САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Курсовая работа:

Аппаратное ускорение алгоритмов компьютерного зрения

Выполнил студент 361гр:
Стефан Бояровски

Научный руководитель:
Сергей П. Шувалкин

1 июня 2011 г.

Содержание

1 Введение

1.1 Компьютерное зрение

Как научная дисциплина, компьютерное зрение относится к теории и технологии создания искусственных систем, которые обрабатывают изображения со целью получения некоторой информации. В зависимости от области их применения и от конкретной задачи, существуют разные алгоритмы и подходы к реализации алгоритмов обработки.

Часто встречаемый случай в реализации алгоритмов — многочисленное выполнение одной и той же операции над большим количеством данных. Этот факт говорит о том, что алгоритмы компьютерного зрения являются хорошими кандидатами для оптимизации.

Программу можно оптимизировать на разных уровнях: на теоретическом (изменить алгоритм), либо на уровне аппаратной реализации (применить совместимую аппаратно-программную разработку — САП). САП (Hardware/Software Codesign) — это аппаратная оптимизация, при которой в схеме работы системы, кроме центрального процессора добавляется дополнительная подсистема, в виде программируемой интергральной схемы (ориентированной на выполнение конкретной задачи).

1.2 FPGA (Field-programmable Gate Array)

В настоящее время при ко-дизайне всё чаще используются системы, которые базируются на программируемых пользователем вентильных матриц (ППВМ, FPGA). Большим плюсом этих устройств является то, что их можно сконфигурировать практически в любой момент в процессе их использования. Компания Xilinx является одним из ведущих разработчиков и произоводителей ППВМ. В данной работе используется устройство Xilinx Vitex-6 ML 605.

2 Постановка и цели задачи

2.1 Ознакомление с процессом проектирования FPGA.

2.1.1 Основной подход

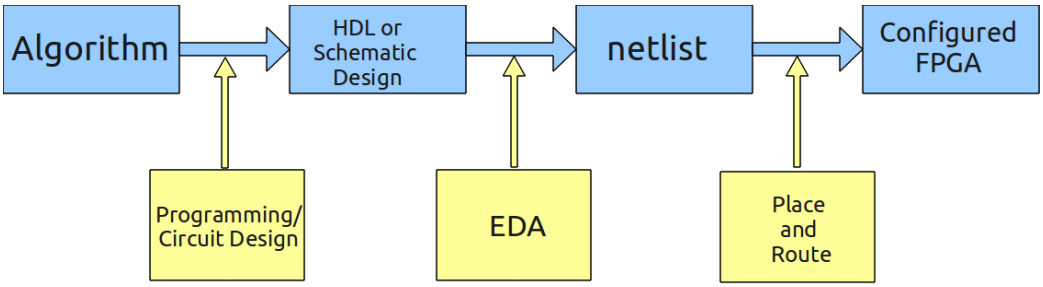

Рис. 1: Основной подход разработки алгоритма при работе с FPGA

Основной подход при проектировании устройств FPGA заключается в следующих последовательных шагах:

- Написание кода алгоритма на языке описания аппаратуры, или рисование электрической схемы, выполняющей работы алгоритма.
- Автоматическое генерирование списка соединений, с помощью некоторого программного средства EDA (Electronic Design Automation, автоматизация проектирования электронных приборов)
- Выполнение процедуры Place and Route, при которой список соединений загружается на устройство. Эту процедуру выполняет программное средство, которое обычно предоставляется пройзводителем устройства.

