

Hardware Orientation Binning in Simulink

An Application Note

John Buether

4/1/2011

This document will explain how to create a structure for segmenting orientation data in the form of x and y magnitude without the use of trigonometric functions in MATLAB R2008a's Simulink add-on, with Xilinx block libraries. It is assumed the user has some basic understanding of Simulink.

Table of Contents

Introduction	2
Implementation	2
Math.....	2
Laying It Out in Simulink	3
Simulation	6
References	8

Introduction

The Histogram of Gradients algorithm requires binning of orientation data. The bin of an angle is determined by the delimiting angles it is between. Finding this bin is trivial to do with simple trigonometric functions; however, trigonometric functions are slow and awkward in hardware, so it is best to avoid them where possible. This paper will demonstrate one way of avoiding these calculations in Simulink, and simulate the result. This example will create nine 20 degree bins (that ignore the sign of the magnitude) using comparison logic, and a lookup table.

Implementation

Math

The bin of an angle is determined by the delimiting angles it is between, so to determine which bin a Gradient vector $G(x,y)$ belongs to, it is necessary to compare it's corresponding angle, calculated by $\alpha = \arctan(G_y/G_x)$, to the delimiting bin angles $\{0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ, 100^\circ, 120^\circ, 140^\circ, 160^\circ, 180^\circ\}$. Within the first quadrant, if α and β are angles $0 < \alpha < \beta \equiv 0 < \tan(\alpha) < \tan(\beta)$. In figure 1, it can be seen how to use this identity to eliminate the evaluation of any trigonometric functions. Using this equivalence and information about which quadrant the vector lies in, the bin number can be determined, as in figure 2.

$$\begin{aligned} 0 < \tan(\alpha) < \tan(20^\circ) \\ 0 < \left| \frac{G_y(x,y)}{G_x(x,y)} \right| < \tan(20^\circ) \\ 0 < |G_y(x,y)| < \tan(20^\circ) \cdot |G_x(x,y)| \\ 0 < |G_y(x,y)| < 0.364 \cdot |G_x(x,y)| \end{aligned}$$

Figure 1

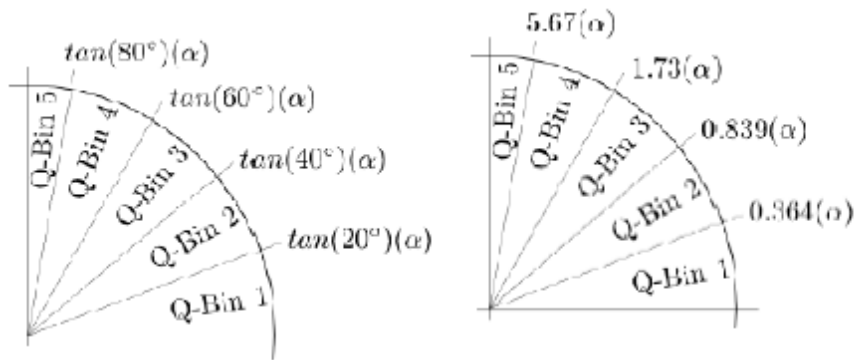


Figure 2

Laying It Out in Simulink

First, open MATLAB. From the start menu at the bottom of the MATLAB screen, open the Simulink Library Browser (SLB) and create a new project. Use Simulink's Commonly Used Blocks, and Xilinx's Basic Elements in the Library Browser to construct figure 3. The System generator block is required any time blocks from the Xilinx Blockset are used. The values in the generators will be used later for simulation, and the scope will display the simulation values. The bin_select block is a subsystem block.

