2D convolution with linebuffer

- **Introduction**

This example is a 2D convolution implemented using hls::streams and a line buffer to conserve resources. With HLS, we can quickly do an implementation on FPGA.

- **Solution 1 - using frame buffer**

```
49  template<typename T, int K>
50  static void convolution_orig(
51          int width, int height,
52          const T *src, T *dst,
53          const T *hcoeff, const T *vcoeff)
54  {
```

The input is passed in with const T datatype, which is implemented by HLS with a frame buffer

Utilization Estimates

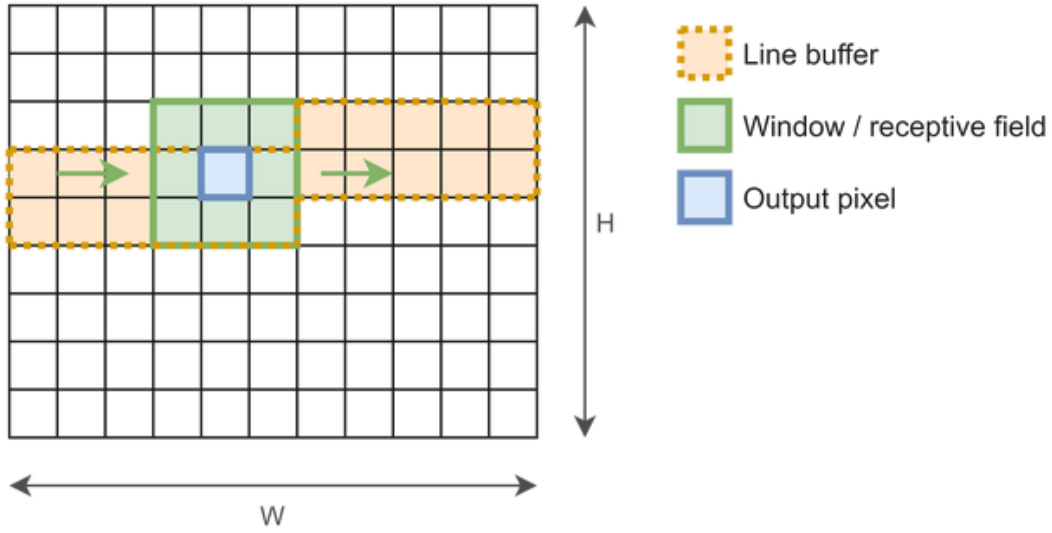
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	56	-
FIFO	0	-	35	308	-
Instance	0	22	2632	3142	-
Memory	4096	-	64	0	0
Multiplexer	-	-	-	72	-
Register	-	-	11	-	-
Total	4096	22	2742	3578	0
Available	280	220	106400	53200	0
Utilization (%)	1462	10	2	6	0

the utilization of RAM explodes.

- **Solution 2 - using line buffer**

The solution is to ensure that data_out and data_in should be accessed as few times as possible. The ideal case is of course that every position gets accessed just once during the whole execution. Luckily, this can be done by using two cooperating data structures: a line buffer and a window.



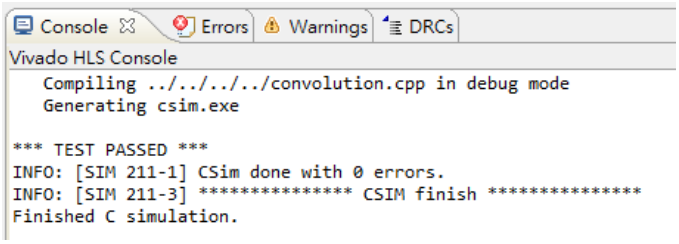
The figure above shows how the input pixels are stored at any point in time for a 3*3 window. The line buffer has dimensions 2*width and serves as sort of a cache that fits just enough elements to fill the rightmost column of the window when it shifts at every new iteration. As such, only one new input pixel has to be read from the I/O port into the window on every clock cycle.

```
186     VConvH:for(int col = 0; col < height; col++) {
187         VConvW:for(int row = 0; row < vconv_xlim; row++) {
188 #pragma HLS DEPENDENCE variable=linebuf inter false
189 #pragma HLS PIPELINE
190             T in_val = hconv.read();
191             // Reset pixel value on-the-fly - eliminates an O(height*width) loop
192             T out_val = 0;
193             VConv:for(int i = 0; i < K; i++) {
194                 T vwin_val = i < K - 1 ? linebuf[i][row] : in_val;
195                 out_val += vwin_val * vcoeff[i];
196                 if (i > 0)
197                     linebuf[i - 1][row] = vwin_val;
198             }
199             if (col >= K - 1)
200                 vconv << out_val;
201         }
202     }
```

