Line Buffer block for CGRA

Assumptions

- 16bit pixel
- 4K / 12MPx image
- Support data directly from sensor - full width, no striding
- Most kernels are 3x3 or 5x5, 3 channels
- No stalls in the implementation
- Each cycle write 1 pxl in raster order and read 1 *column* of pixels
- Boundary conditions: Fill or Repeat

Choosing the size

RAM macro library elements

There is a variety of library RAM blocks available in different configurations. In general, blocks with smallest capacity have best *access energy per byte* while blocks with highest capacity have best *area per byte* (see Fig. 1).

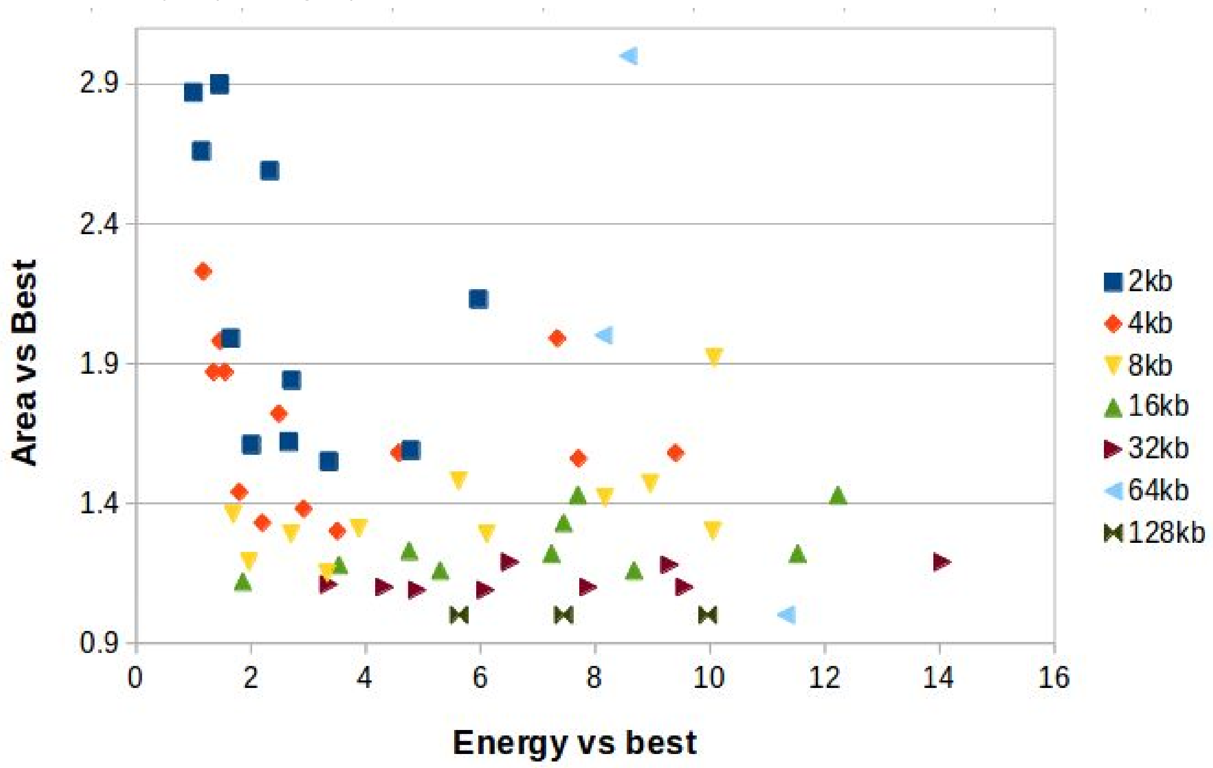

Figure 1: SRAM blocks energy/area based on 28nm library

For a programmable fabric, both energy and area efficiency parameters are important, thus we are interested in the lower-left corner of the graft. There two blocks emerge: 8kb and 16kb.