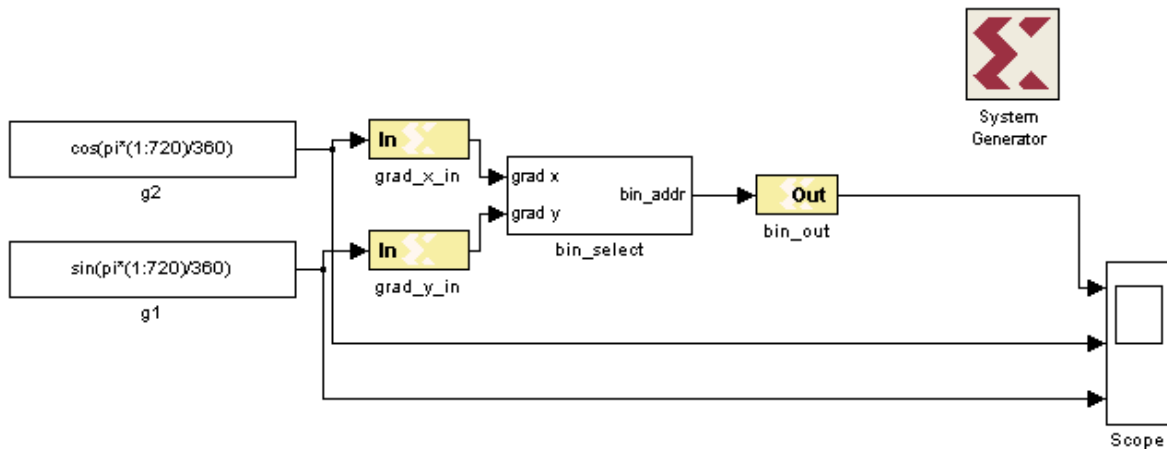


Figure 3

The sign bit, and the absolute values of the x and y gradients are needed, so construct an absolute value subsystem block as seen in figure 4. The slice block property options should be set as seen in figure 5, to extract the sign bit.

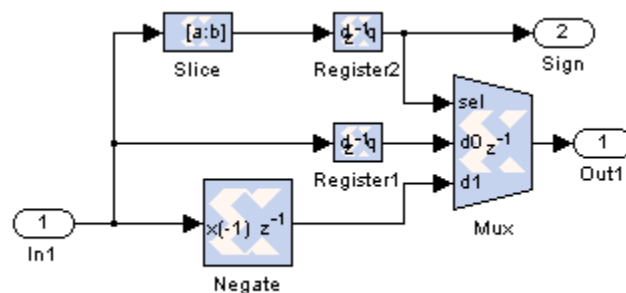


Figure 4

Basic

Advanced

Width of slice (number of bits) 11

☐ Boolean output

Specify range as:

☐ Two bit locations
 ☒ Upper bit location + width
 ☐ Lower bit location + width

Offset of top bit 0

Relative to:

☐ LSB of input
 ☐ Binary point of input
 ☒ MSB of input

Offset of bottom bit 0

Relative to:

☒ LSB of input
 ☐ Binary point of input
 ☐ MSB of input

Figure 5

The next section will detail recreating figure 6 seen below.

