefifo**
I/O ports:

dfifo_bus : A FIFOBus connection interace buffer_sel : select a buffer

Constants :

depth : depth of the fifo used width_data : width of the data to be stored in FIFO

1.2.2 entropycoder module

This module takes a input and returns amplitude of the input and number of bits required to store the input.

jpegenc.subblocks.rle.entropycoder.**bit_length**(*num*, *maxlen=32*)
Determine the number of bits required to represent a value This functions provides the same functionality as the Python int.bit_length() function but is convertible.

This function generates the combinatorial logic to determine the maximum number of bits required to represent an unsigned value.

Currently the function computes a maximum of maxlen bits.

for values larger than 2**maxlen this function will fail. myhdl convertible

jpegenc.subblocks.rle.entropycoder.**entropy_encode**(*amplitude*)
Model of the entropy encoding

Arguments: amplitude (int): given an integer generate the encoding

Returns: amplitude_ref: size_ref:

jpegenc.subblocks.rle.entropycoder.**entropycoder**
This module return the amplitude and number of bits required to store input

io ports:

data_in : input data into the entropy coder

size : number of bits required to store amplitude amplitude : amplitude of the input

constants:

width_data : width of the input data

jpegenc.subblocks.rle.entropycoder.**two2bin**(*num*)
converts negative number to positive

1.2.3 rle module

This module is the MyHDL implementation of run length encoder top module

class jpegenc.subblocks.rle.rle.**BufferDataBus**(*width_data*, *width_size*, *width_runlength*)
Bases: *jpegenc.subblocks.rle.rlecore.RLESymbols*

Connections related to output data buffer

Amplitude : amplitude of the number size : size required to store amplitude runlength : number of zeros dovalid : asserts if ouput data is valid buffer_sel : select the buffer in double buffer read_enable : read data from the output fifo fifo_empty : asserts if any of the two fifos are empty

jpegenc.subblocks.rle.rle.**rlencoder**
The top module connects rle core and rle double buffer

I/O Ports:

datastream : input datastream bus buffer data bus : output data bus rleconfig : configuration bus

Constants:

width_data : input data width width_addr : address width width_size : width of register to store amplitude size max_addr_cnt : maximum address of the block being processed width_runlength : width of runlength value that can be stored limit : value of maximum runlength value width_depth : width of the FIFO Bus

1.2.4 rlecore module

This module if the core of the run length encoder module

class jpegenc.subblocks.rle.rlecore.**Component**
Bases: object

Select the color component

class jpegenc.subblocks.rle.rlecore.**DataStream**(*width_data=12*, *width_addr=6*)
Bases: object

Input data streams into Rle Core

data_in : input to the rle module read_addr : address of input data from the input ram

class jpegenc.subblocks.rle.rlecore.**RLEConfig**
Bases: object

RLE configuration Signals are the generic signals used in the block

color_component [select the color component] to be processed(Y1, Y2, Cb or Cr)

start : start signal triggers the module to start processing data sof : start of frame asserts when next frame is ready

class jpegenc.subblocks.rle.rlecore.**RLESymbols**(*width_data=12*, *width_size=6*, *width_runlength=4*)
Bases: object

Output symbols generatred by RLE Core

Amplitude : amplitude of the number size : size required to store amplitude runlength : number of zeros dovalid : asserts if ouput is valid

jpegenc.subblocks.rle.rlecore.**rle**
This is the RLE Core module

IO Ports:

datastream : input data and address to the input bus rlesymbols : output generated by core module rleconfig : configuration ports for rle core

constants:

width_data : input data width width_addr : address width width_size : width of register to store amplitude max_addr_cnt : maximum address of the block being processed width_runlength : width of runlength value that can be stored limit : value of maximum runlength value

jpegenc.subblocks.rle.rlecore.**sub**(*num1*, *num2*)
subtractor for Difference Encoder

1.3 Huffman Module

1.3.1 Submodules

1.3.2 ac_cr_rom module

MyHDL implementation of AC Chrominance ROM

jpegenc.subblocks.huffman.ac_cr_rom.**ac_cr_rom**
build ac ROM for chrominance

1.3.3 ac_rom module

MyHDL implementaton of Luminance AC ROM

jpegenc.subblocks.huffman.ac_rom.**ac_rom**
Build AC ROM here

1.3.4 dc_cr_rom module

MyHDL implementation of Chrominance DC ROM

jpegenc.subblocks.huffman.dc_cr_rom.**dc_cr_rom**
Build Chrominance ROM for Huffman Tables