2.1.2 Альтернативный подход

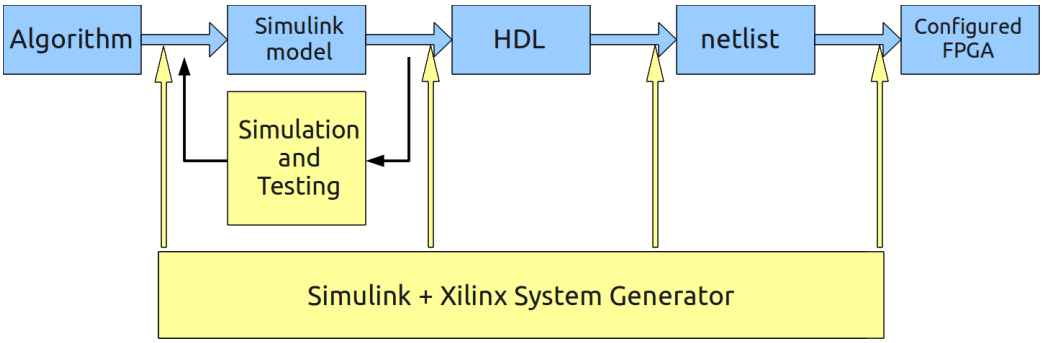

Рис. 2: Альтернативный подход разработки алгоритма при работе с FPGA

В данной работе выбран подход, намного отличающийся от основного. Каждый шаг процесса проектирования выполняется в одной среде разработки, которая представляет собой сочетанием среды визуального моделирования Simulink и Xilinx System Generator.

Simulink — часть пакета прикладных программ MATLAB, являющийся интерактивным инструментом для моделирования, имитации и анализа динамических систем. Широко используется при проектировании модели управления и в цифровой обработке сигналов на основе модельно—ориентированной разработке (MBD).

Xilinx System Generator это комплекс программных средств, который включает в себя библиотеку блоков, с помошью которой можно строить модели реализующие нужного алгоритма. Эти модели можно использовать для симуляции работы алгоритма, тестирования, генерирования кода HDL и список соединений, загрузки списка на устройство, а также и для выполнения аппаратно-программной ко-симуляции.

Вместе с библиотекой Xilnix System Generator, Simulink предоставляет возможность проектировать и выполнять симуляцию модели на компьютере и анализировать её работу, без выполнения длительного процесса генерирования HDL кода, netlist и загрузка на плату.

MBD позволяет работать с алгоритмом на достаточно высоком уровне абстракции, при этом возможность опускаться до более низкого уровня абстракции в самой среде разработки предоставляет более глубокий контроль работы модели.

3 Реализация конкретного алгоритма

3.1 Описание алгоритма фильтра Лапласа

Для наглядного описания процесса построения модели и её проектирование на плату, выбрана задача выделения границ, которая основывается на алгоритмах, которые выделяют точки цифрового изображения, в которых резко изменяется яркость или есть другие виды неоднородностей. Одним из способов реализации такого алгоритма является применение к изображению т.н. фильтр Лапласа, имеющий следующий вид

$$L_{ker} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Значение пикселя вычисляется как сумма поэлементного умножения соседства пикселя на ядро фильтра.

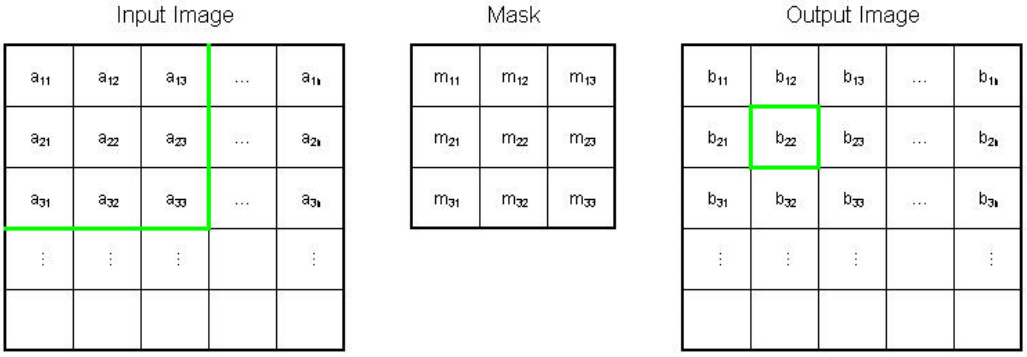

Рис. 3: Вычисление нового значения пикселя

$$b_{22} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} \cdot m_{ij}$$

Для крайных пикселей, «несуществующие» соседы равны нулю.