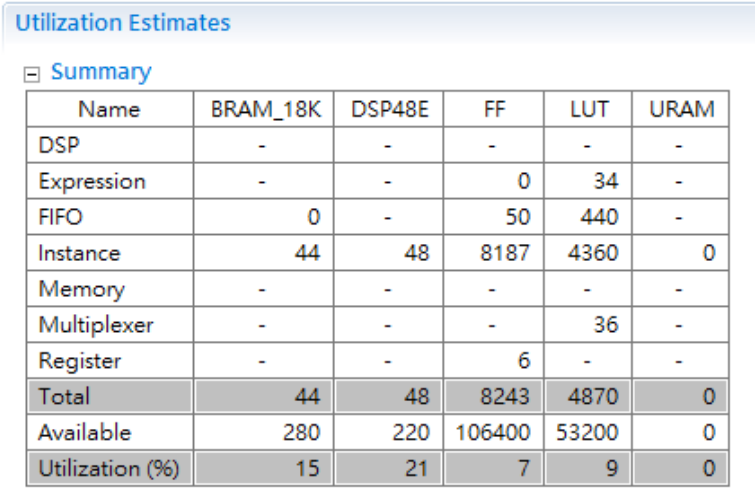
The input is passed in with hls::stream<T> datatype, which is implemented by HLS with line buffers

```
144  static void convolution_strm(int width, int height,
145          hls::stream<T> &src, hls::stream<T> &dst,
146          const T *hcoeff, const T *vcoeff)
147  {
```

C-sim

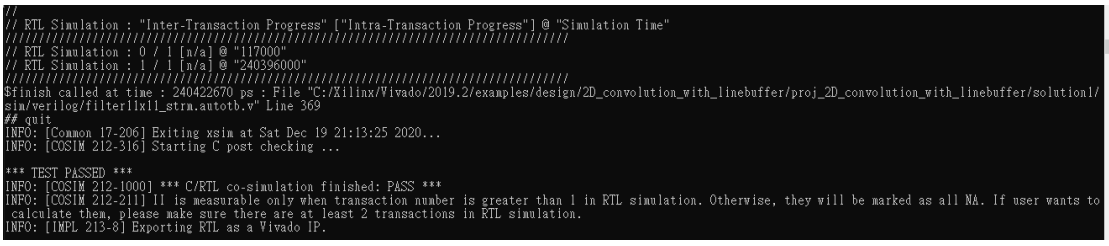


Synthesis



Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	34	-
FIFO	0	-	50	440	-
Instance	44	48	8187	4360	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	44	48	8243	4870	0
Available	280	220	106400	53200	0
Utilization (%)	15	21	7	9	0

Co-sim



<Optimization- pipeline>

Without pipeline

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2203681	2203681	2203681	2203681	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- HConvH	2203680	2203680	9182	-	-	240	no
+ HConvW	9180	9180	68	-	-	135	no
++ HConv	66	66	6	-	-	11	no

