LAB#C

介紹

Codec 為提供一個開源 C/C++的影像解編碼和相關演算法的 Library。提供以下的演算法：

JPEG decoding:有一個 L2 的 API 提供整個 JPEG 解碼的加速，基於序向離散餘弦轉換，可以在一個 cycle 處理一個霍夫曼 token 和產生八個 DCT coefficients。

pik encoding: 提供了三個 L2 api，可加速 90%的 Google's pik 的損失壓縮的 workload

WebP encoding:有兩個 L2 api，可加速 90%的 WebP 的損失壓縮的 workload

lepton encoding: "jpegDecLeptonEnc"這個 API 可以用來加速"Lepton"這個新的影像格式。是 Dropbox 所推出的。

JPEG-XL encoding: 有兩個 L2 api，可加速 90%的 JPEG-XL 的損失壓縮的 workload

bicubic resizing: 'resizeTop'這個 L2 API 是加速 bicubic 演算法。

這裡介紹 JPEG decoding 的演算法

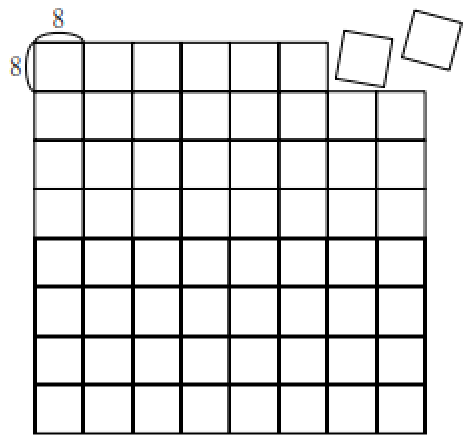
JPEG 是一種針對影像的失真壓縮方式，並且被廣泛使用。

首先，我們在做影像處理時使用的色彩模型為 YC_bC_r模型，我們圖片常見的色彩模型為 RGB，因此我們要先做色彩座標的轉換

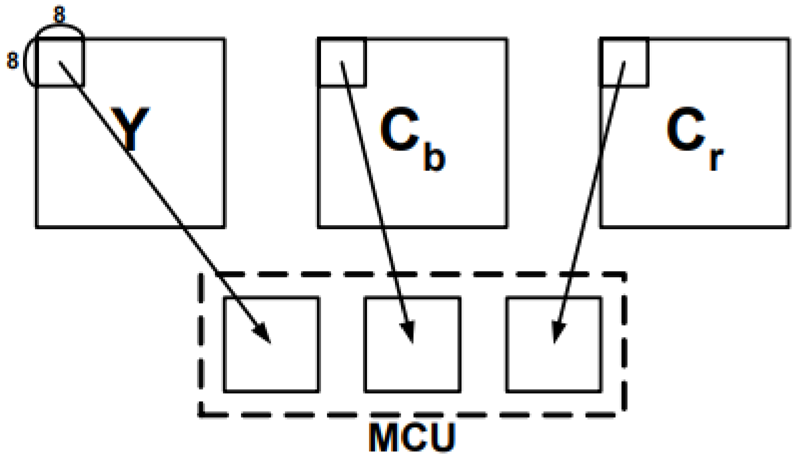
$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ C_b = -0.168R - 0.331G + 0.499B \\ C_r = 0.5R - 0.419G - 0.081B \end{cases}$$

做轉換的目的在於人類對於亮度較為敏感，也就是 Y 分量，而彩度則較為不敏感，也就是另外兩個分量。

做完轉換後，便可以對象素做取樣，再將影像切割為 8*8 大小的區塊



由於模型有三個分量，我們可以切出三個分量的 8*8 區塊，我們將之稱做一組 MCU



接著每一組都會經過 DCT 轉換，也就是離散餘弦轉換(Discrete Cosine Transform)，可將影像的資料轉到頻域

$$X_{k_1,k_2} = \frac{2}{N} c(k_1) c(k_2) \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x_{n_1,n_2} \cdot \cos\frac{(2n_1+1)k_1\pi}{2N} \cos\frac{(2n_2+1)k_2\pi}{2N}$$

$$n_1, n_2, k_1, k_2 = 0, 1, \cdots, N-1$$

$$where \quad c(0) = \frac{1}{\sqrt{2}} \quad and \quad c(n) = 1 \quad for\ n \neq 0$$

