

Embedded System Design

We selected the Sobel filter as the focus of our project. The processing pipeline is divided into two main stages, executed in parallel through buffering:

1. Grayscale conversion of color pixels
2. Sobel filter computation

These stages are pipelined using a DMA and custom instructions as follows:

1. Load color pixels
2. Convert them to grayscale
3. Buffer three distinct lines
4. Apply the Sobel filter
5. Store the Sobel output

To implement this, we partitioned the DMA memory into eight segments (see diagram). Half is used for a ping-pong buffer handling 16-bit color pixels. During the "ping" phase, pixels are loaded; during "pong", grayscale values are computed and stored in a second ping-pong buffer (`sobel1`) with four levels. Once three grayscale lines are ready, the Sobel filter is applied using the `sobel2`, `sobel3`, and `sobel4` buffers. With a one-cycle delay, results are written back into `sobel2`, as its data is no longer required.

To simplify edge handling, the Sobel filter operates line by line, setting border pixel values to zero. Since both grayscale and Sobel pixels are 8-bit wide, a single line requires 640 bytes. With four Sobel lines and two color lines (16-bit), the total required memory is $640 \text{ bytes} \times 8 = 5 \text{ KB}$. Consequently, we increased the DMA memory size from 2 KB to 5 KB. In addition to the DMA, we implemented a Verilog module that computes the Sobel filter for better optimization, along with a large software program that manages all the buffers and timing delays required to avoid memory issues.

