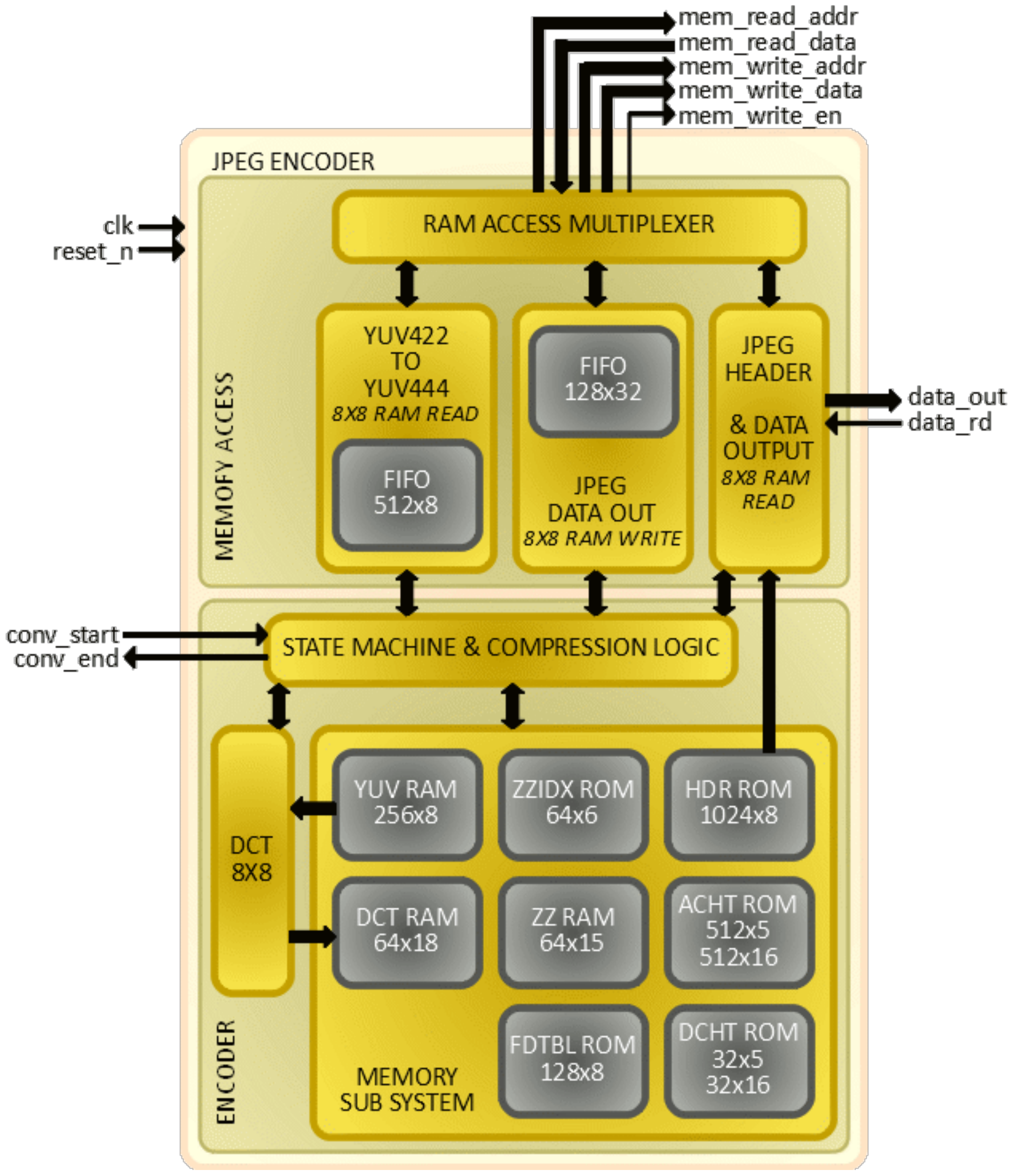
Gnarly Grey
A microelectronic architecture company

Gnarly Grey JPEG Encoder IP

The Gnarly Grey JPEG Encoder IP provides JPEG encoding of images in a very small number of FPGA gates. The JPEG Encoder features Baseline DCT, Huffman Encoding and a JFIF Header. It has configurable image height and width parameters and takes input in YUV422 [Y_0 U Y_1 V] format. Upon a trigger to start image compression, the IP reads image raw data (YUV422) from the memory interface and simultaneously writes back encoded data to memory interface. Once encoding operation is finished it reads back JPEG image header data from embedded block RAM and encoded data from the memory interface and is provided to the user with a simple FIFO-like interface.

IP Block Diagram



The JPEG Encoder IP design is primarily divided in two parts; MEMORY ACCESS and ENCODER.