1.3.5 dc_rom module

MyHDL implementation of DC ROM used for Huffman Encoder

jpegenc.subblocks.huffman.dc_rom.**dc_rom**
build dc rom here

1.3.6 doublebuffer module

The above module is a double buffer to store huffman encoded data

jpegenc.subblocks.huffman.doublebuffer.**doublefifo**
I/O ports:

dfifo_bus : A FIFOBus connection interace buffer_sel : select a buffer

Constants :

depth : depth of the fifo used width_data : width of the data to be stored in FIFO

1.3.7 huffman module

MyHDL implementation of Huffman Encoder Module

class jpegenc.subblocks.huffman.huffman.**HuffBufferDataBus**(*width_packed_byte*)
Bases: object

Output Interface of the Huffman module read_req : access to read the output data stored in FIFO fifo_empty : output fifo is empty buffer_sel : select a buffer from Double Fifo huf_packed_byte : Huffman Encoded Output

class jpegenc.subblocks.huffman.huffman.**HuffmanCntrl**
Bases: object

These are the control signals for Huffman block start : start sending block ready : request for next block color_component : select the component to be processed sof : start of frame

class jpegenc.subblocks.huffman.huffman.**HuffmanDataStream**(*width_runlength*, *width_size*, *width_amplitude*, *width_addr*)

Bases: object

Input interface bus to the Huffman module runlength : runlength of the data byte vli_size : number of bits required to store vli vli : amplitude of the data data_valid : input data is valid

class jpegenc.subblocks.huffman.huffman.**ImgSize**(*width=8*, *height=8*)

Bases: object

Indicates dimensions of the Image width : width of the image height : height of the image

class jpegenc.subblocks.huffman.huffman.**VLControl**

Bases: object

Contains the four states in which the FSM Operates

jpegenc.subblocks.huffman.huffman.**huffman**

HDL Implementation of Huffman Module. This module takes Variable Length Encoded Inputs and serialise them to VLC using Huffman Rom Tables

Args: huffmancntrl : control signals interface huffmandatastream : Input Interface img_size : Image data class rle_fifo_empty : asserts when Input buffer is empty

Returns: bufferdatabus : Output FIFO Interface

Constants: block_size : size of each block vlcontrol : contains the states used to run huff_fsm image_size.width : width of image image_size.heigth : height of image bits_block_count : width to store number of blocks in image width_word : maximum width of the word register

1.3.8 tablebuilder module

Used to build Huffman Tables

jpegenc.subblocks.huffman.tablebuilder.**build_huffman_rom_tables**(*csvfile*)

build huffman tables

1.4 ByteStuffer Module

1.4.1 bytestuffer module

This module is MyHDL implementation of Byte Stuffer used for JPEG Encoder

class jpegenc.subblocks.bytestuffer.bytestuffer.**BSInputDataStream**(*width_data*)

Bases: object

Input interface for the Byte Stuffer

data_in : Input data to Byte Stuffer read : read signal sent to input FIFO fifo_empty : asserts if input FIFO is empty

class jpegenc.subblocks.bytestuffer.bytestuffer.**BSOutputDataStream**(*width_data*, *width_addr_out*)

Bases: object

Output Interface for the Byte Stuffer

byte : output byte from the Byte Stuffer addr : output address to the RAM data_valid : asserts when output data is valid

class jpegenc.subblocks.bytestuffer.bytestuffer.**BScntrl**
Bases: object

Control Interface for Byte Stuffer

sof : start of frame start : send input frame when start asserts ready : ready to access next frame

jpegenc.subblocks.bytestuffer.bytestuffer.**bytestuffer**
Byte stuffer checks for 0xFF byte and adds a 0xFF00 Byte

Constants:

width_addr_out : maximum adress width of the output RAM width_out : width of the data in the ouput RAM

I/O Ports :

bs_in_stream : input interface to the byte stuffer bs_cntrl : control interface to the byte stuffer bs_out_stream : output interface to the byte stuffer num_enc_byte : number of bytes encoded to output RAM

1.5 Backend Module

1.5.1 backend module

MyHDL implementation of Backend Module

jpegenc.subblocks.backend.backend.**backend**
Constants:

width_data : width of the input data width_addr : width of the address accessed by a module width_runlength : width of the runlength value width_size : width of the size value width_out_byte : width of output byte width_num_bytes : max encoded bytes width

