Adapt Net

Architecture Specification

(version v1\_00\_a)

TABLE OF CONTENTS

Convolution Engine

Adaptive Window Engine

Quad

Fetch Aggregate Store

Configuration

Figure [] is a visual depiction of how convolution layer is performed on an input map. A DxWxH input map is convolved against N DxKxK kernels which are summed across the D dimension to produce NxWxH maps. This then serves as the input to the next layer.

Fig. []

Fig. [] Convolution Engine Unit Architecture



Figure [] shows the architecture of a convolution engine (CE). Each engine runs at 5x the system clock rate consists of two multiply and accumulate (MAC) units which are fed by a weight table holding 3x3 weights, and by a collection of row buffers which hold 3 rows of pixels with max size of 1024. The input to the row buffers comes from either a prefetch buffer (PFB), or an activation unit whose input is the PFB. The activation unit is present for the cases when intermediate feature maps from a previous layer needs to be activated.

Fig []

The row buffers comprise of two BRAMS which are partitioned into two sections, totaling four sections total. The row buffers implement a 3x3 sliding window for incoming pixel data. This is accomplished by loading the rows into the correct row buffers then over time matriculating the incoming row and current rows and Figures [] and [] depict the matriculation pattern and row layout.

Fig []

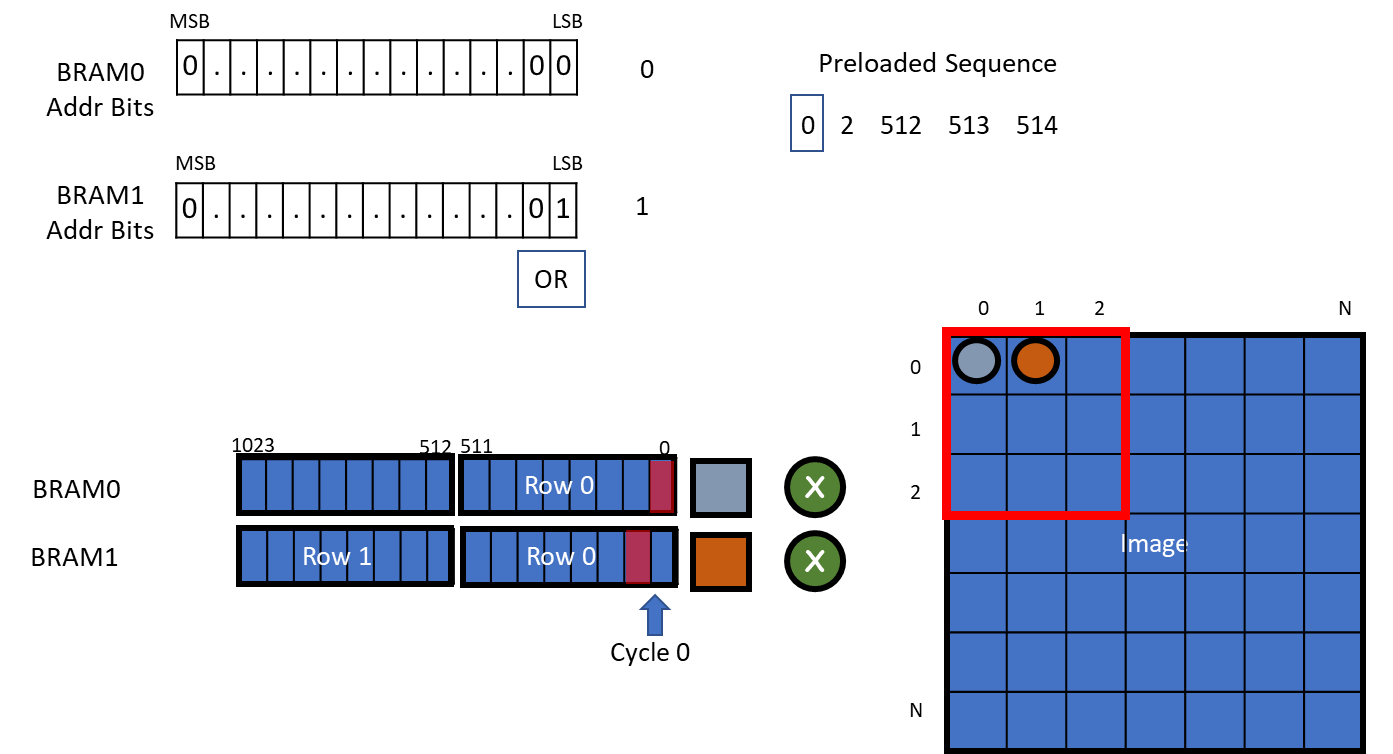
To aid in the row matriculation process, the accelerator is preloaded with pixel sequence data which contains the read addresses for the row buffers. The MSB bit of the BRAM address is properly toggled during reading and writing operations along with logical operations between the LSB of the address and specific flags. Figures [], [], [], [], [] shows this process as well as illustrating the output pattern for a 3x3 window. Thus it takes 5 cycles to output an entire 3x3 window.

