# OL_QOID
# QOI Imge Decompressor Core

**User's Manual Rev 1.0**
**Core version 1.0**

## Overview

This core implements the QOI lossless image compression algorithm decompressing QOI header-less files and producing RGB 24 bits pixels. Simple, fully synchronous design with low gate count.

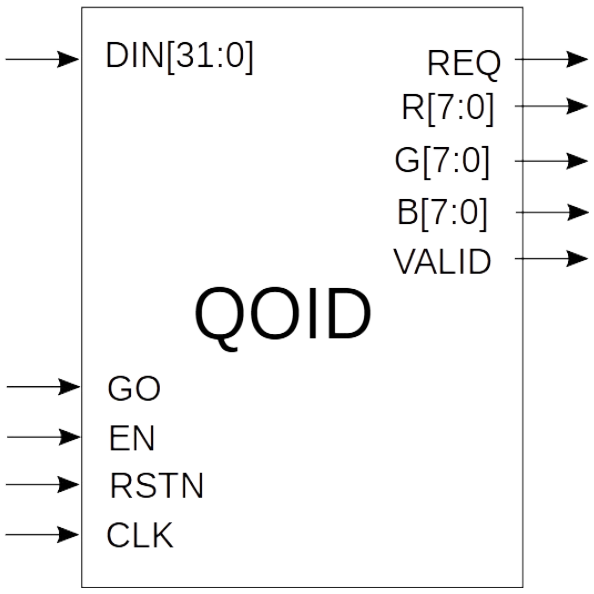
## Applications

- Bandwidth and storage reduction.
- Lossless compressed frame store.
- Space imaging applications.
- SoC to SoC image transmission.

## Features

- Implemented according to the QOI image format.
- Decompresses one 24 bits pixel per clock cycle.
- High throughput : up to 800 Mpixles/s in high end, 4K@30 in low end FPGAs.
- Optional header processing available.
- Fully synchronous design.
- Available as fully functional and synthesizable VHDL or Verilog soft-core.
- Test benches provided.

## Symbol



## The QOI Lossless Image Compression Algorithm

QOI stands for Quite OK Image Format and it is a fast, simple lossless image compression algorithm.

As it can be seen in these benchmarks, the performance of the algorithm is usually somewhere in between two well known image compression libraries (like libpng and stb) but at a much lower computational cost.

QOI uses a combination of some well known image compression techniques and innovative ideas.
Pixels are compressed as:

- runs of identical pixels
- an index to a 64 pixel cache of previously encountered pixels
- a difference to the previous pixel
- a full RGB pixel

The QOI algorithm is an excellent compromise between compression performance and low algorithmic complexity.

## QOI Lossless Image Compression Core

OL_QOID is a fast, low complexity implementation of the compression algorithm that decompresses one 24 bits RGB pixel per clock cycle.

# Pin Description

| Name | Type | Description |
|---|---|---|
| RSTN | Input | Core asynchronous reset, active low. |
| CLK | Input | Core clock signal. |
| EN | Input | Synchronous enable signal. When LOW operations stall and the core ignores all its inputs. |
| GO | Input | While HIGH, decompression continues. |
| DIN[31:0] | Intput | Input compressed data. |
| REQ | Output | When HIGH, a new compressed data word is requested. |
| R[7:0] | Output | Red component of the input pixel. |
| G[7:0] | Output | Green component of the input pixel. |
| B[7:0] | Output | Blue component of the input pixel. |
| VALID | Output | When HIGH, the RGB output is valid. |

## Functional description

The core is activated by rising the GO input and maintaining it HIGH as long as decompression is required.
The core will immediately rise the REQ signal to request at least two compressed data words at the DIN[31:0] input. More will be subsequently requested, as necessary.

As described in the QOI format, in the case of 24 bits pixels, up to 4 bytes can be required to represent a pixel. Therefore the input interface is 32 bit wide so that the core can decode one pixel per clock cycle in all circumstances.

After 4 cycles from raising GO, the core will start outputting pixel form the R[7:0], G[7:0] and B[7:0] output. This will be indicated by the VALID signal being HIGH.

If compressed data cannot be provided when the core requests it, then the EN signal must be lowered. This will stall the core as long as EN is LOW. Likewise for the output, if not ready to accept pixels being output.