With pipeline

```
169     HConvH:for(int col = 0; col < height; col++) {
170 #pragma HLS LOOP_TRIPCOUNT min=240 max=240
171         HConvW:for(int row = 0; row < width; row++) {
172 #pragma HLS LOOP_TRIPCOUNT min=135 max=135
173 #pragma HLS PIPELINE
```

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
32411	32411	32411	32411	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- HConvH_HConvW	32406	32406	8	1	1	32400	yes

After add pipeline pragma, we can have a much lower latency

<Optimization – loop dependence>

Dependency analysis may be too conservative, so we can clarify the dependence

```
188     VConvH:for(int col = 0; col < height; col++) {
189 #pragma HLS LOOP_TRIPCOUNT min=240 max=240
190         VConvW:for(int row = 0; row < vconv_xlim; row++) {
191 #pragma HLS LOOP_TRIPCOUNT min=135 max=135
192 #pragma HLS DEPENDENCE variable=linebuf inter false
```

<Optimization- array partition>

Without array partition

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
324011	324011	324011	324011	none

Do array partition on line buffer can reduce lots of latency of a loop

```
155     static T linebuf[K - 1][MAX_IMG_COLS];
156     hls::stream<T> vconv("vconv");
157     const int vconv_xlim = width - (K - 1);
158     // Line-buffer for border pixel replication
159     T borderbuf[MAX_IMG_COLS - (K - 1)];
160 #pragma HLS ARRAY_PARTITION variable=linebuf dim=1 complete
```

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
32414	32414	32414	32414	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- VConvH_VConvW	32409	32409	11	1	1	32400	yes

<Optimization-fixedtype>

The original code uses unsiged 32 integer for coefficients and accumulators. Here you will use the ap_fixed<16,9> type, representing a 9-bit signed integer with seven decimal bits. This type was chosen because it improves performance and resource utilization, while maintaining the necessary precision for the application.

Include the `ap_fixed.h` header at the top of the file.

```
53  #include "ap_fixed.h"
```

Create a typedef for a fixed point type that maps to `ap_fixed<16,9>`.

```
62  typedef ap_fixed<16,9> data_t;
```

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	34	-
FIFO	0	-	50	408	-
Instance	22	26	4642	3250	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	22	26	4698	3728	0
Available	280	220	106400	53200	0
Utilization (%)	7	11	4	7	0

Detail

<Final version of all optimization included>

C-sim

Console | Errors | Warnings | DRCs

Vivado HLS Console

```
   Compiling ../../../../convolution.cpp in debug mode
   Generating csim.exe

*** TEST PASSED ***
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Synthesis

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	34	-
FIFO	0	-	50	408	-
Instance	22	26	4642	3250	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	22	26	4698	3728	0
Available	280	220	106400	53200	0
Utilization (%)	7	11	4	7	0

Detail

Co-sim

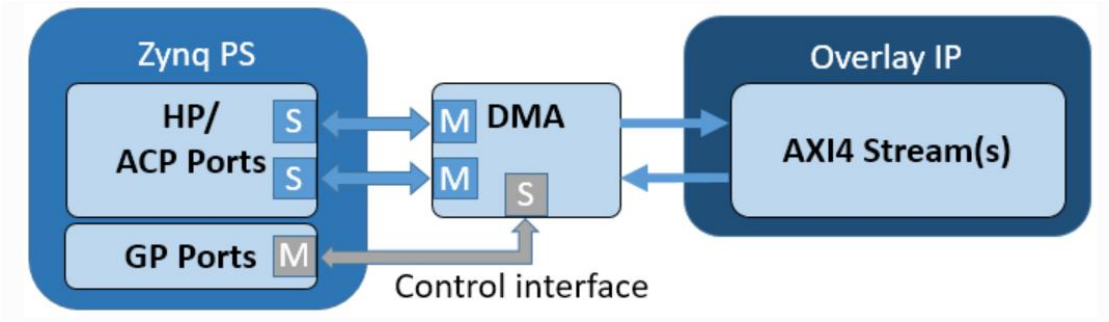
```
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
////////////////////////////////////////////////////////////////////////////////////
// RTL Simulation : 0 / 1 [n/a] @ "117000"
// RTL Simulation : 1 / 1 [n/a] @ "240396000"
////////////////////////////////////////////////////////////////////////////////////
$finish called at time : 240422670 ps : File "C:/Xilinx/Vivado/2019.2/examples/design/2D_convolution_with_linebuffer/proj_2D_convolution_with_linebuffer/solution1/
sim/verilog/filter11x11_strm.autotb.v" Line 369
## quit
INFO: [Common 17-206] Exiting xsim at Sat Dec 19 21:13:25 2020...
INFO: [COSIM 212-316] Starting C post checking ...