再透過量化表進行量化，將高頻訊號大幅量化為 0，盡量保留低頻的值

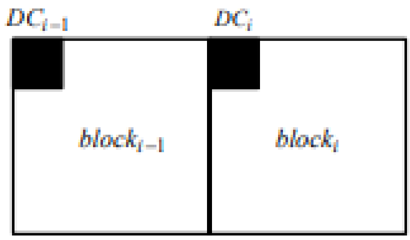
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	67	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

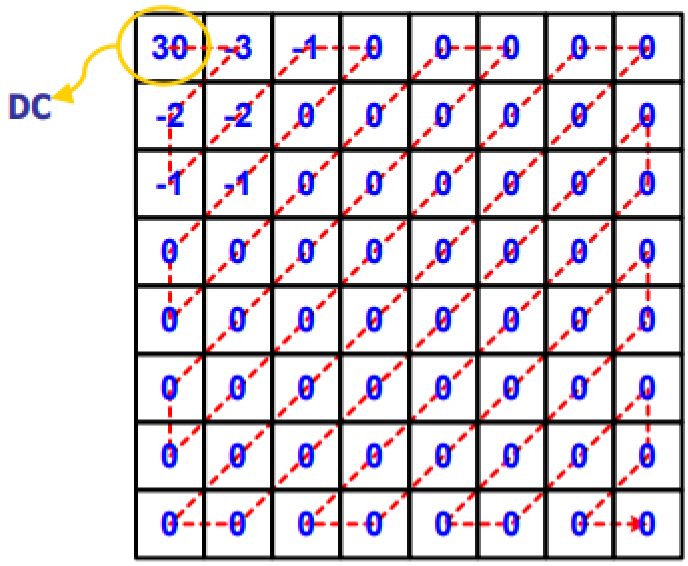
亮度量化表　　　彩度量化表

經過量化之後，可將信號分為 DC 值與 AC 值，所謂 DC 值即為每個 8×8 區塊中最左上角的值，而

剩下的 63 個皆為 AC 值。對於 DC 值我們使用差分編碼(Differential Pulse Code Modulation) ，由於每個 8×8 區塊的 DC 值基本上都很接近，因此與其儲存各 DC 值，不如儲存前後兩兩 DC 值間的差值

