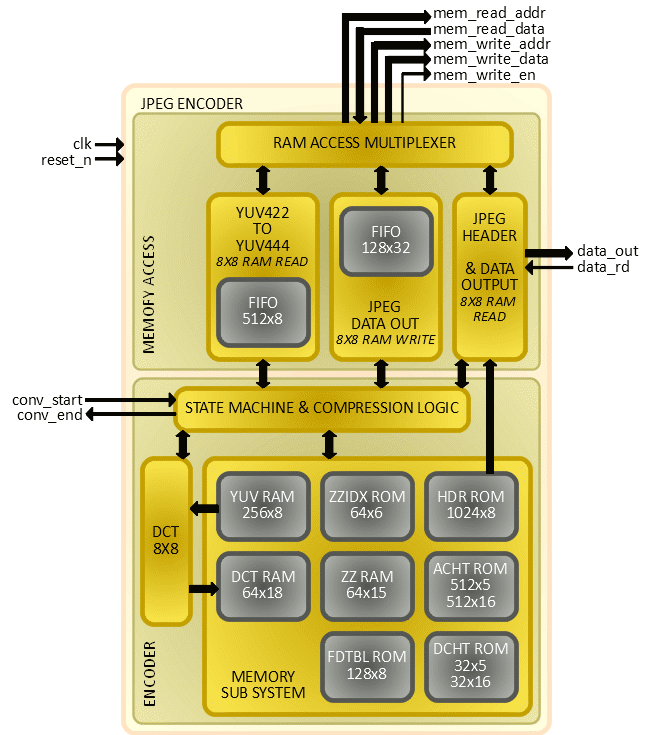
# JPEG Encoder IP

JPEG Encoder IP is a JPEG Encoder featuring Baseline DCT, Huffman Encoding and JFIF Header. It has configurable image height and width parameter and takes input in YUV422 [Y0 U Y1 V] format. Upon a trigger to start image compression, the IP reads image raw data (YUV422) from memory interface and simultaneously writes 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 memory interface and provides data output to user logic in simple FIFO read like interface.

## IP Block Diagram

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

### MEMORY ACCESS

It is responsible to handle all data transactions to/from image data memory interfaced with the IP. It comprises of three sub modules:

1. YUV422 TO YUV444 – It reads YUV422 data from memory interface in 8x8 pixel blocks, converts data from YUV422 to YUV444 format and keep converted data in 512 bytes deep FIFO for further operation on it by ENCODER part.
2. JPEG DATA OUT – It temporarily stores encoded image data received from ENCODER part in a FIFO with depth of 128 units, and once it finds YUV422 TO YUV444 module is not accessing memory, it writes data from FIFO to memory interface in 8x8 addressing fashion.
3. JPEG HEADER & DATA OUTPUT – Once image compression is done, this module provides interface to user logic to read JPEG image size (without header), JPEG header data and JPEG compressed data in simple FIFO read fashion.

### ENCODER

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

1. STATE MACHINE & COMPRESSION LOGIC – This is a kind of heart of the IP design, it regulates different states of the design and performs some crucial operations like quantization, Huffman encoding and zigzag sequencing.
2. DCT – As the name suggest, this module performs 8x8 DCT on YUV data.
3. MEMORY SUB SYSTEM – It contains various embedded RAM and ROM blocks to help achieve image compression.