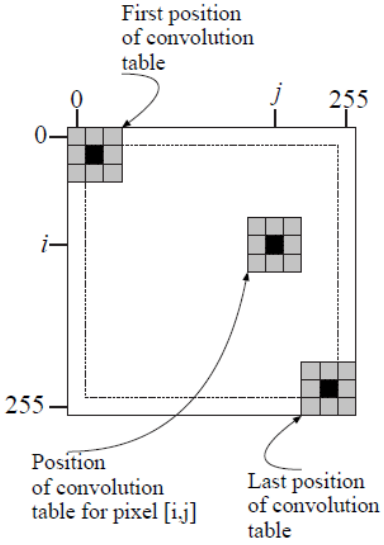

Рис. 4: Обход изображения при вычислении свёртки

3.2 Реализация алгоритма на языке высокого уровня

Написать код для такого алгоритма на языке высокого уровня очень легко.

```
%
%   Laplce 2D filter
%   (c)Stefan Bojarovski, 2011
%

clear;      %Clear the Workspace
clc;        %Clear the Command Window

input = imread('input.jpg');        %Read the input image
[inputHeight, inputWidth, Channel] = size(input);       %Get the image dimensions
L_kernel = [0  1  0 ; ...
            1 -4  1 ; ...
            0  1  0 ];      %This is the kernel of the filter

inputPadded = padarray(input, [1 1], 0);        %Pad the image with 0's on the borders
inputPadded = im2double(inputPadded);       %We must convert it to double in order to do the calculations

windowSize = 3;     %The size of the 'neighbourhood' of the central pixel
output = zeros(inputHeight, inputWidth);        %Pre-declared for optimization

for i = 1 : inputHeight
    for j = 1 : inputWidth
        %Select the neighbouring pixels
        window = inputPadded(i : i + windowSize - 1, ...
                             j : j + windowSize - 1);
        %Calculating the convoltion of the neighbourhood and the filter
        output(i,j) = sum( sum( window().* L_kernel()));
    end
end

imshow(output);
```

Рис. 5: Реализация фильтра Лапласа на языке MATLAB

3.3 Реализация алгоритма в Simulink

Также легко спроектировать модель в Simulink–е с помощю M-code блока:

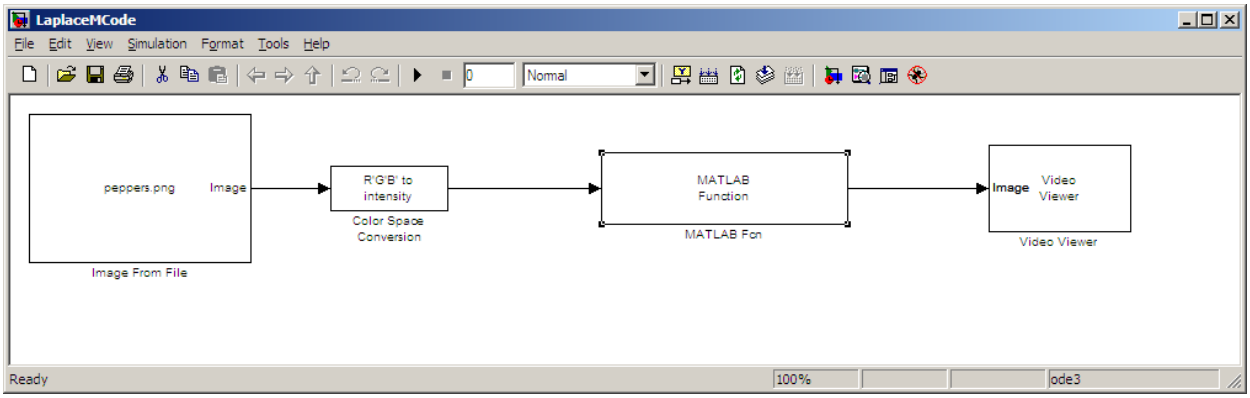

Рис. 6: Модель алгоритма фильтра Лапласа в Simulink

В Simulink–е есть встроенные блоки, которые автоматизируют и облегчают множество полезных функций, такие как чтение с файла, видеоустройства, преобразование цветного изображения в карту интенсивности (черно-белое изображение) и вывод изображения на экране.

Также полезен блок M-Code, который позволяет внедрить любую функцию написанную на коде MATLAB в нашу модель. Число и типы входных параметров функции будет соответствовать числу входных портов блока и форматам ввода. Аналогично, число и типы выходных параметров соответствует числу и форматам выходных портов блока.

К сожалению, FPGA не умеет напрямую работать с такими структурными данными как матрицы. Все элементы умеют работать с одномерными сигналами, т.е. с потоками битов. В библиотеке блоков Xilinx не существуют блоки которые автоматически превращают входные данные в нужный формат. Это значит, что нужно не только найти способ преобразовать входные данные в нужный формат, но и модифицировать алгоритм и создать его реализацию с помощью элементов и инструментов, предоставляемые библиотекой Xilinx.

3.4 Реализация с блоками Xilinx System Generator

Процесс работы с использованием блоков Xilinx (самый верхний уровень модели).