Fig []

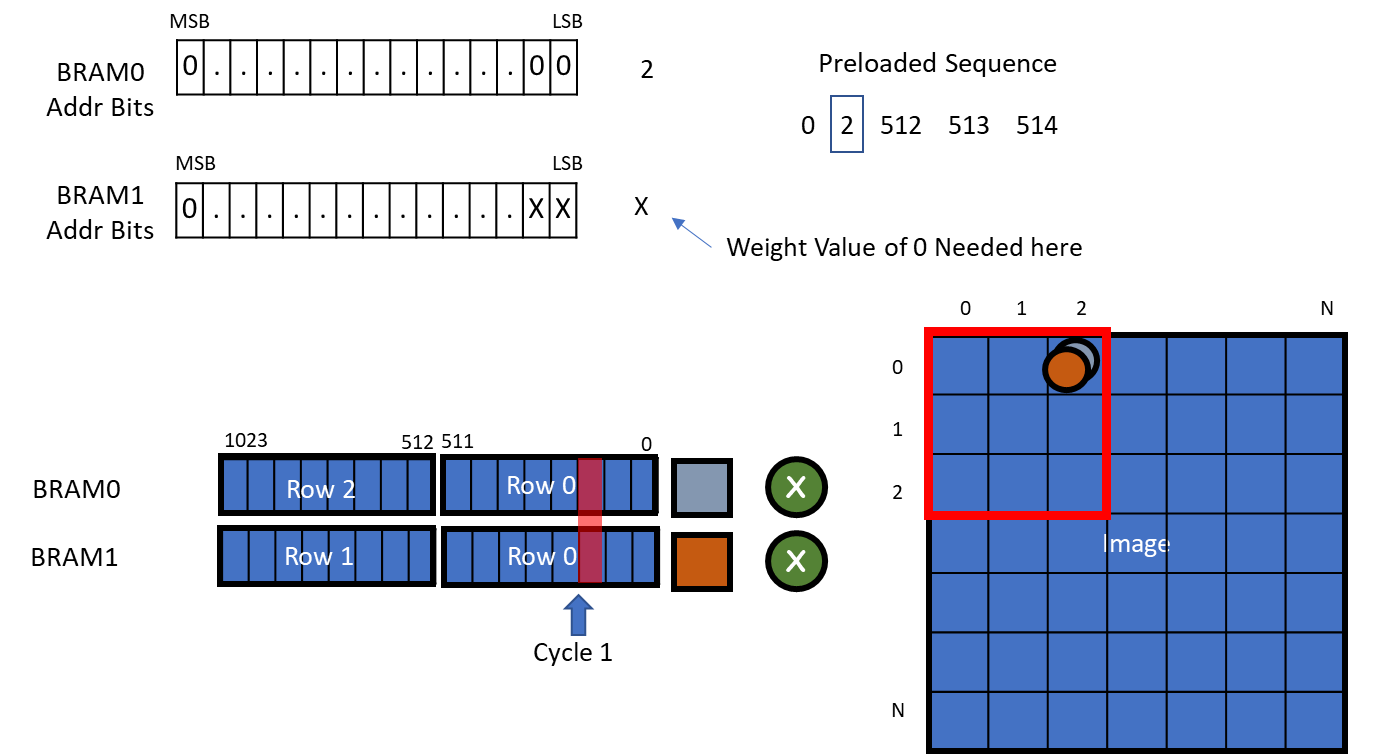