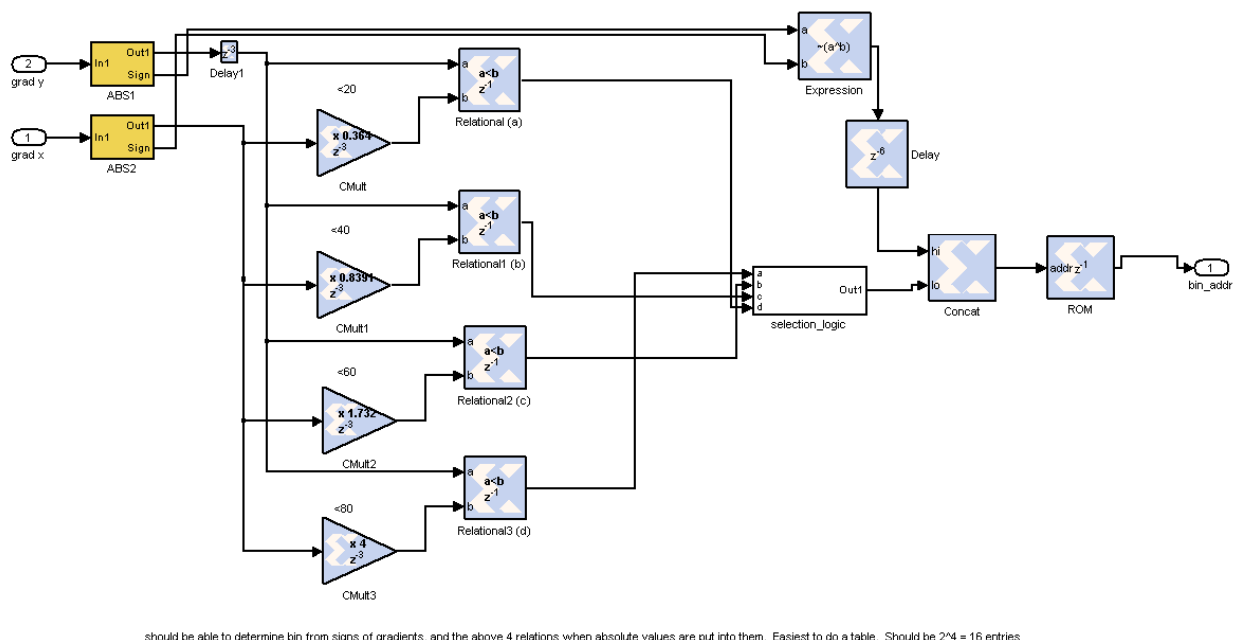


Figure 6

The first quadrant gradient values coming out of the yellow absolute value blocks in figure 6 will be compared to each other as in figure 1 using Xilinx Constant Multiplier and Relational blocks. Set the latency on the Multiplier blocks to 3 – this is required for optimum pipelining. Set the relational block latency to 1 – they will be powering some zero latency bitwise logic in the subsystem block following the relations. Delay the y gradient by 3 so that it matches the delay on the multiplier. The values in the

constant multipliers can be set in the properties menu. Note that they should be the same floating point values as are seen in right half of figure 2. The expression block has both sign bits as inputs, and computes not (a xor b). The output of the expression block needs to be delayed by 6 to cancel with the delays from the absolute value, multiplier, and relational blocks. The purpose of the selection block is to condense the information from the relational blocks from four bits with five possible patterns {0000, 0001, 0011, 0111, 1111} down to three bits by any means possible. The logic seen in figure 7 below is one way to accomplish this. This step, and this subsystem, can be skipped by later doubling the size of the ROM that will contain the final bin values, and changing the ROM initialization vector accordingly.