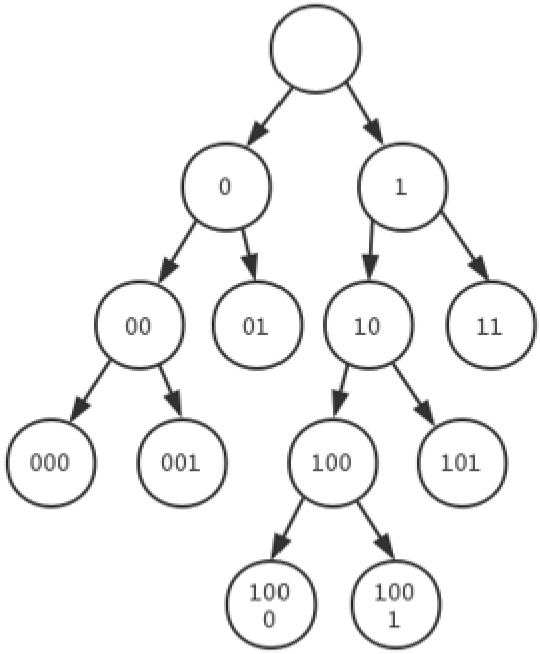


剩下的 AC 值會經過 Zig-Zag 掃描，可以讓零值的訊號連續出現，接下來使用變動長度編碼(Run-Length Coding)時，可以有效的提高壓縮率。

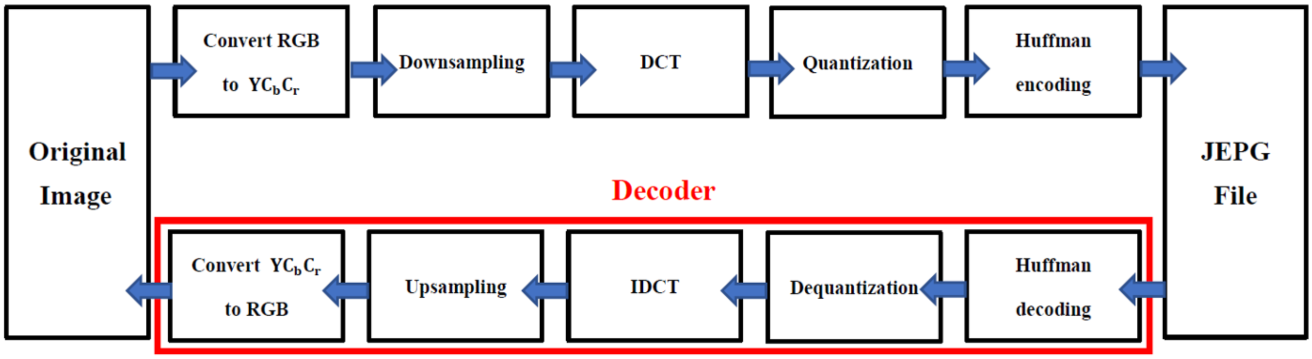


(R,L) => (0,-3)(0,-2)(0,-1)(0,-2)(0,-1)(2,-1)(EOB)

最後，將處理完的 DC 與 AC 值做霍夫曼編碼(Huffman Coding)，霍夫曼編碼的作法是將越常出現的值使用越短的編碼，其中用到了二元樹的概念，以達到無前綴碼的格式。



這樣便完成了 JPEG 的 encoder，若要 decode 回原圖，只需要按照 enoder 的順序反過來執行便可。



JPEG 本身只有描述如何將一個圖像轉換為位元組的數據，但並沒有說明這些位元組如何在任何特定的儲存媒體上被封存起來。JFIF（JPEG File Interchange Format，JPEG 檔案交換格式）定義了如何從一個 JPEG 產出一個適合於電腦儲存和傳輸的檔案，稱作 JFIF 檔；而 JPEG Header parser 則是 JPEG 裡的格式，像是如何判定為一個 JPEG 檔，或是開始字節與結束字節等等

TLA	Name	Hex	Size	Required	Special Notes
SOI	start of image	0xFF 0xD8	This tag does *not* have a size.	Yes	This tag *must* be the first one in the file.
APP0	application data	0xFF 0xE0	0x00 0x10 (16 bytes) for a standard image without a thumbnail.	Yes	This tag *must* come immediately after the SOI.
DQT	define quantization table	0xFF 0xDB	Variable size. Typically 0x00 0x43 (67 bytes) per table if this tag appears multiple times in the file. 0x00 0x84 (204 bytes) if two tables have been combined into a single tag. More if there are multiple tables, or if the tables are 16-bit instead of 8-bit.	Yes	The standard allows for multiple tables to be combined into a single DQT tag. I've seen both in use, JPEG files with multiple DQT segments, and JPEG files where the tables have been combined.
DHT	define Huffman table	0xFF 0xC4	Variable, depending on the number and the size of the tables.	Yes	The standard allows for multiple tables to be combined into a single DHT tag. I've seen both in use, JPEG files with multiple DHT segments, and JPEG files where the tables have been combined.
SOF0	start of frame (baseline DCT)	0xFF 0xC0	Variable size. Typically 0x00 0x11 (17 bytes) for images with 3 components (e.g., YCrCb).	Yes, but see "special notes".	SOF0 can be replaced with SOF1 (0xFFC1, extended sequential DCT), SOF2 (0xFFC2, progressive DCT), etc...
COM	comment	0xFF 0xFE	Variable size.	No	
SOS	start of scan	0xFF 0xDA	Complicated. See below for details.	Yes	The compressed image data comes *immediately* after the SOS tag.
EOI	end of image	0xFF 0xD9	This tag does *not* have a size.	Yes	This tag *must* be the last one in the image.

實作

1. 使用 L1 level 的 API 去實現 JPEG 解碼的加速。

事前設定：

- Source "/opt/Xilinx/Vitis_HLS/2022.1/settings64.sh"
- Source "/opt/Xilinx/Vitis/2022.1/settings64.sh"
- Source "/opt/xilinx/xrt/setup.sh"
- export PLATFORM_REPO_PATHS="/opt/xilinx/platforms/xilinx_u50_gen3x16_xdma_5_202210_1/xilinx_u50_gen3x16_xdma_5_202210_1.xpfm"
- export DEVICE= xilinx_u50_gen3x16_xdma_5_202210_1
- export TARGET=sw_emu