MEMORY ACCESS

This portion of the IP is responsible for handling all data transactions to/from the memory interfaced with the IP. It comprises of three sub modules:

1. YUV422 TO YUV444 – Reads YUV422 data from the memory interface in 8x8 pixel blocks, converts data from YUV422 to YUV444 format and keeps converted data in a 512 bytes deep FIFO for further operation on it by the ENCODER.
2. JPEG DATA OUT – Temporarily stores encoded image data received from the ENCODER in a 128 unit deep FIFO. Once the YUV422 TO YUV444 module is no longer accessing memory, it writes data from the FIFO to memory interface in 8x8 addressing blocks.
3. JPEG HEADER & DATA OUTPUT – Once image compression is complete, this module provides interface to user logic to read the JPEG image size (without header), JPEG header data and JPEG compressed data in simple FIFO read fashion.

ENCODER

This portion of the IP is responsible to process data using JPEG compression algorithms. It comprises of three sub modules:

1. STATE MACHINE & COMPRESSION LOGIC – Regulates different states of the design and performs crucial operations like quantization, Huffman encoding and zigzag sequencing.
2. DCT – Performs 8x8 discrete cosine transfers on YUV data.
3. MEMORY SUB SYSTEM – Contains various embedded RAM and ROM blocks to help achieve image compression.

Port and Parameter Details

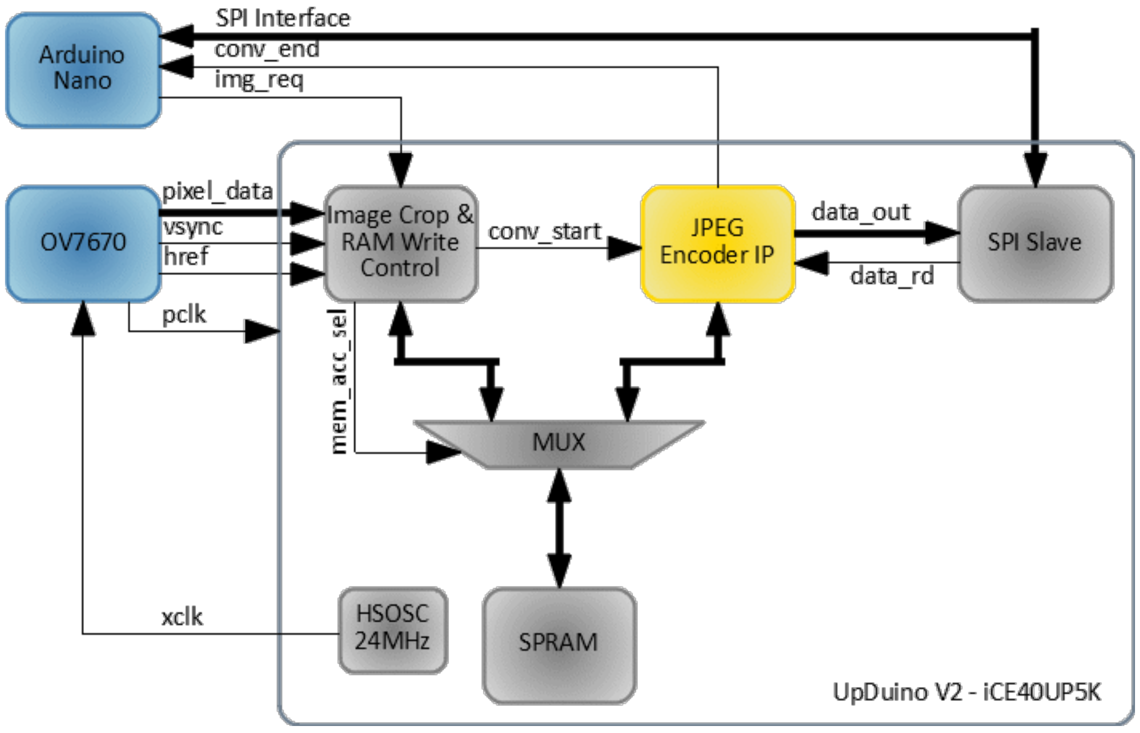
Name	Type	Size	Description
WIDTH	parameter	-	Image Width Parameter
HEIGHT	parameter	-	Image Height Parameter
clk	input	1	Clock
reset_n	input	1	Reset – Active High
conv_start	input	1	Trigger to Initiate Encoding – Active High
conv_end	output	1	Feedback on Encoding done – Active High
mem_read_addr	output	parameter dependent	Memory Read Address
mem_read_data	input	8	Memory Read Data
mem_write_addr	output	parameter dependent	Memory Write Address
mem_write_data	output	8	Memory Write Data
mem_write_en	output	1	Memory Write Enable – Active High
data_out	output	8	JPEG Data Output
data_rd	input	1	JPEG Data Read – Active High

Resource Utilization and FMAX

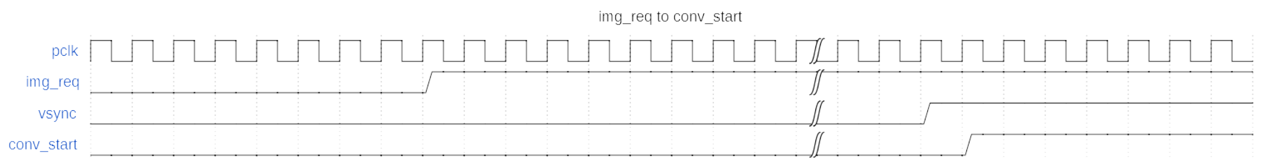
Device	LUT4	PFUREG	EBR	DSP	Fmax(MHz)
iCE40UP5K	3509	1264	13	3	9.713

Hardware Platform Example and Operation Sequence

This section describes a tested and provide hardware platform example the of JPEG Encoder IP and the control and data flow through the IP. The hardware platform uses OV7670 Camera Module, Updduino V2 (UltraPlus 5K), and Arduino to control JPEG Encoder and read JPEG encoded data over SPI interface.