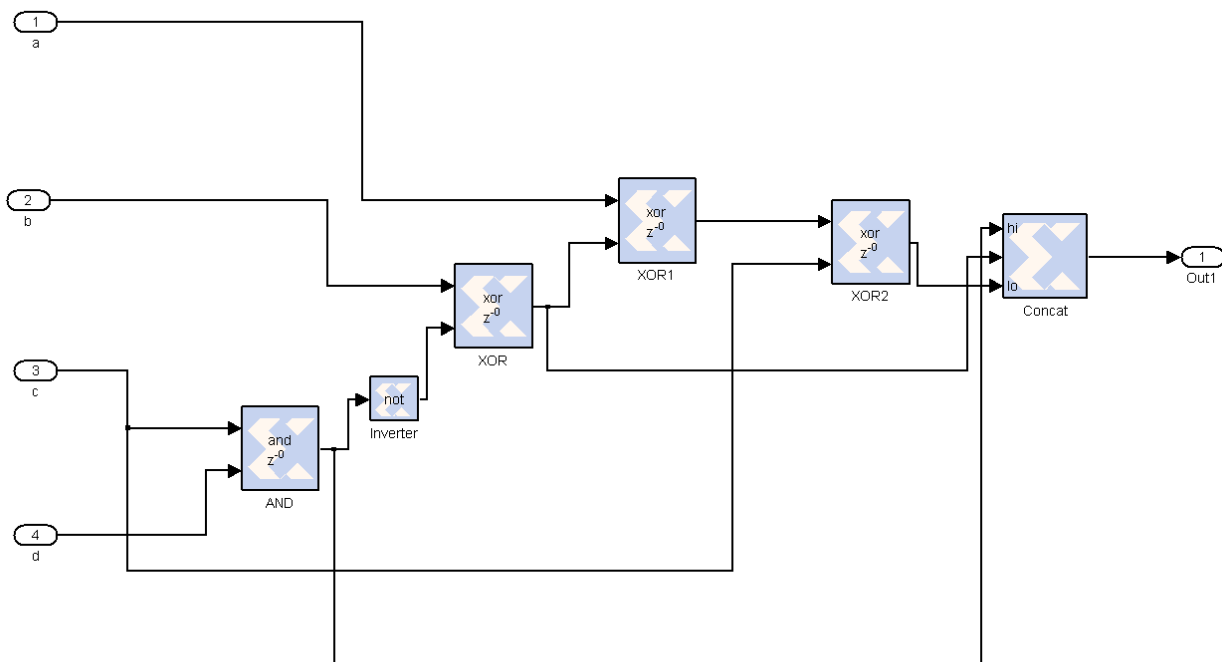


Figure 7

The three bit output from the selection logic block in figure 6 is concatenated with the combined sign bits to form a four bit address which we will use to access a lookup-table stored in ROM. For this example the rom should be set as in figure 8 below, with an output type of 4 bits unsigned. The locations with a value of 15 in figure 8 are meant to represent invalid table entries. For other designs these values can be either logically reasoned out, or inferred from simulation data by analyzing the address for values with known bins. A look up table of 32 values can also be used without the selection block logic, but this uses more resources on the FPGA.

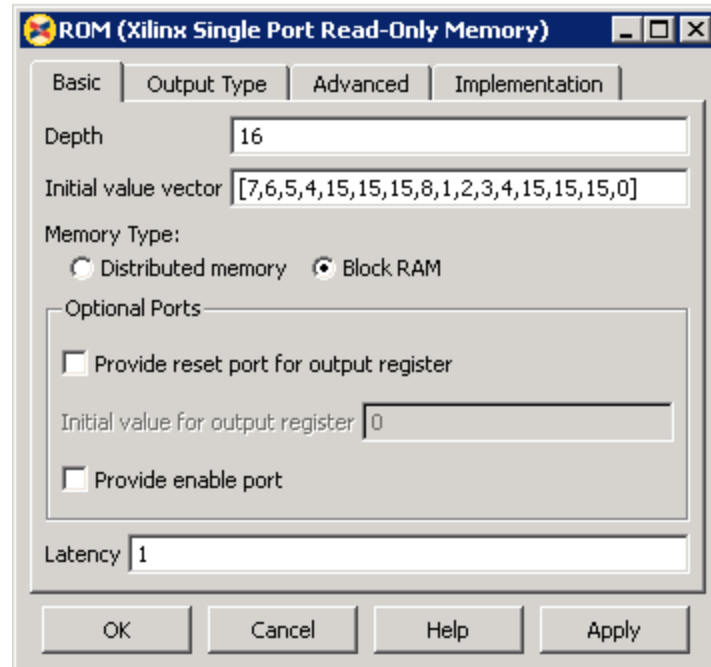


Figure 8

Simulation

To verify that the design works, select start from the simulation menu, and open the scope seen on the right in figure 3. You should see output similar to figure 9.