We expect that ISP will be processing 4K images (~12MPx) and each pixel will be 16bit. Additionally, we know that ISP pipelines are composed of kernels that process odd sizes of stencils. A typical 3x3 stencil has to store two full rows or 2*4k*16b =16 kB, so 16kB macro is the best choice both because of its parameters and because of application needs.

Control logic and extras for a typical case

The macro we've chosen has a single 128bit read/write port however, logically we need to write one 16bit and read three 16bit pixels, one of which is a copy of the input. So the bandwidth requirement is only 48bit/cycle. To make it work with our block we need to buffer writes and reads and have some control logic that would manage this process - see Fig. 2.

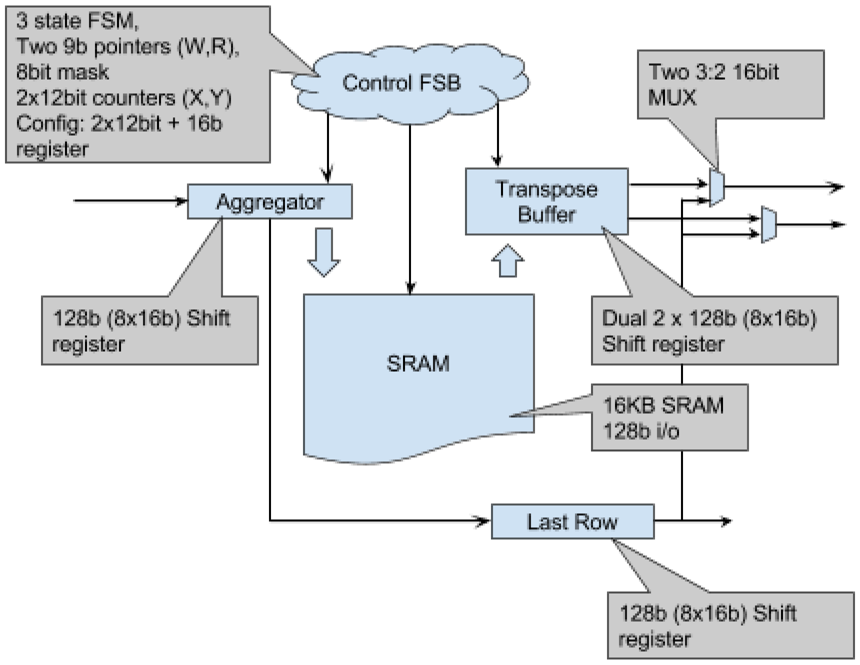

Figure 2: Typical (smallest) case

Description of control elements

SRAM - 16kB single port. 128bit x 1024 words
Aggregator - buffer inputs until we can do a write. This is just a 128bit shift register (8 deep 16bit wide)

Transpose Buffer - buffers read data and transposes it. This is two sets of two 128bit shift registers (8 deep 16bit wide). Need two sets because we will be doing shifts from the first set while we are filling up the second set.
Note: Second set is mostly needed for energy saving and virtualization. In basic case one set is enough because we have more than 2x the bandwidth from SRAM. This allows us to treat memory as 64bit interface and ignore half of the reads.
Note 2: The data stored in the transpose buffer is the same as in "shift registers", which are implemented in fabric. In fact, even the behaviour is similar - "transpose buffer" also does data shifts, however we need both structures in CGRA. The reason is that without the shift register, the number of tracks that LB block has to drive would increase by at least 6 (9 data outputs instead of 3) which would require larger SB in everywhere. The second reason is the timing - implementing shift registers in the fabric allows them to be distributed and to reduce the fanout of a LB block.
Last row - copy of the "Aggregator" to provide delayed version of the inputs. This is required to support boundary conditions and it's needed by application
Boundary support - two 3:1 16bit muxes that select between: transpose buffer value (normal case) or constant form a register (fill with value boundary case) or form 'Last row' (repeat boundary)
Control - control logic that manages read/write cycles of the SRAM, generates various signals: write mask, enables to the shifters, select of MUXes,etc. At the minimum it needs to know image dimensions (two 12bit reg), have read/write pointers (two 9bit reg), track position of the read stencils centroid (two 12bit reg), store fill value for the boundary (16bit register), have small FSM (~10 states) and various misc control (~16-32bit of registers), another 8bits are needed for the write mask

Virtualizing LB size

Multiple channels

Channels operate independently and can be supported by multiple LB blocks - one block per channel

Larger stencils

There are two possible approaches: "rotating" and "dizzy chain".