Csim

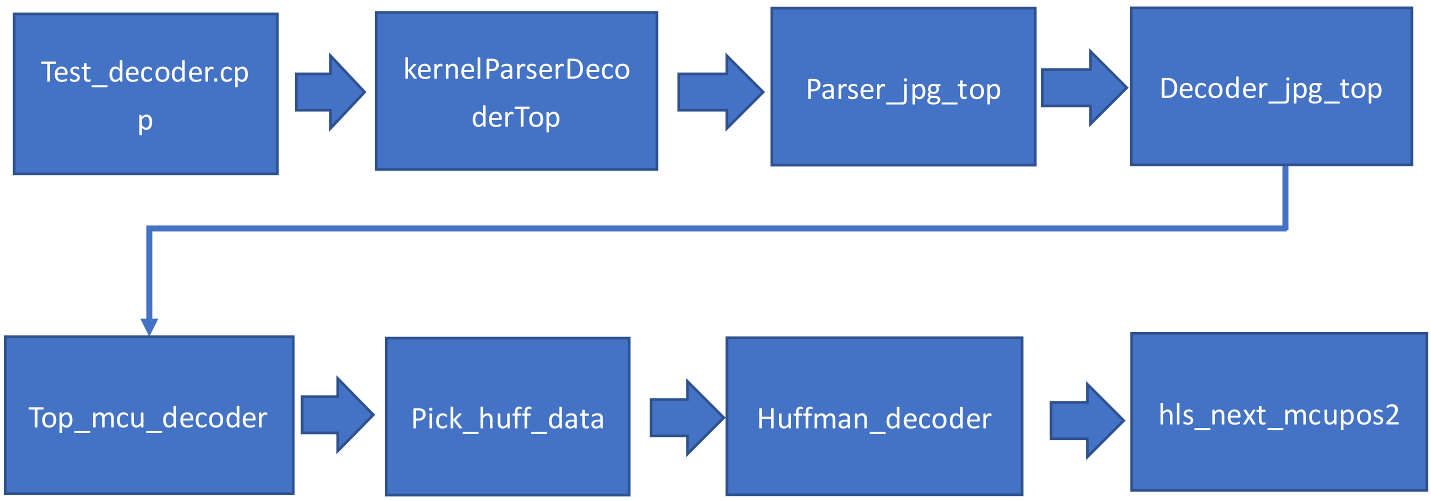
- make run DEVICE=xilinx_u50_gen3x16_xdma_5_202210_1 CSIM=1

Csyn

- make run DEVICE=xilinx_u50_gen3x16_xdma_5_202210_1 CSYNTH=1

Cosim

Code 流程圖：



Kernel_parser_decoder、KernelParserDecoderTop：定義整個流程的運作。
Parser_jpg_top：用來解析 JPEG 檔當中，JFIF 格式的 header mark(Huffman table、Data、DQT、SOF、COM、SOS 等等)。並用於後續的 decoder 中。
Decoder_jpg_top：整個 decoder 流程的運作。
Top_mcu_decoder、Pick_huff_data、Huffman_decoder、hls_next_mcupos2：處理後續的 huffman decode、Dequantization、IDCT 等等運算，最後產生一個 YUV 的圖片。

實際操作：
裡面有一個 run_hls.tcl 裡面有設定一個 targeted frequency 。

run_hls.tcl × Makefile

L1 > tests > jpegdec > run_hls.tcl

```
#
# Copyright 2019-2021 Xilinx, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

source settings.tcl

set PROJ "test.prj"
set SOLN "solution1"

if {![info exists CLKP]} {
  set CLKP 3.33
}

open_project -reset $PROJ

add_files "test_decoder.cpp ${XF_PROJ_ROOT}/L1/src/XAcc_jfifparser.cpp ${XF_PROJ_ROOT}/L1/sr
add_files -tb "test_decoder.cpp" -cflags "-I${XF_PROJ_ROOT}/L1/include -std=c++0x"
set_top kernel_parser_decoder
#set_top Huffman_decoder

open_solution -reset $SOLN




set_part $XPART
create_clock -period $CLKP

if {$CSIM == 1} {
  csim_design -argv "-JPEGFile ${XF_PROJ_ROOT}/L1/images/t0.jpg"
}

if {$CSYNTH == 1} {
  csynth_design
}

if {$COSIM == 1} {
  cosim_design -argv "-JPEGFile ${XF_PROJ_ROOT}/L1/images/t0.jpg"
```