*** TEST PASSED ***
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: [IMPL 213-8] Exporting RTL as a Vivado IP.
```

- **System level bring-up (Pynq or U50)**

  Code modification

  If there're stream interface in the design, the PS to kernel function may require a master to stream conversion, which require DMA.

  The DMA has an AXI lite control interface, and a read and write channel which consist of a AXI master port to access the memory location, and a stream port to connect to an IP.



  The read channel will read from PS DRAM, and write to a stream. The write channel will read from a stream, and write back to PS DRAM.

  Note that the DMA expects any streaming IP connected to the DMA (write channel) to set the AXI TLAST signal when the transaction is complete. If this is not set, the DMA will never complete the transaction. This is important when using HLS to generate the IP - the TLAST signal must be set in the C code.

  Due to the DMA IP in vivado have a TLAST signal, so you need to use ap_axiu datatype, which contain data, keep, last signals etc.

```
67   typedef ap_axiu<32,1,1,1> value_t;
68   typedef hls::stream<value_t> AXI_STREAM;
```

```
295  void filter11x11_strm(
296          int width, int height,
297          AXI_STREAM& src, AXI_STREAM& dst)
298  {
299  #pragma HLS INTERFACE axis port=src bundle=IN
300  #pragma HLS INTERFACE axis port=dst bundle=OUT
301
302  // #pragma HLS INTERFACE m_axi port=width offset=slave bundle=width
303  // #pragma HLS INTERFACE m_axi port=height offset=slave bundle=height
304
305  #pragma HLS INTERFACE s_axilite port=width
306  #pragma HLS INTERFACE s_axilite port=height
307  #pragma HLS INTERFACE s_axilite port=return bundle=control
308
309  #pragma HLS DATAFLOW
310  #pragma HLS INLINE region // bring loops in sub-functions to this DATAFLOW region
311
312
313      const data_t filt11_coeff[11] = {
314          36, 111, 266, 498, 724, 821, 724, 498, 266, 111, 36
315      };
316
317      convolution_strm<data_t, 11>(width, height,
318          src, dst,
319          filt11_coeff, filt11_coeff);
320  }
321
```

Inside kernel function

Create valTemp as ap_axiu datatype, and then read in source, and assign to in_val.

```
170      HConvH:for(int col = 0; col < height; col++) {
171  #pragma HLS LOOP_TRIPCOUNT min=240 max=240
172          HConvW:for(int row = 0; row < width; row++) {
173  #pragma HLS LOOP_TRIPCOUNT min=135 max=135
174  #pragma HLS PIPELINE
175              value_t valTemp = src.read();
176              T in_val = valTemp.data;
```

After computation,

Create valTempOut as ap_axiu datatype to store output