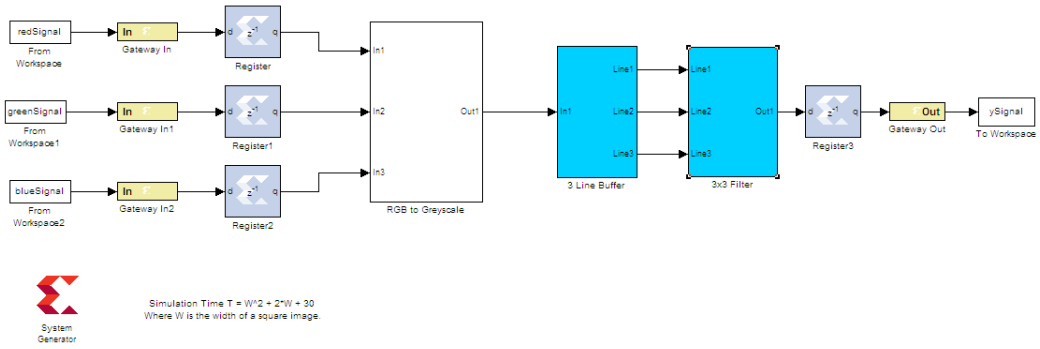

Рис. 7: Модель алгоритма фильтра Лапласа с использованием блоков Xilinx SG

3.5 Преобразование представления входного изображения

Цветное изображение является трёхмерной матрицей размера $Height \times Width \times 3$. Точнее, оно состоит из трёх двумерных матриц, в каждой их которых находятся значения красного, зелёного и синего цвета пикселей соответственно. Для того, чтобы работать с любой $2D$ матрицей, сначала нужно преобразовать её в $1D$ вектор, конкатенируя все строки матрицы в одну.

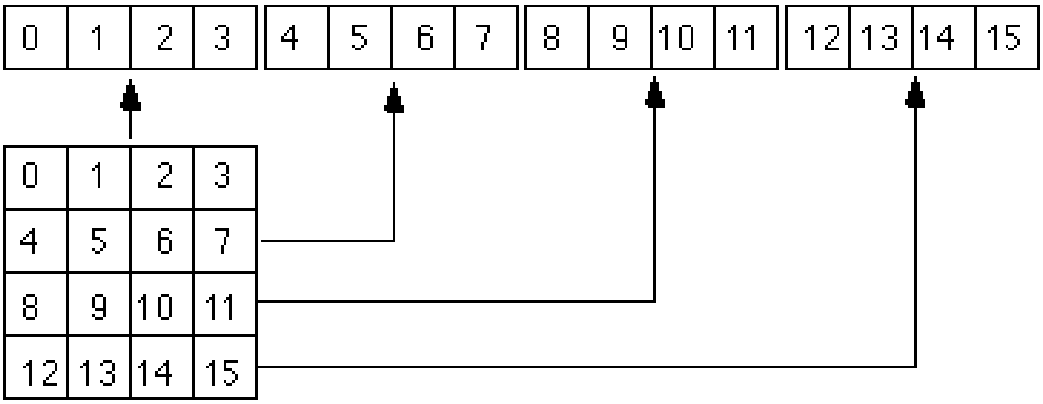

Рис. 8: Преобразование матрицы в строку

После этого, $1D$ вектору нужно «приклеить» еще одну строчку, в которой будет записан порядок поступления каждого элемента в схему.

Рис. 9: Дополнение вектора значений

Все эти операции можно выполнить в самом MATLAB-е, используя встроенные и легко понятные функции для работы с матрицами. Когда имеется такое представление всех трёх каналов, можно начать их обработку.

3.6 Преобразование цветного изображения в карту интенсивности

Для распознавания границ, нам не нужна информация о значений цветов пикселей изображения, а только их интенсивность. Поэтому можно