Rotating scheme uses several LB blocks such that input is shared among all of them. Control logic has to be aware of this mode - it will generate less enables to the aggregator and less writes to the SRAM. Additionally, general fabric will have to deal with "circle shift" of produced rows, depending on the position. This can be done with MUXes -- one per output -- and Y position (supplied by the block) can be used as the MUX select.

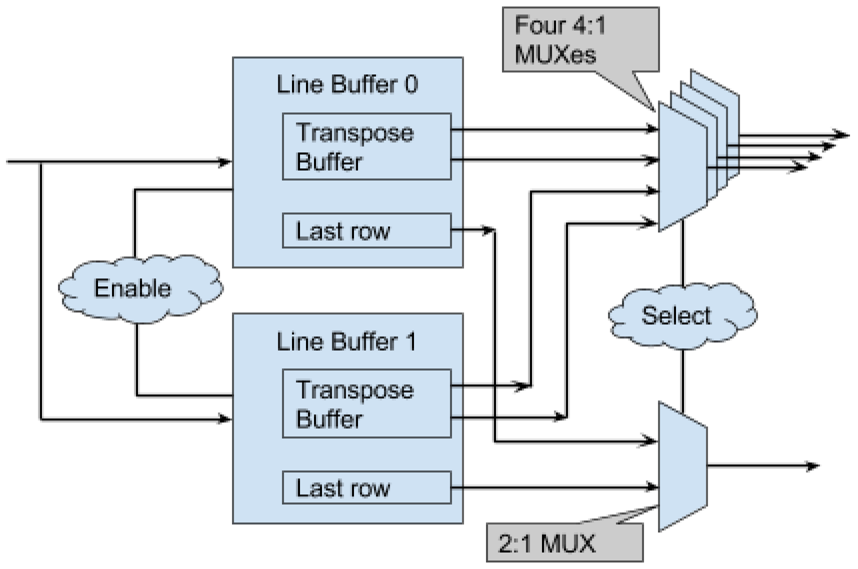

Figure 3: LB for larger stencil with rotating scheme

Alternative is to dizzy chain multiple LB blocks such that the last output of the first block is an input to the second and so on. This avoids “circle shift” but reads and writes the same pixel multiple times in the LB by copying it from one SRAM to the next and dissipates more power. For this reason, the first approach is preferred.

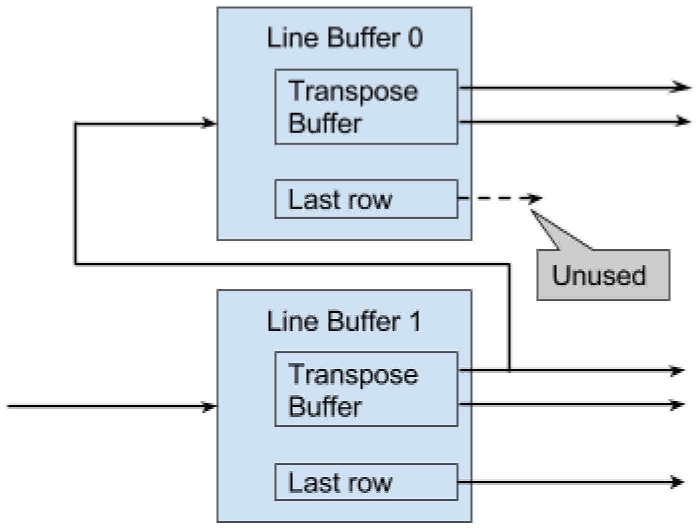

Figure 4: LB for larger stencil with dizzy-chain scheme

The choice between the two is essentially a choice between building a (N-1) to (N-1) crossbar in general fabric or writing each pixel (N-1)/2 times to SRAM instead of just once. Mostly likely, for for small N, crossbar would require less energy, while for the large N the situation will be reversed. However, because of wide SRAM interface, writes don’t happen every cycle (in fact we need ~½ the writes of 48bit SRAM) the energy advantage of crossbar is small even for the case of 2 LB blocks. Also this is case is easier much easier so we’ll use it for the model.