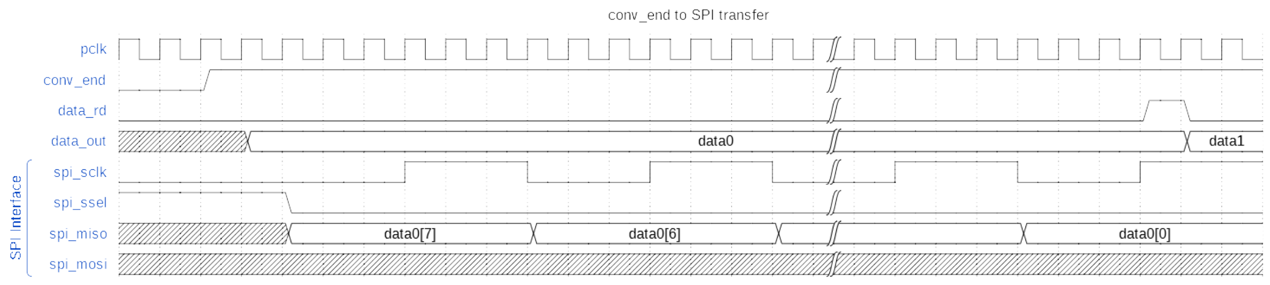


The block diagram above shows how overall system is setup. OV7670 is configured in YUYV mode with 320x240 resolution and 30 frames per second. Data received from OV7670 is stored in SPRAM blocks of the Lattice iCE40 Ultraplus FPGA (single buffer). Due to limitation of available SPRAM size on this device, image data is cropped to resolution 320x200. Unless image conversion request received from Arduino through GPO (img_req), new frame data received from OV7670 overwrites old frame data on SPRAM. Once image conversion request is received, user logic (Image Crop & RAM Write Control) makes sure current frame is written to SPRAM and then trigger is given to JPEG Encoder IP to start conversion/encoding.

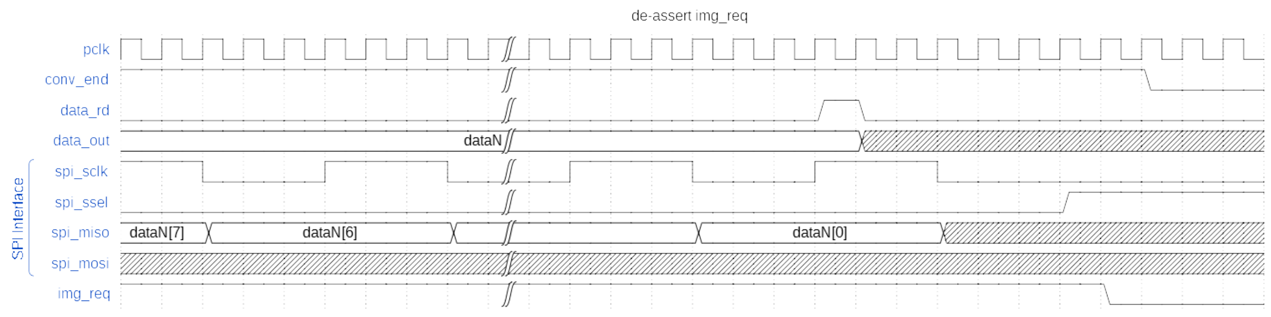


From this point of time to the time when JPEG encoded image is read by the Arduino microcontroller and acknowledged by de-asserting img_req signal, data from OV7670 is dropped without being written on SPRAM. Once JPEG Encoder complete encoding of data YUYV data present on SPRAM, it generates a signal (conv_end) to notify the Arduino microcontroller through

GPIO input pin.



The Arduino microcontroller can then read JPEG data through a SPI interface, a user logic- SPI Slave needs to be implemented to read data from the JPEG Encoder and pass it to Arduino microcontroller through SPI interface. The first four bytes send the size of the JPEG encoded data (it excludes the size of the JPEG header, which is of 607 bytes). After initial four bytes are transferred, a header data of 607 bytes is sent followed by the JPEG data. Once data transfer is complete, user code in the Arduino microcontroller can de-assert img_req signal in order to fill up SPRAM with new image content received from OV7670. Image conversion will start again when img_req signal is asserted.


de-assert img_req
pclk
conv_end
data_rd
data_out
dataN
SPI interface
spi_sclk
spi_ssel
spi_miso
dataN[7]
dataN[6]
dataN[0]
spi_mosi
img_req