```
209      value_t valTempOut;
210      //Handle border by replicating the exact same pixels as orig, but in
211      // a single loop taking the minimum (height*width) number of cycles
212      bool sof = 1;
```

Furthermore, pull up the TLAST signal in kernel function in order to let the write_dma detect the TLAST signal.

=> assign valTempOut.last = 1

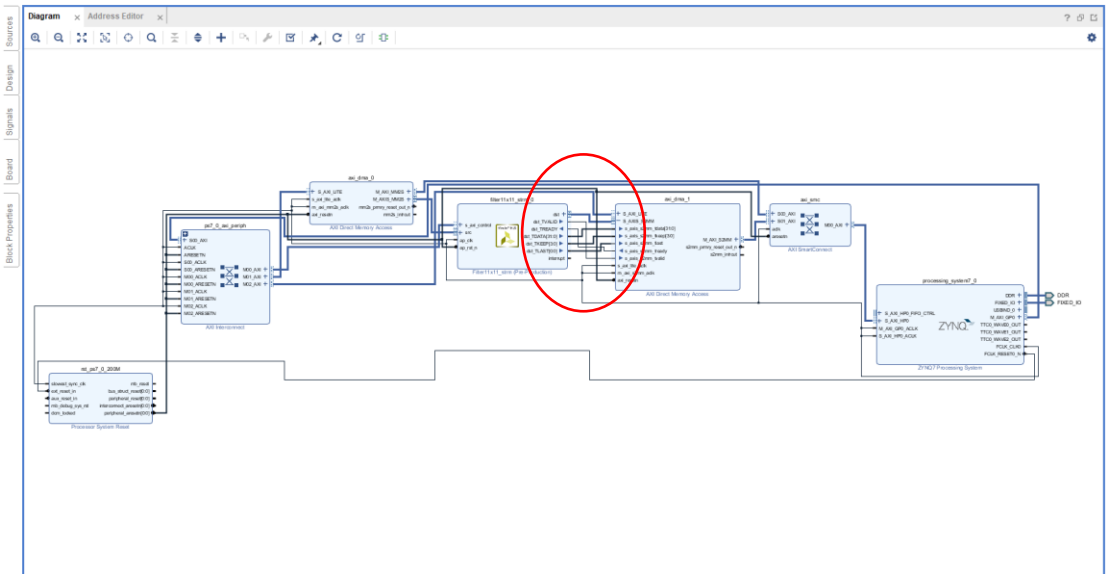
If necessary, assign .user, .keep signals…etc. also!

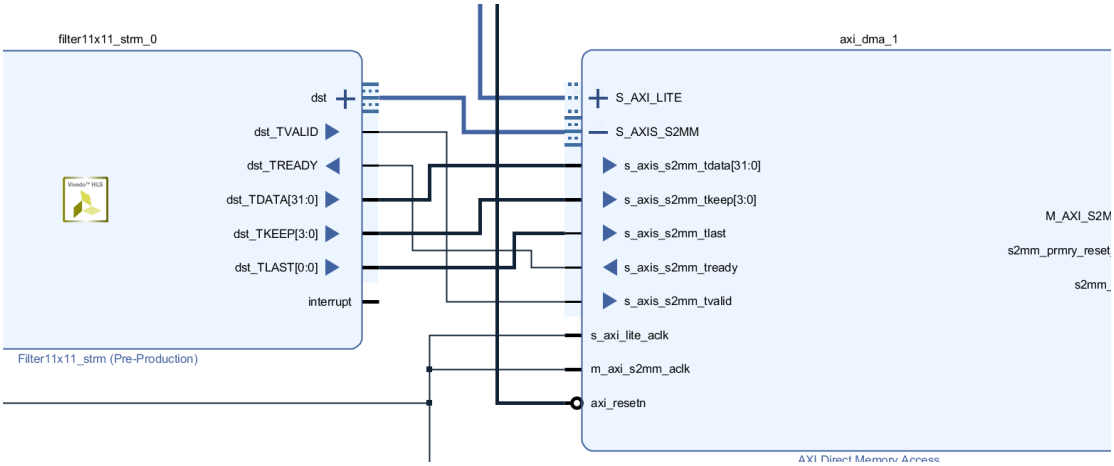
```
239
240             valTempOut.keep = -1;
241             if (sof) {
242                 valTempOut.user = 1;
243                 sof = 0;
244             } else {
245                 valTempOut.user = 0;
246             }
247             valTempOut.data = pix_out;
248             if (i==(height-1) && j==(width-1))
249             {
250                 valTempOut.last = 1;
251             }
252             else
253             {
254                 valTempOut.last = 0;
255             }
256             dst << valTempOut;
257
258             if (valTempOut.last) break;
259         }
260     }
```

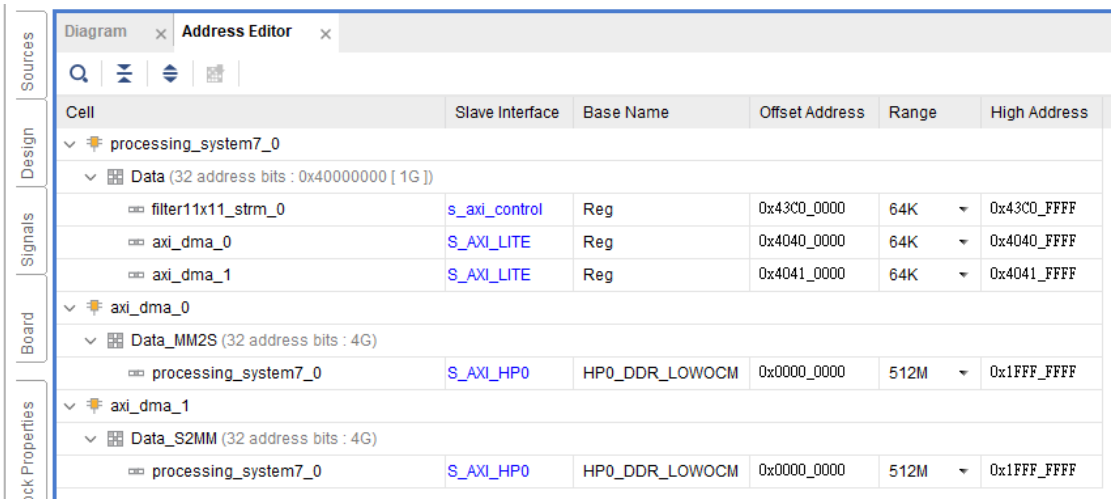
Check the block diagram, you should have the corresponding interface.

Debugging skills:

Open synthesis => set up debug you can check whether all signals on the interface is defined.







Host program

Program to FPGA