C-simulation

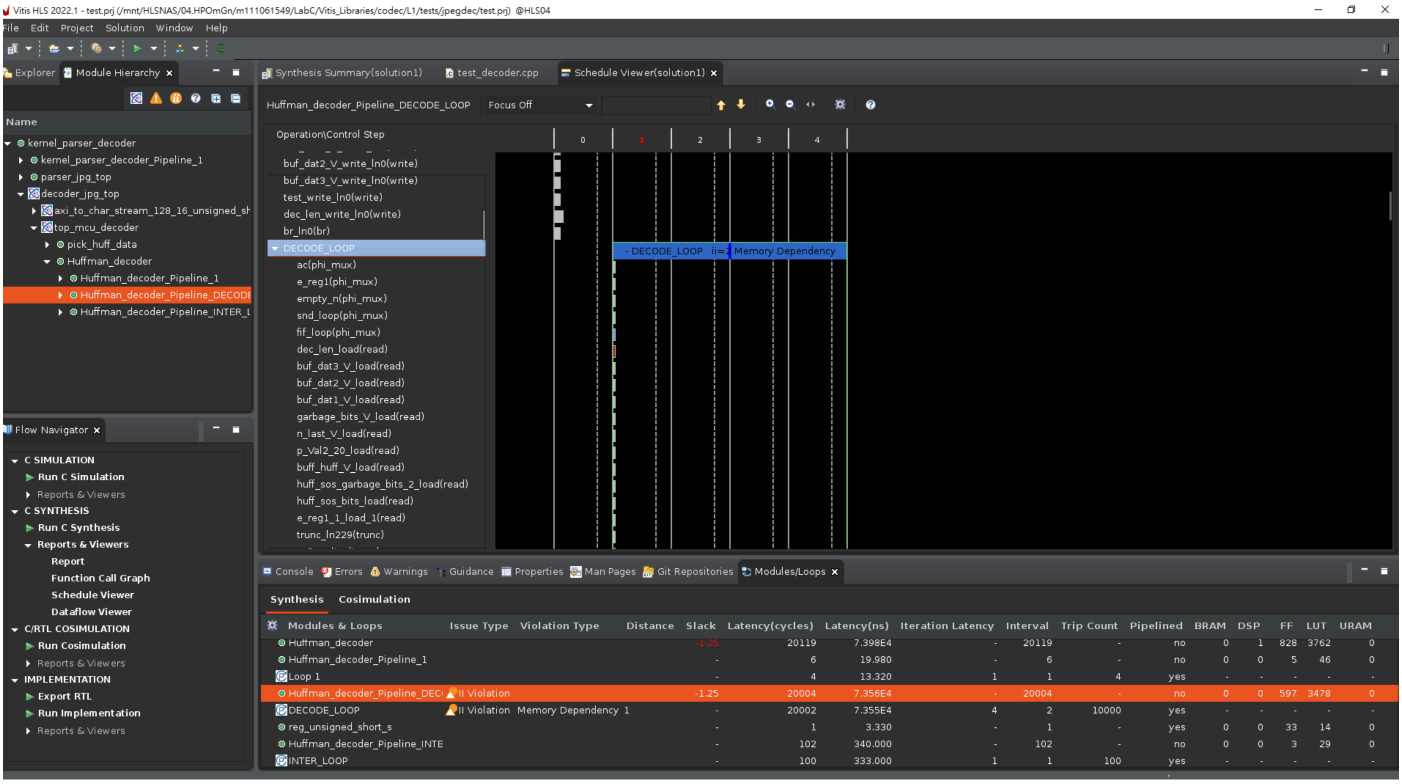
跑出結果的片段擷取：

```
huffman 1 bits codes is :0b0000000000000000
huffman 2 bits codes is :0b0000000000000000
huffman 3 bits codes is :0b0000000000000100
huffman 4 bits codes is :0b0000000000001010
huffman 5 bits codes is :0b0000000000011000
huffman 6 bits codes is :0b0000000000111000
huffman 7 bits codes is :0b0000000001111000
huffman 8 bits codes is :0b0000000011110110
huffman 9 bits codes is :0b0000000111110100
huffman 10 bits codes is :0b0000001111110110
huffman 11 bits codes is :0b0000011111110110
huffman 12 bits codes is :0b0000111111110100
huffman 13 bits codes is :0b0001111111110000
huffman 14 bits codes is :0b0011111111100000
huffman 15 bits codes is :0b0111111111000010
huffman 16 bits codes is :0b1111111110001000
huffman 1 bits start addr is :0
huffman 2 bits start addr is :0
huffman 3 bits start addr is :2
huffman 4 bits start addr is :7
huffman 5 bits start addr is :19
huffman 6 bits start addr is :47
huffman 7 bits start addr is :107
huffman 8 bits start addr is :230
huffman 9 bits start addr is :480
huffman 10 bits start addr is :987
huffman 11 bits start addr is :2006
huffman 12 bits start addr is :4048
huffman 13 bits start addr is :8136
huffman 14 bits start addr is :16312
huffman 15 bits start addr is :32665
huffman 16 bits start addr is :-163
```

```
****the end 3 blocks before zigzag are :
 ffffffb6,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  ffffffe6,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0015,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
 Ready for next image!
 INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 4.43 seconds. CPU system time: 0.31 seconds. Elapsed t
nds; current allocated memory: -1033.656 MB.
INFO: [HLS 200-112] Total CPU user time: 5.28 seconds. Total CPU system time: 0.54 seconds. Total elapsed time: 4.72 s
llocated memory: 209.762 MB.
INFO: [Common 17-206] Exiting vitis_hls at Thu Apr 20 21:51:01 2023...
```