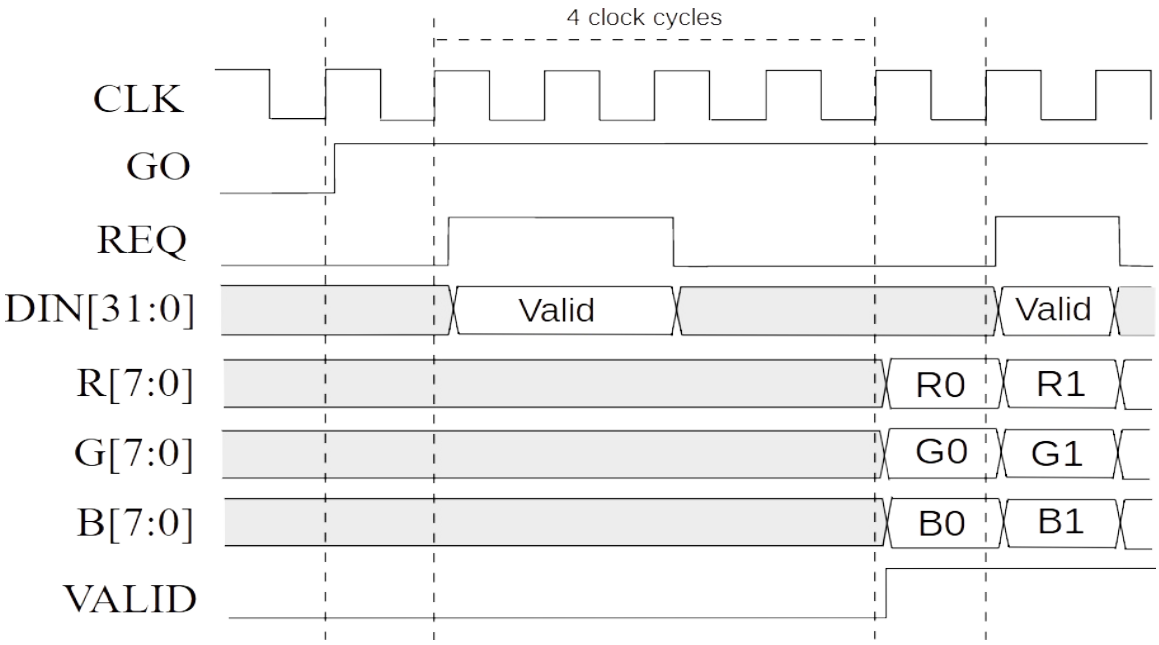



**Figure 1: Start core operations.**

Decoding of pixels will continue as long as GO is kept HIGH. Lowering GO will return the core to an idle state, resetting its internal status. GO should only be lowered to terminate decoding. If decompression needs to be suspended, the signal EN should be lowered instead.

Because of internal pipeline delays, the core might request data past the end of a QOI file before the last pixel is output. In this case, dummy data can be input into the core.

## C Model and test vectors generation

A C model of the core is provided for test vector generation. The C model provided is based on a the reference implementation, located on GitHub and included in the deliverable in the **qoi-master/** directory.

The test vector generators, **qoie.c** is also provided.
In order to compile this program, make sure that the **qoi.h** file from the **qoi-master/** directoryis visible by the c compiler.
The test vector generator **qoie.c** is compiled with :

**gcc –o qoie image-io.c qoie.c**

The resulting **qoie** executable usage is:

**./qoie** <input image in PPM format> <output image in QOI format>

Example :
**./qoie** image.ppm image.qoi

The resulting image.qoi file is a raw QOI (without header) binary file.
Two other utilities are also provided **bin2hex32.c** and **ppm2hex.c** .
They can be compiled with:
**gcc -o bin2hex32  bin2hex32.c**
**gcc -o ppm2hex  ppm2hex.c**

bin2hex32.c is used to convert the binary QOI file into a hex text file suitable for the testbench.
Example :
**./bin2hex** image.qoi qoi.dat

Likewise, **ppm2hex** is used to convert the PPM image file into a hex text file suitable for the testbench.
Example :
**./ppm2hex** image.ppm pixel.dat

The hex text files qoi.dat and pixel.dat can then be moved to the testbench directory to be used by the simulation.

It is possible to also use **qoie** to generate a QOI binary file that includes the QOI header, suitable for viewing by using the -h option.
Eample :
**./qoie** image.ppm image.qoi -h

In the Windows environment the free Irfanview can be used to view QOI files. The -h option should not be used to generate files for the testbench as this version of the core does not generate an header.

## Test bench

The directory **tb** contains the HDL file **tb.v(hd)**. This files represents the self-checking test benches provided with the core.
The figure below shows a block diagram of the test bench.

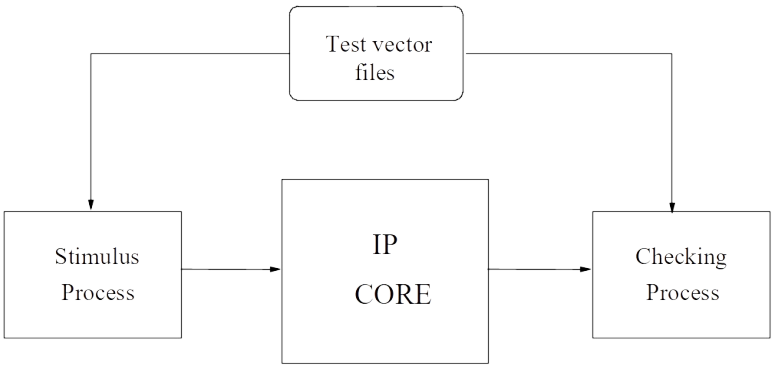



**Figure 2 Test bench block diagram.**

The Stimulus Process reads the input vectors and passes them to the core. The core results are verified by the Checking Process.

The testbench reads the compressed data from qoi.dat file and feeds them to the OL_QOID core. It also verifies that the output matches the pixels in the pixel.dat file.