Wider images

The easiest case is to “stripe” and use several LB blocks to store one line. Alternatively, we can modify existing block to store one wide line, but this could lead to inefficient memory use and to the loss of block’s bandwidth

Narrow images

The problem with narrow images is that we can have a lot of unused space in the LB block if we use it as is. To reduce the waste we could either support more rows or support multiple independent buffers in the same block. Both cases would be limited by the SRAM bandwidth. Given that the memory block has 128bit or 8 16bit pixels interface and that each LB has to store 1 pixel each cycle, we can build one LB with up to 7 lines in SRAM or two LB with 3lines, etc.

Increasing maximum number of lines per LB would increase the height of the transpose buffer accordingly, while supporting more independent LBs would require replicating the entire control structure (aggregator, transpose buffer, FSM, last buffer, etc.). So a tall LB is a bit more efficient, but less common in the applications.

Final consideration is how many tracks we need to support such a block. With our choice of interconnect (1,4,8 long wires; 12 tracks) we have 6 outputs per direction and given that SRAM itself spans 2-3 tiles high, we have at least 2*6*2 = 24 outputs (and similar number of inputs). The base case interface needs 1 wire, 2+1+[2] = 5 outputs and 1 input.

Overall, for simplicity we assume that smaller images are supported by mapping independent LBs tot the black and as we have just established there could be only 2 such LBs in a block.

Because we can “split” our LB only in two, we can support images of around 2K pxl relatively efficient, if we need to support smaller images, the only good option would be to use smaller SRAM. However, with smaller SRAM, the relative cost of all the extra logic will be significantly higher.

Area overhead estimation

For a base case, we need at least 922bits in registers across various elements. Assuming SRAM cells are 6 times smaller that a register, that gives 4.2% area overhead,compared to a 128bitx1024 word SRAM block.

With support for smaller images, the overhead would roughly be twice that size or ~9%

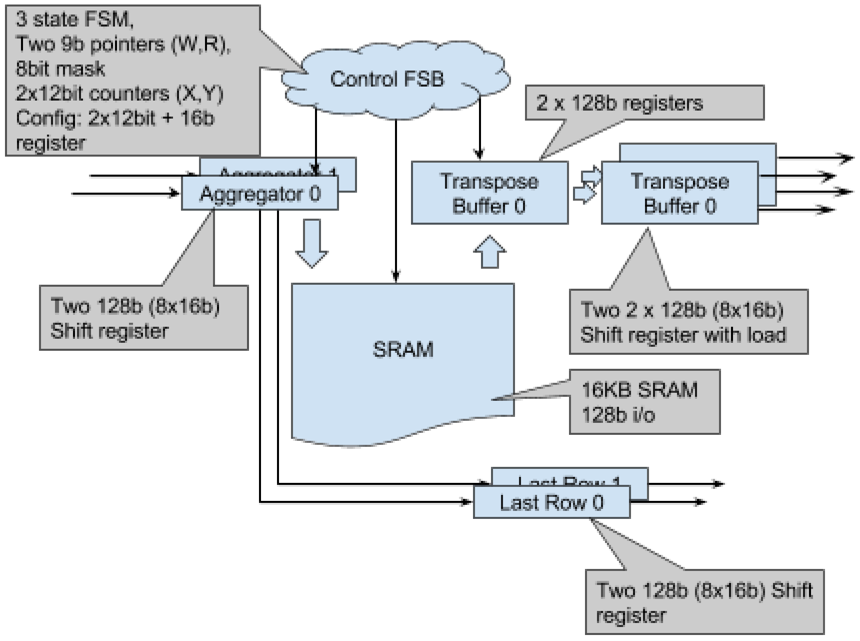

Figure 5: Support for two LBs and for smaller image width