Synthesis：

其實會發現，他的 II 並沒有=1，可以透過減少 bit width 來提升 timing，但那會導致 bandwidth 的降低。



Cosim:

```
//////////////////////////////////////////////////////////////////////////////////
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
//////////////////////////////////////////////////////////////////////////////////
// RTL Simulation : 0 / 1 [n/a] @ "109000"
// RTL Simulation : 1 / 1 [n/a] @ "725871000"
//////////////////////////////////////////////////////////////////////////////////
$finish called at time : 725890550 ps : File "/mnt/HLSNAS/04.HPOmGn/m111061549/LabC/Vitis_Libraries/codec
/L1/tests/jpegdec/test.prj/solution1/sim/verilog/kernel_parser_decoder.autotb.v" Line 1564
run: Time (s): cpu = 00:00:00.52 ; elapsed = 00:00:53 . Memory (MB): peak = 2707.812 ; gain = 0.000 ; fre
e physical = 38649 ; free virtual = 77631
## quit
INFO: xsimkernel Simulation Memory Usage: 297936 KB (Peak: 362472 KB), Simulation CPU Usage: 41390 ms
INFO: [Common 17-206] Exiting xsim at Sun Apr 23 18:49:21 2023...
INFO: [COSIM 212-316] Starting C post checking ...

------------ Test for decode image.jpg  -------------
WARNING: /mnt/HLSNAS/04.HPOmGn/m111061549/LabC/Vitis_Libraries/codec/L1/images/t0.jpg will be opened for
binary read.
51193 entries read from /mnt/HLSNAS/04.HPOmGn/m111061549/LabC/Vitis_Libraries/codec/L1/images/t0.jpg
****the end 3 blocks before zigzag are :
 ffffffb6,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  00
00,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  ffffffe6,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0
000,  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0015,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,  0000,
  0000,
 Ready for next image!
 INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation.
Otherwise, they will be marked as all NA. If user wants to calculate them, please make sure there are at
least 2 transactions in RTL simulation.
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 92.35 seconds. CPU system time: 2.72 sec
onds. Elapsed time: 86.33 seconds; current allocated memory: 9.750 MB.
INFO: [HLS 200-112] Total CPU user time: 126.98 seconds. Total CPU system time: 4.65 seconds. Total elaps
ed time: 159.26 seconds; peak allocated memory: 1.214 GB.
INFO: [Common 17-206] Exiting vitis_hls at Sun Apr 23 18:49:23 2023...
04.HPOmGn@HLS04:~/m111061549/LabC/Vitis_Libraries/codec/L1/tests/jpegdec$
```

