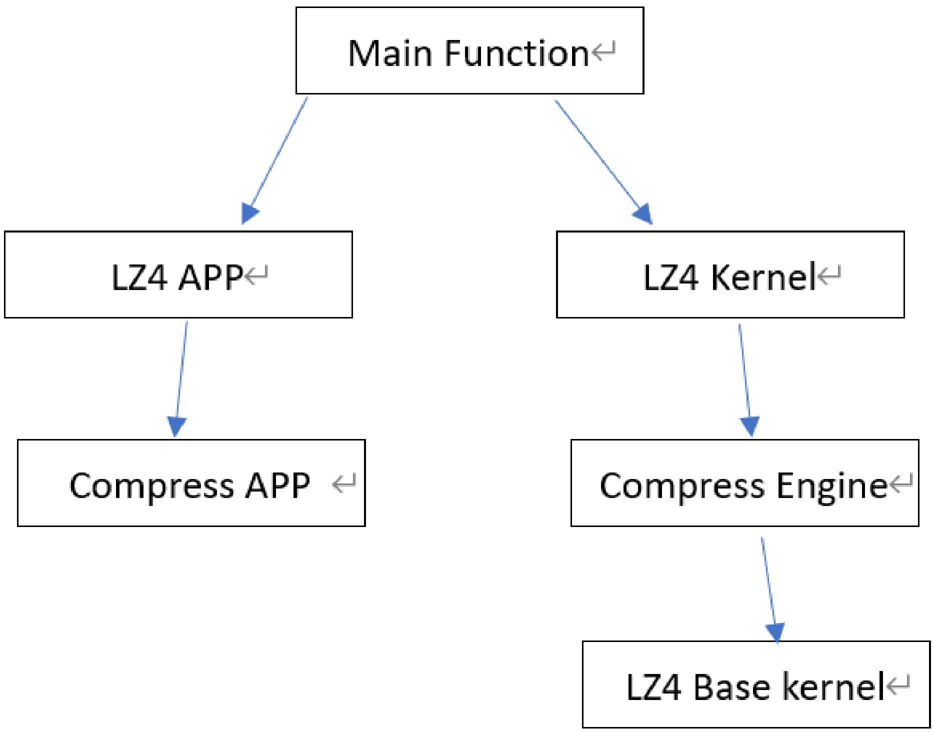
Lab C

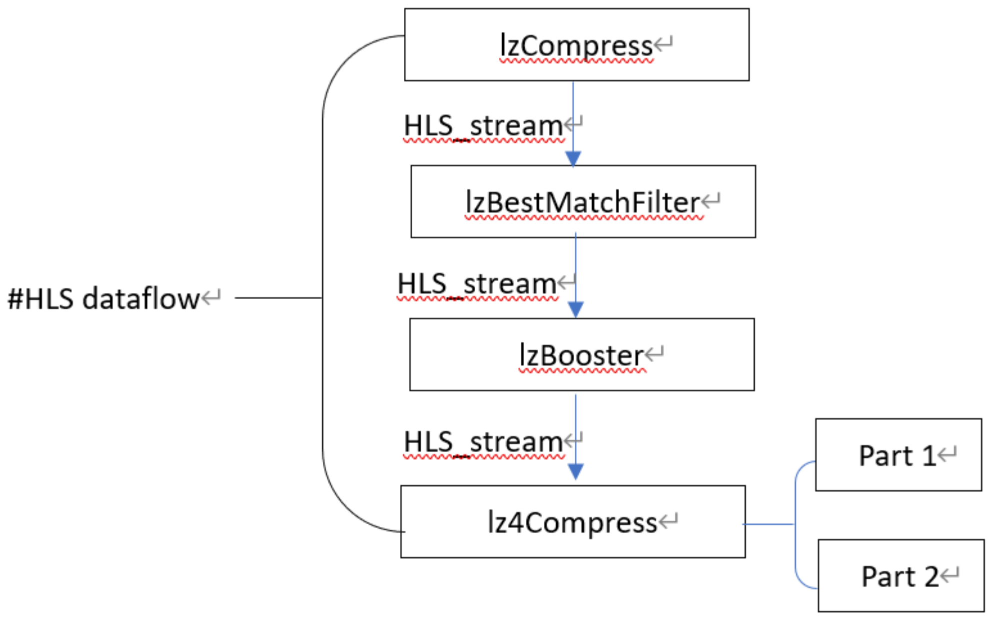
Background introduction

首先我做的 lab C 主題是 LZ4 data compression。那這個 library 可以通過以下的 host function hierarchy 直觀地描述：