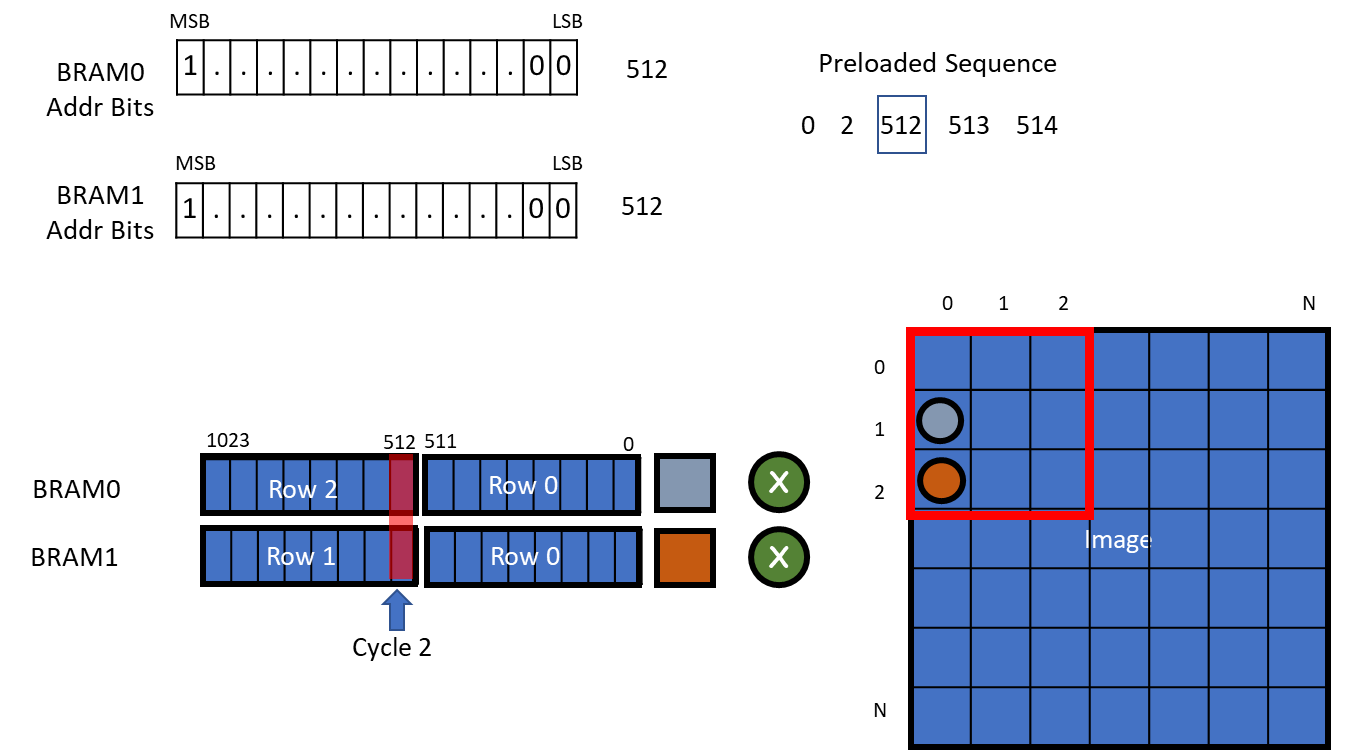