Design with export：

- make run DEVICE=xilinx_u200_gen3x16_xdma_2_202110_1.xpfm VIVADO_IMPL=1

```
Report date:         Thu Apr 20 23:40:50 CST 2023

#== Post-Implementation Resource usage ==
SLICE:            0
LUT:           7718
FF:            7437
DSP:             12
BRAM:             5
URAM:             0
LATCH:            0
SRL:            656
CLB:           1643

#== Final timing ==
CP required:                    3.330
CP achieved post-synthesis:     3.639
CP achieved post-implementation: 3.153
Timing met

TIMESTAMP: HLS-REPORT: implementation end: 2023-04-20 23:40:50 CST
INFO: HLS-REPORT: impl run complete: worst setup slack (WNS)=0.176786, worst hold slack (WHS)=0.0
 number of unrouted nets=0
# hls_vivado_reports_finalize $report_options
TIMESTAMP: HLS-REPORT: all reports complete: 2023-04-20 23:40:50 CST
INFO: [Common 17-206] Exiting Vivado at Thu Apr 20 23:40:50 2023...
INFO: [HLS 200-802] Generated output file test.prj/solution1/impl/export.zip
INFO: [HLS 200-111] Finished Command export_design CPU user time: 209.07 seconds. CPU system time
current allocated memory: 11.773 MB.
INFO: [HLS 200-112] Total CPU user time: 243.6 seconds. Total CPU system time: 18.61 seconds. Tot
d memory: 1.214 GB.
INFO: [Common 17-206] Exiting vitis_hls at Thu Apr 20 23:40:55 2023...
04.HPOmGn@HLS04:~/m111061549/LabC/Vitis_Libraries/codec/L1/tests/jpegdec$
```

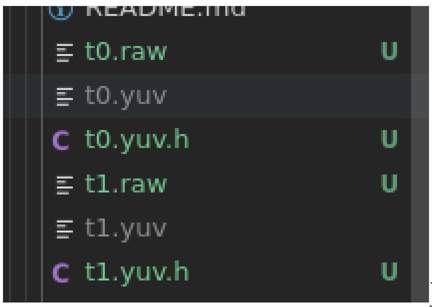
這篇報告顯示說 timing met，也就是有達到先前 run_hls.tcl 所設定的 target frequency

2. 使用 L2 API 調用加速演算法

L2 API 提供了一個 hostcode，可以將想要做 decoder 的 jpeg 圖片輸入，得到原圖：

實際跑 code：

```
INFO: writing the YUV file!
WARNING: t1.raw will be opened for binary write.
WARNING: t1.yuv will be opened for binary write.
INFO: fmt 1, bas_info->mcu_cmp = 6
INFO: bas_info->hls_mbs[cmp] 4, 1, 1
AE, AD, AD, AE, AF, B0, B0, B0,
B0, AF, AE, AB, A9, A8, AA, AC,
AF, B1, B2, B3, B2, B2, B3, B3,
B0, AF, AF, AD, AA, A7, A4, A3,
A1, A1, A0, A0, A0, A2, A4, A6,
A4, A4, A3, A2, A1, A1, A1, A1,
A2, A4, A5, A4, A2, A0, A0, A1,
A5, A3, A3, A3, A0, 9B, 99, 9A,
99, 99, 99, 98, 97, 96, 97, 97,
9C, 9E, A1, A3, A4, A4, A2, A1,
9F, 9E, 9D, 9D, 9E, 9F, A0, A0,
A0, A0, 9F, A0, A0, 9F, 9E, 9D,
9C, 9D, 9D, 9C, 9A, 98, 98, 99,
98, 98, 97, 98, 98, 98, 97, 96,
97, 96, 96, 96, 95, 94, 93, 91,
93, 8F, 8D, 8F, 90, 8D, 88, 84,
174, 173, 173, 174, 175, 176, 176, 176,
176, 175, 174, 171, 169, 168, 170, 172,
175, 177, 178, 179, 178, 178, 179, 179,
176, 175, 175, 173, 170, 167, 164, 163,
161, 161, 160, 160, 160, 162, 164, 166,
164, 164, 163, 162, 161, 161, 161, 161,
162, 164, 165, 164, 162, 160, 160, 161,
165, 163, 163, 163, 160, 155, 153, 154,
153, 153, 153, 152, 151, 150, 151, 151,
156, 158, 161, 163, 164, 164, 162, 161,
159, 158, 157, 157, 158, 159, 160, 160,
160, 160, 159, 160, 160, 159, 158, 157,
156, 157, 157, 156, 154, 152, 152, 153,
152, 152, 151, 152, 152, 152, 151, 150,
151, 150, 150, 150, 149, 148, 147, 145,
147, 143, 141, 143, 144, 141, 136, 132,
Please open the YUV file with fmt 1 and (width, height) = (416, 432)
WARNING: t1.yuv.h will be opened for binary write.
Ready for next image!
device process sw_emu_device done
 04.HPOmGn@HLS04:~/m111061549/LabC/Vitis_Libraries/codec/L2/demos/jpegDec$
```



並且可以生成.raw 檔圖片。’

Github 連結: https://github.com/s095339/LABC-Codec