Finding from lab work

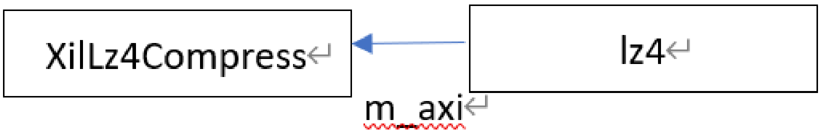
L1 架構(即 lz4Core 的 block)如下




Lzcompress 會把 input literal 做初步壓縮，並更新 match length 與 offset，得到相

關資訊後送入 lzBestMatchFilter，從 offset 的 window range 找到更高的 match length 來輸出 best match stream，再送入 lzBooster，在這裡會通過消耗 BRAM 來提高壓縮率在送入 lz4Compress，來做主要的 lz4 壓縮工作，最終輸出 literal stream 跟 offset stream。

L2 架構如下



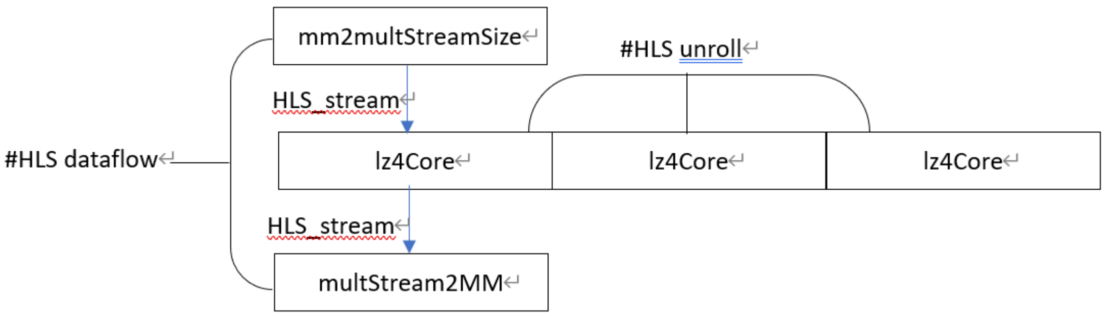

L2 Kernel global function(XilLz4Compress)如下：

```
void xilLz4Compress(const xf::compression::uintMemWidth_t* in,
                    xf::compression::uintMemWidth_t* out,
                    uint32_t* compressd_size,
                    uint32_t* in_block_size,
                    uint32_t block_size_in_kb,
                    uint32_t input_size);
}
```

參數列表

| | |
|---|---|
| in | Input raw data |
| out | Output compressed data |
| compressd_size | Compressed output size of each block |
| in_block_size | Input block size of each block |
| block_size_in_kb | Input block size in bytes |

lz4 架構如下

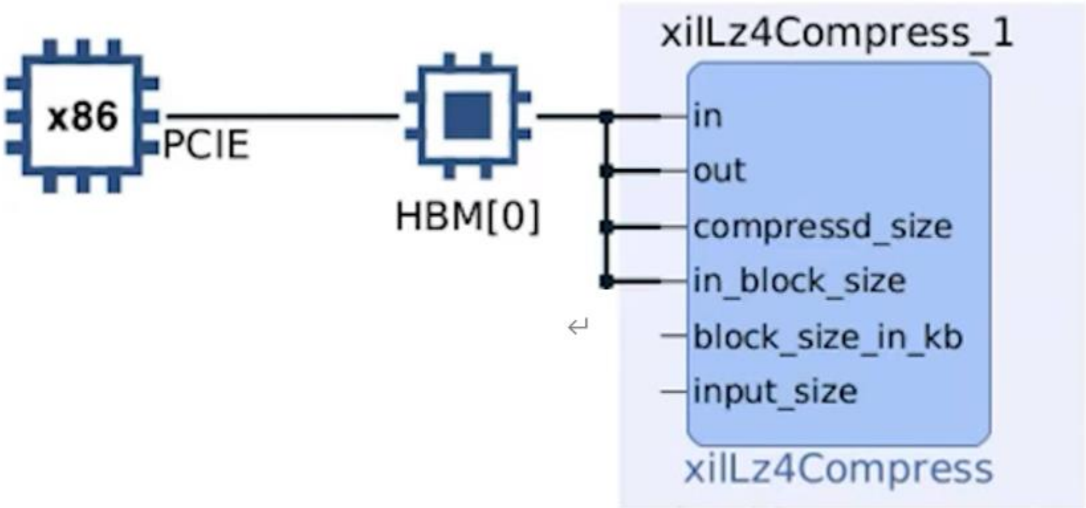



L2 會先得到所有 block 以及 block size，並將沒壓縮資訊送入，經過 lz4 輸出壓縮後的資訊，也會計算壓縮後的每個 block size。在 lz4 中，遵從 data flow 的概念，先 MM to Multiple Stream Size，然後走 dataflow 到 lz4core，這裡有做 unroll，數量取決與有多少平行的 block 來做平行處理，最後再換回去從 Multiple Stream Size to MM。