Fig []

Fig []

Fig []

Fig []

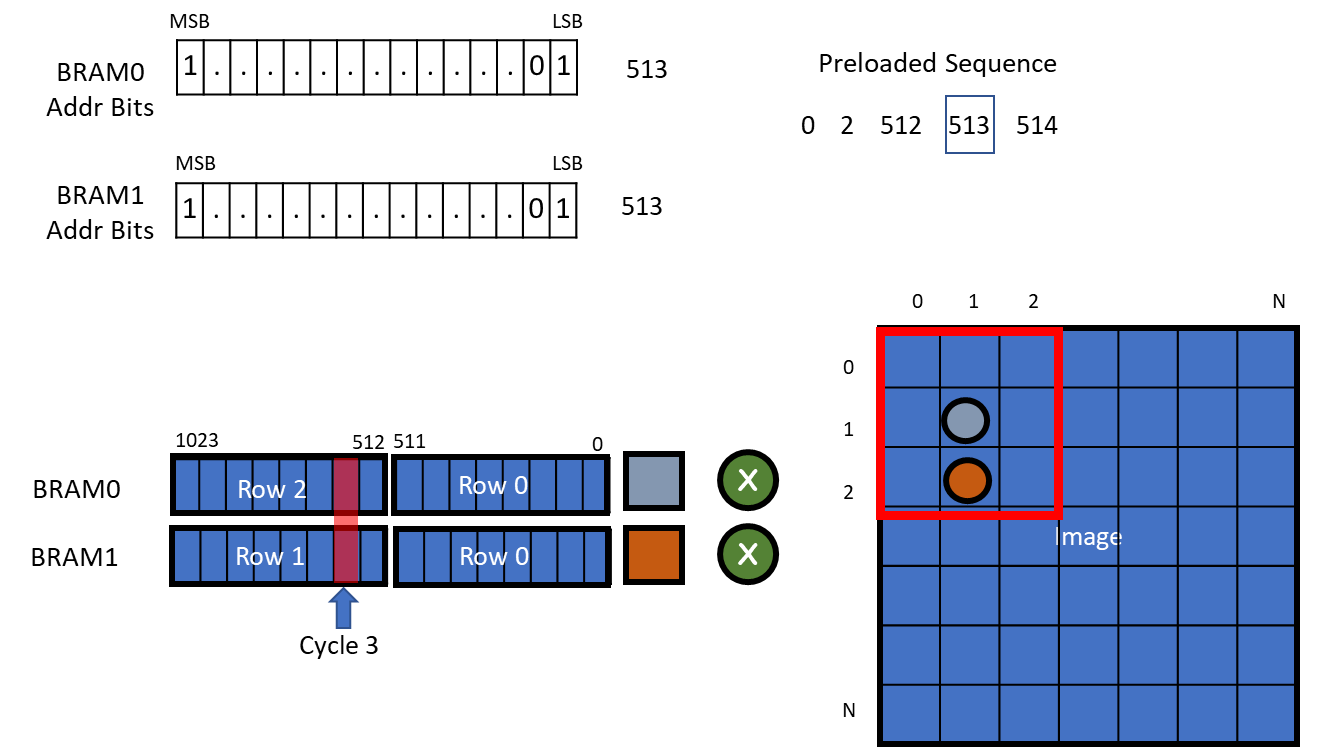
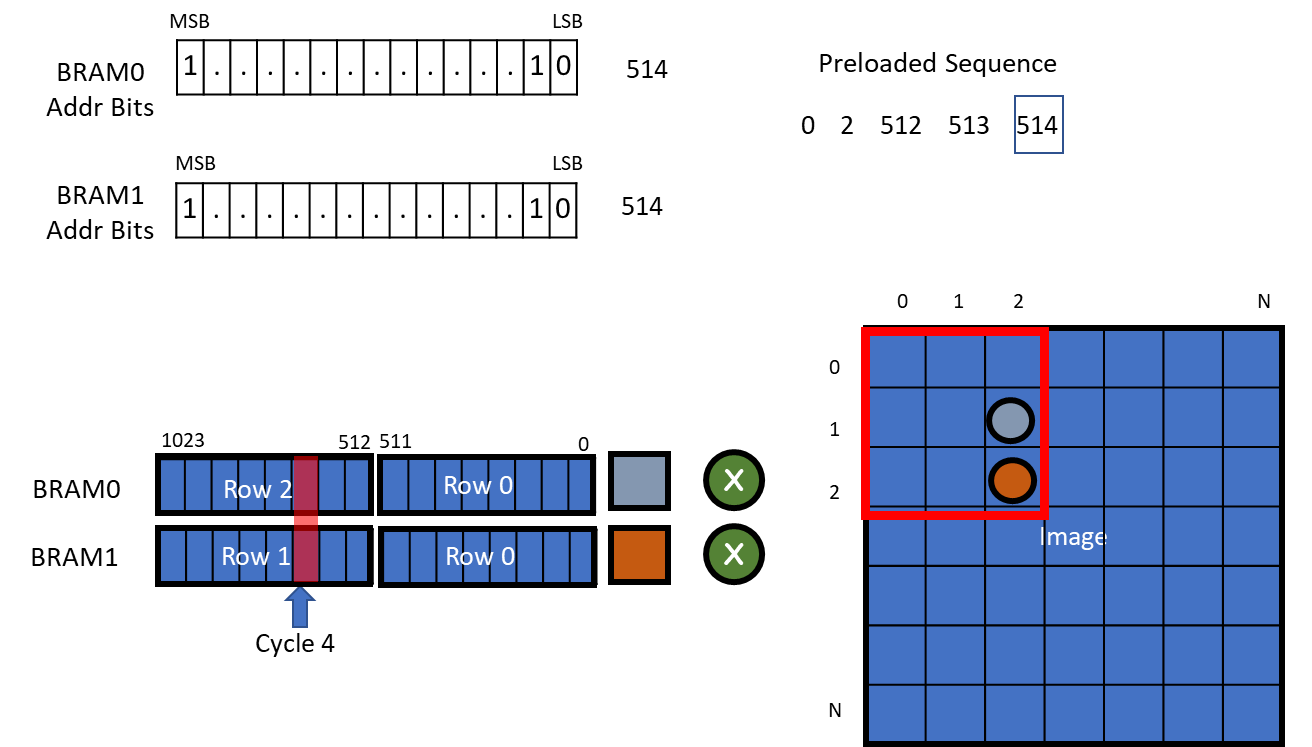