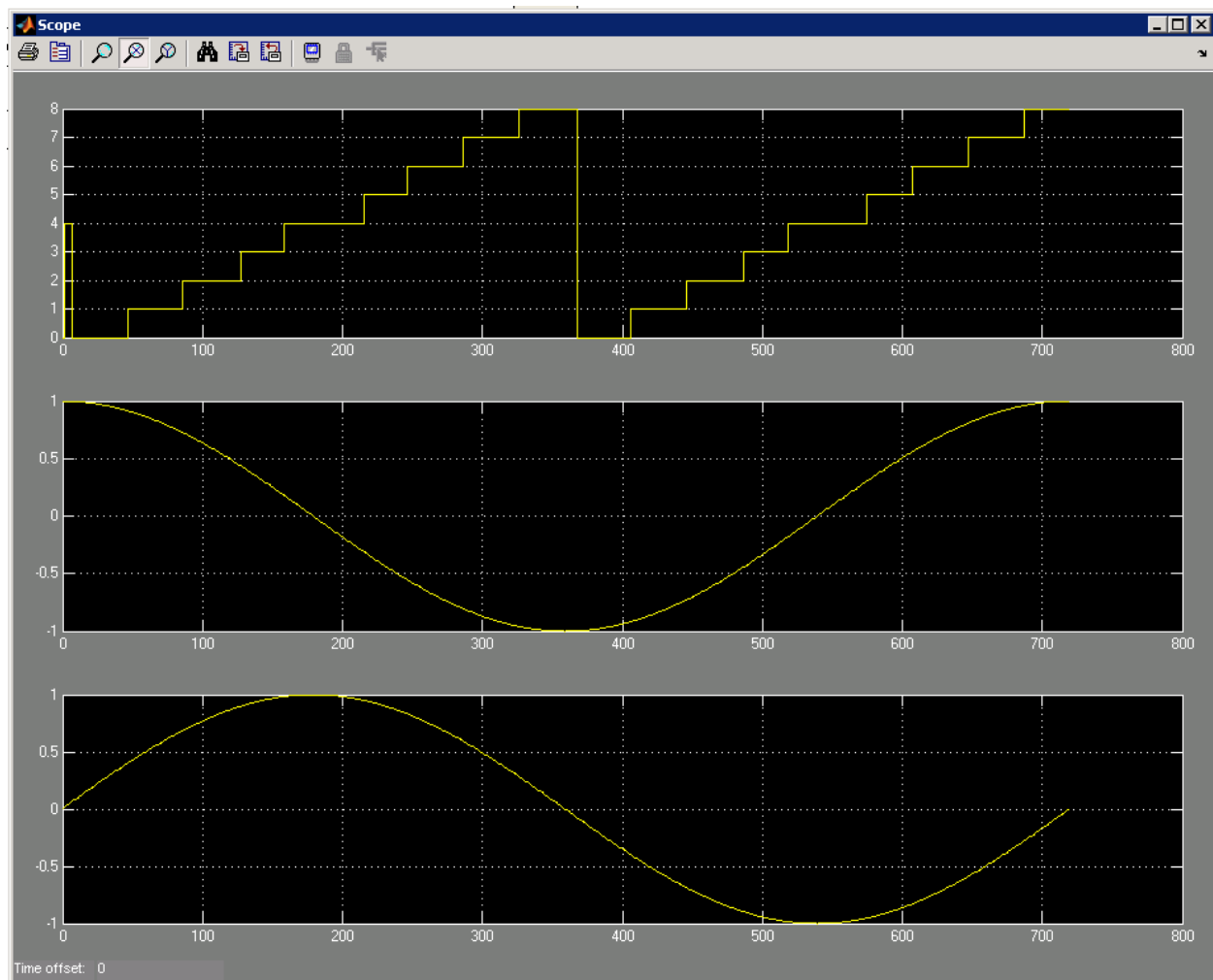


Figure 9

References

FPGA Implementation of a HOG-based Pedestrian Recognition System by Sebastian Bauer

<http://www5.informatik.uni-erlangen.de/Forschung/Publikationen/2009/Bauer09-FIO-talk.pdf>

Matlab/Simulink Help Provided by Install