```
24    ol = Overlay("/home/xilinx/IPBitFile/yclin/conv_desing.bit")
25    ipConv = ol.filter11x11_strm_0
26    ipDMAin = ol.axi_dma_0
27    ipDMAout = ol.axi_dma_1
```

Allocate buffer

```
inBuffer0 = allocate(shape=(numSamples,), dtype=np.uint32)
outBuffer0 = allocate(shape=(numSamples,), dtype=np.uint32)
for i in range(numSamples):
    inBuffer0[i] = src[i]
```

Run and check DMA status

```
57      ipConv.write(0x10, int(TEST_IMG_COLS))
58      ipConv.write(0x18, int(TEST_IMG_ROWS))
59      ipConv.write(0x00, 0x01)
60      ipDMAin.sendchannel.transfer(inBuffer0)
61      ipDMAout.recvchannel.transfer(outBuffer0)
62      ipDMAin.sendchannel.wait()
63      ipDMAout.recvchannel.wait()
```

```
finished sendchannel.transfer
finished recvchannel.transfer
finished sendchannel.wait
finished recvchannel.wait
Index 0: 2205403455
Index 1: 2205403455
Index 2: 2205403455
Index 3: 2205403455
Index 4: 2205403455
Index 5: 2205403455
Index 6: 2322483390
Index 7: 2366794230
Index 8: 2322483390
Index 9: 2205403455
```

- **Github submit**
  https://github.com/tingyungchen/2D_convolution_with_linebuffer