1.5.2 backend_soft module

software prototype for backend module

jpegenc.subblocks.backend.backend_soft.**backend_ref**(*block*, *prev_dc_0*, *prev_dc_1*, *prev_dc_2*, *register*, *color_component*, *pointer*)
backend reference module

jpegenc.subblocks.backend.backend_soft.**build_huffman_rom_tables**(*csvfile*)
build huffman tables

jpegenc.subblocks.backend.backend_soft.**build_rom_tables**(*csvfile*)
build huffman tables

jpegenc.subblocks.backend.backend_soft.**bytestuffer**(*block*)
bytestuffer reference module

jpegenc.subblocks.backend.backend_soft.**divider**(*block*, *color_component*)
divider reference module

jpegenc.subblocks.backend.backend_soft.**divider_ref**(*dividend*, *divisor*)
software implementation of divider

jpegenc.subblocks.backend.backend_soft.**entropy_encode**(*amplitude*)
Model of the entropy encoding

Arguments: amplitude (int): given an integer generate the encoding

Returns: amplitude_ref: size_ref:

jpegenc.subblocks.backend.backend_soft.**huffman_final**(*register*, *pointer*)
divide huffman code into bytes

jpegenc.subblocks.backend.backend_soft.**huffman_ref**(*runlength_block*, *amplitude_block*, *size_block*, *color_component*, *register*, *pointer*)
reference model for huffman encoder

jpegenc.subblocks.backend.backend_soft.**runlength**(*block*, *color_component*, *prev_dc_0*, *prev_dc_1*, *prev_dc_2*)
reference for runlength encoder module

jpegenc.subblocks.backend.backend_soft.**table_huff_gen**(*filename*, *base*)
huffman table generator

1.5.3 dualram module

jpegenc.subblocks.backend.dualram.**dram**
default addr width 6, data width 12

Test-Bench:

1.6 Quantizer Test

1.6.1 Quantizer-Top Test

1.6.2 Quantizer_core Test

1.6.3 Divider Test

This module is the testbench for the divider used in Quantiser module

test.test_divider.**test_divider**()
The functionality of the divider is tested here

test.test_divider.**test_block_conversion**()
Test bench used for conversion purpose

1.7 RLE Test

1.7.1 RLE-Top Test

1.7.2 RLE_core Test

1.7.3 Entropy Coder Test

This module tests the functionality and conversion of Entropy Coder

test.test_entropycoder.**test_entropycoder**()
: We will test the functionality of entropy coder in this block

  constants:

  width_data : width of the input data size_data : size required to store the data

test.test_entropycoder.**test_block_conversion**()
: Test bench used for conversion purpose

1.7.4 RLE Double Buffer Test

Test file for doublebuffer to check its conversion and funtioning

test.test_rledoublebuffer.**test_doublebuffer**()
: The functionality of Double Buffer is tested here

test.test_rledoublebuffer.**test_doublebuffer_conversion**()
: This block checks the conversion of Rle Double Fifo

1.8 Huffman Test

1.8.1 Huffman Test

1.8.2 Huffman Double Buffer Test

Test file for doublebuffer to check its conversion and funtioning

test.test_huffdoublebuffer.**test_doublebuffer**()
: The functionality of Double Buffer is tested here

test.test_huffdoublebuffer.**test_doublebuffer_conversion**()
: This block checks the conversion of Rle Double Fifo

1.9 Bytestuffer Test

This module tests the functionality and conversion of ByteStuffer Module

test.test_bytestuffer.**test_bytestuffer**()
: We will test the functionality of bytestuffer in this block

  Constants:

width_addr_out : maximum adress width of the output RAM width_out : width of the data in the ouput RAM

test.test_bytestuffer.**test_block_conversion**()
Test bench used for conversion purpose

1.10 Backend Test

This module tests the functionality and conversion of Backend Module

test.test_backend.**backend_soft**()
backend reference model

test.test_backend.**test_backend**()
We will test the functionality of entropy coder in this block

constants:

width_data : width of the input data size_data : size required to store the data width_addr : width of the address

test.test_backend.**test_backend_conversion**()
We will test the functionality of entropy coder in this block

constants:

width_data : width of the input data size_data : size required to store the data width_addr : width of the address

Results:

1.11 Coverage Results for Backend Modules

Module	Coverage
Quantizer	100
Quanizer_Core	100
Divider	100
RLE	100
RLE_Core	100
EntropyCoder	100
RLE_Doublebuffer	100
Huffman	99
Huffman_Doublebuffer	100
ByteStuffer	100
Backend	99

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

j

t

A

B

C

D

E

H

I

J

Q

R

S

T

V