L3 host function 步驟如下

Step 1: read files (.txt or .lz4)
Step 2: compress/decompress by kernel
Step 3: Validate result with preprocessed golden file

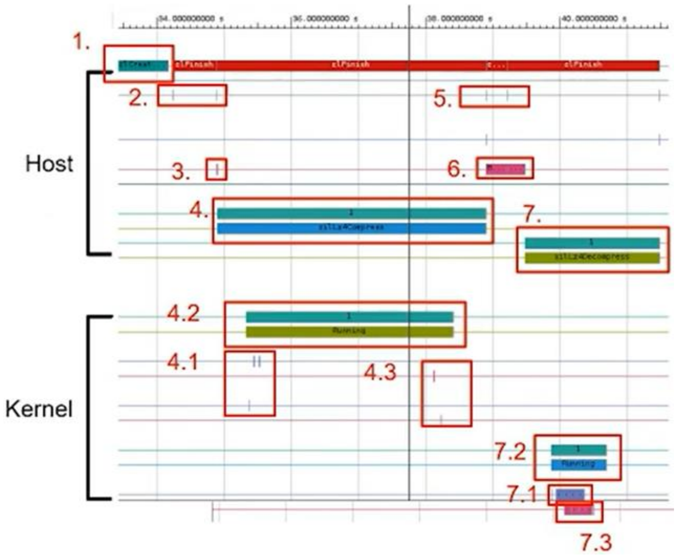
系統架構如下




Analysis

Compress time line analysis

1. CL object creation
2. Enqueue object & compress task
3. Write config & data to HBM
4. Waiting compress kernel execution
    1. Kernel read data from HBM
    2. Kernel execution
    3. Kernel write result to HBM



Report analysis

1. Low resource usage

| Name | Kernel | LUT (% Used) | Register (% Used) | BRAM (% Used) | URAM (% Used) | DSP (% Used) | Calls | CU Utilization (%) | Total Time (ms) | Avg Time (ms) |
|---|---|---|---|---|---|---|---|---|---|---|
| xilLz4Compress_1 | xilLz4Compress | 51,756 (5.94 %) | 52,909 (N/A) | 56 (4.17 %) | 48 (N/A) | 1 (0.02 %) | N/A | N/A | N/A | N/A |

2. Computing bound design

| Compute Unit | Device | Number of Transfers | Avg Bytes per Transfer | Transfer Efficiency (%) | Total Data Transfer (MB) | Total Write (MB) | Total Read (MB) | Total Transfer Rate (MB/s) |
|---|---|---|---|---|---|---|---|---|
| xilLz4Compress_1 | xilinx_u50_gen3x16_xdma_201920_3-0 | 3 | 640.000 | 15.625 | 0.002 | 0.001 | 0.001 | 18580.600 |

3. Suspected data transfer rate

**Data Transfer: Kernels to Global Memory**

| Compute Unit Port | Kernel Arguments | Device | Memory Resources | Transfer Type | Number of Transfers | Transfer Rate (MB/s) | vg Bandwidth tilization (%) | Avg Size (KB) |
|---|---|---|---|---|---|---|---|---|
| xilLz4Compress_1/m_axi_gmem0 | in_r\|out_r | xilinx_u50_gen3x16_xdma_201920_3-0 | HBM[0] | WRITE | 1 | 17454.500 | 100.000 | 0.640 |
| xilLz4Compress_1/m_axi_gmem0 | in_r\|out_r | xilinx_u50_gen3x16_xdma_201920_3-0 | HBM[0] | READ | 2 | 19200.000 | 100.000 | 0.640 |
| xilLz4Compress_1/m_axi_gmem1 | compressd_size\|in_block_size | xilinx_u50_gen3x16_xdma_201920_3-0 | HBM[0] | WRITE | 1 | 600.000 | 6.250 | 0.004 |
| xilLz4Compress_1/m_axi_gmem1 | compressd_size\|in_block_size | xilinx_u50_gen3x16_xdma_201920_3-0 | HBM[0] | READ | 1 | 1200.000 | 12.500 | 0.004 |

Suspected

Suggestion for improvement

1. 因為 lab 中的方法只支援 single file processing，一個 file 做完才能做下一個，所以考慮做 task pipeline
2. 因為 top level 只用了 20%的 memory bandwidth，所以可以增加 kernel 的數量
3. Data 寫到 HBM 裡很沒有效率，所以考慮用 AXI_lite 直接寫到 kernel。

Github

https://github.com/Barry-Sung/LAB_C_data_compression