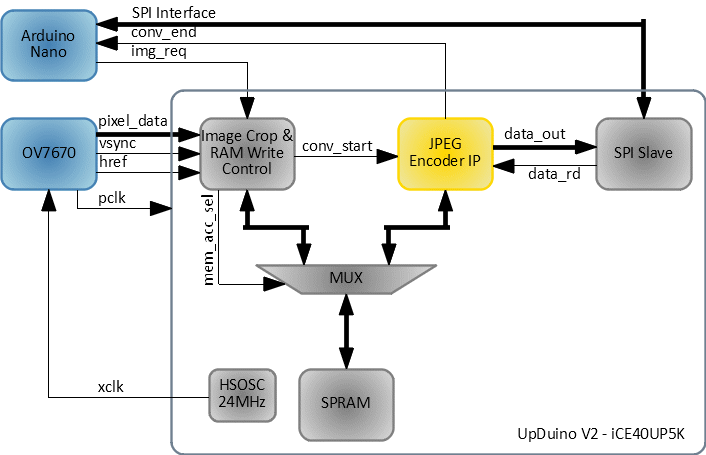
## Port and Parameter Details

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

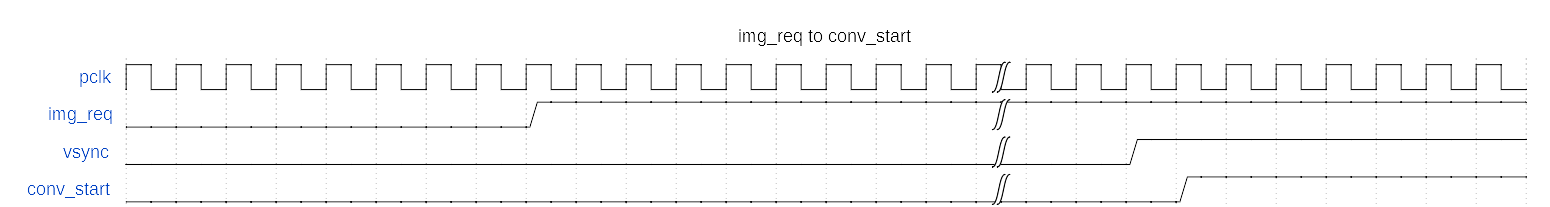
## Resource Utilization and Fmax

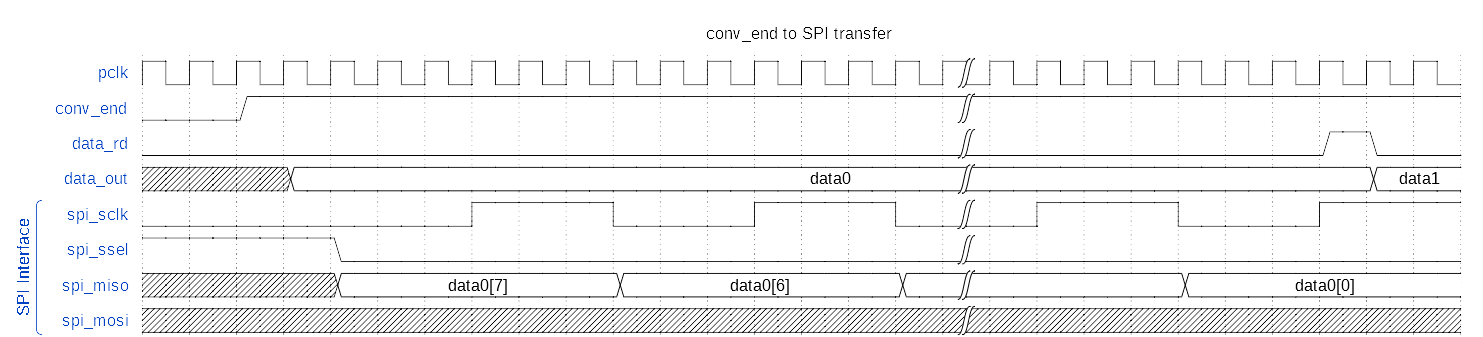
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Device** | **LUT4** | **PFUREG** | **EBR** | **DSP** | **Fmax(MHz)** |
| iCE40UP5K | 3509 | 1264 | 13 | 3 | 9.713 |

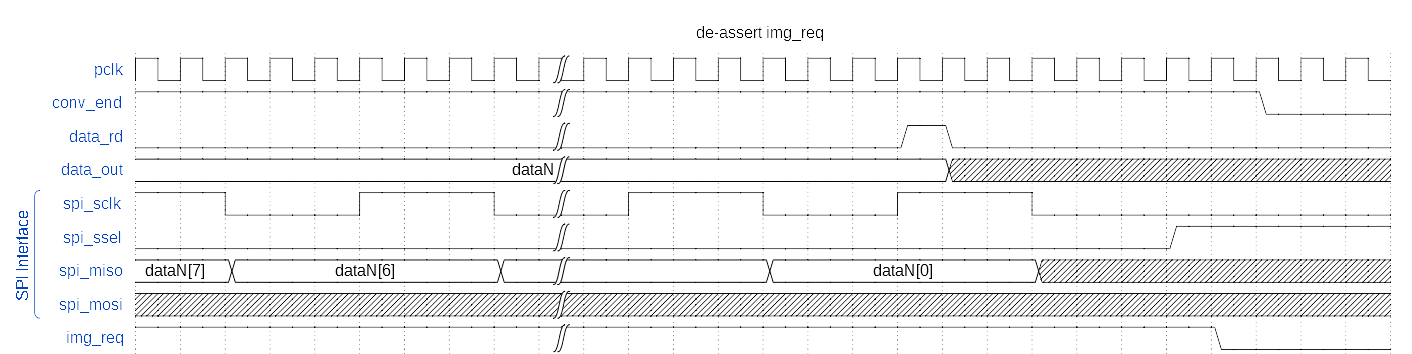
## Typical Example and Operation Sequence



This section describes a typical use case of JPEG Encoder IP- it uses OV7670 Camera Module, Upduino 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 iCE40UP5K (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 Arduino and acknowledged by de-asserting img\_req signal, data from OV7670 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 Arduino through GPI.

Arduino can then read JPEG data through SPI interface, a user logic- SPI Slave needs to be implemented to read data from JPEG Encoder and pass it to Arduino through SPI interface. First four bytes are sent as a size of JPEG encoded data (it excludes size of JPEG header, which is of 607 bytes, so user can find total JPEG file size by adding header size to it). After initial four bytes are transferred, header data of 607 bytes and then after jpeg encoded data of size received in first four bytes will be transferred over SPI interface. Once data transfer is complete, user code in Arduino can de-assert img\_req signal in order to fill up SPRAM with new image content received from OV7670.

Next image conversion will start again when img\_req signal is asserted again.