Fig []

The CE’s are grouped into pairs to create an adaptive Window Engine (AWE) as shown in Figure []. This grouping provides a method for the AWE to convolve two 3x3 windows, or for the unit to combine both CE’s to convolve one 5x5 window (future architecture).

Fig []

An AWE is capable of convolving a 3x3 window from two Feature maps or one 5x5 window for a single Feature Map. To further process a convolutional layer along the depth dimension multiple AWE’s can be stacked together to work in tandem to form a quad as shown in figure [].

Fig []

A Quad is able to convolve a 3x3 window from 8 Feature Maps (ie depth 8) Additional Quads can be allocated to processes more Feature maps simultaneously. In addition to, the QUADS can be cascade with one another if the user wishes to process a convolutional map with a depth larger than 8. Because of practical limitations only a handful of QUAD’s can be allocated. Thus the user will have to iterate the feature maps through the QUAD over time.

Fig []

Each Quad will produce a set of feature maps which then must be aggregated across the depth dimension and sent to intermediate storage. To facilitate the aggregation, storage and, as well as fetching feature maps, the partial maps are sent to the Fetch Aggregate and Store (FAS) unit. Figures [] depicts the architecture of the Aggregation function of the FAS unit. For Feature Maps whose depth cannot fit entirely in the QUAD and you must iterate. Figure [] depicts the pathway on the first iteration where the partial results are accumulated and sent to system memory. Figure [] shows the path where stored featured maps are brought into the system to be accumulated against the partial maps from the quads. The FAS has an additional weight table which is used for 1x1 convolutions as depicted in figure [] in the cases where 3x3 layers are followed by 1x1 layers.

Fig []



Fig []

Fig []

Fig []

There are 17 configuration registers for each Quad that need to be set before the start of a convolution operation. Table [] lists the long names of these registers, their widths, and their purpose.

|  |  |  |
| --- | --- | --- |
| Register Name | Width (bits) |  |
| Stride | 7 | Indicates the convolutional stride between 1 and 2 |
| Convolution output Format | 1 | Changes the output format for 3x3 convolutions which are followed by a 1x1 layer. 0 is for no |
| Padding | 5 |  |
| Number of output Columns | 11 |  |
| Number of output Rows | 11 |  |
| Pixel Sequence Data Full Count | 11 |  |
| Upsample | 1 |  |
| Number of Expanded input Columns | 11 |  |
| Number of Expanded input Rows | 11 |  |
| Cropped Input Column Start | 11 |  |
| Cropped Input Row Start | 11 |  |
| Cropped Input Column End | 11 |  |
| Cropped Input Row End | 11 |  |
| Number of Kernels | 6 |  |
| Master Quad | 1 |  |
| Cascade | 1 |  |
| Activation | 1 |  |

During transmission from system memory to accelerator memory, 3x3 kernel data needs to be laid out in row major format. Because of the nature of the CE’s each individual filter of a 3x3 kernel needs to be appended with an extra zero at the end. A CE’s weight table can hold a maximum of 64 kernels. Therefore, a quad can at max perform convolution with 64 kernels of max depth 8. More quads can be allocated to work on the similar pixel data with a different slice of kernels, but iterations of kernel groupings are required if an entire kernels depth cannot fit inside the accelerator memory. Image data is also transmitted in row major format from system to accelerator memory. Figures [] and [] illustrate example workloads and how kernels and images should be packed.