преобразовать цветное изображение в карту интенсивности. Чтобы получить интенсивность определённого пикселя, вычисляется взвешенная сумма значений трёх каналов (R, G, B), по формуле:

$$Y = 0.3R + 0.6G + 0.1B.$$

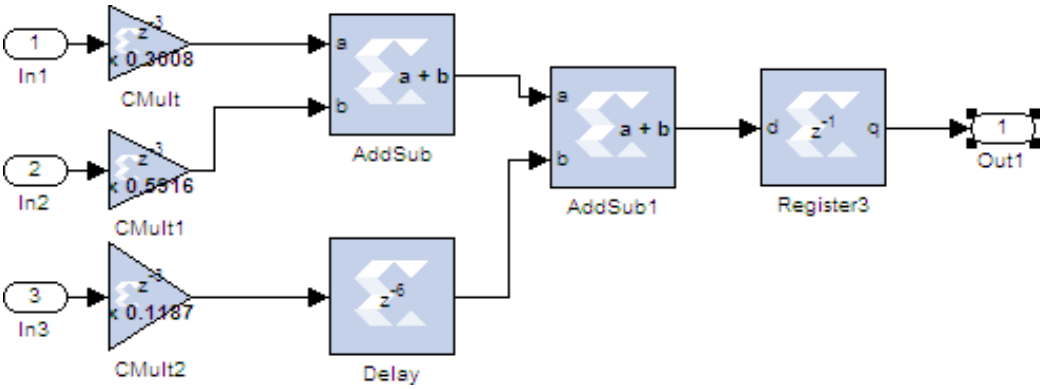

Рис. 10: Схема преобразования цветного изображения в карту интенсивности

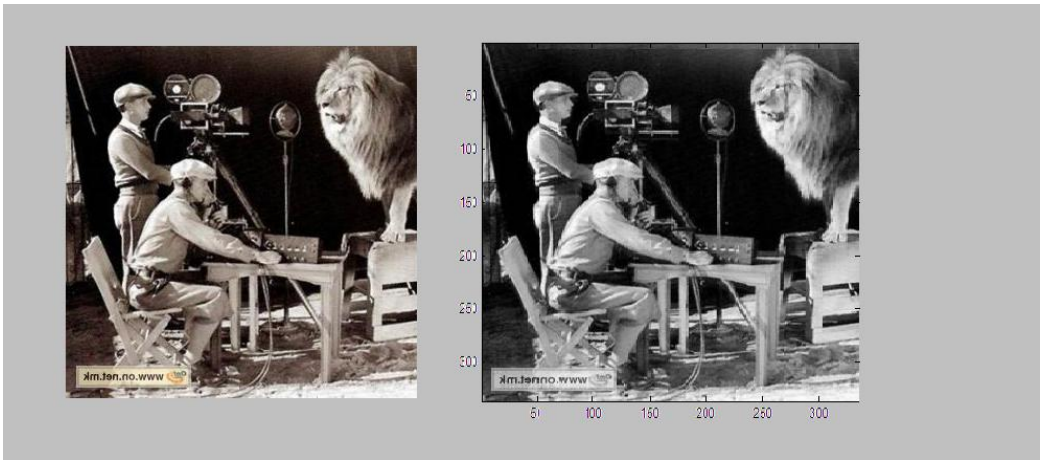

Рис. 11: Результат преобразования

3.7 Построение системы буфферов

На этом шаге уже можно начать обрабатывать изображение фильтром Лапласа. Фильтр двумерный, и ему нужны значения пикселей из двумер-

ного соседства фильтруемого писеля. С другой стороны, значения пикселей изображения поступают только в виде одномерного потока (конкатенация всех строк). Для того, чтобы выполнить работу над одной строкой нам нужны её соседные верхняя и нижняя строка. С помощью блоков Xilinx SG, можно построить буффер содержащий любого количества строк произвольной длины. Нам нужен буффер с тремя строками, длина которых равна ширине изображения. Значения пикселей последовательно входят в систему буфферов (каждый из которых работает по принципу FIFO), и выходят в правильной для вычисления операции фильтрирования форме.

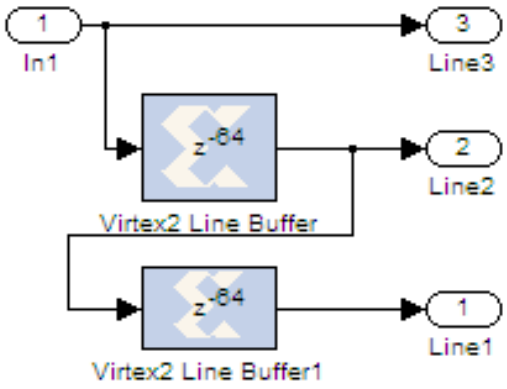

Рис. 12: Подсистема буфферов

Для подсистемой буфферов можно построить «маску», в которой можно задавать в качестве параметра размер каждого буффера.

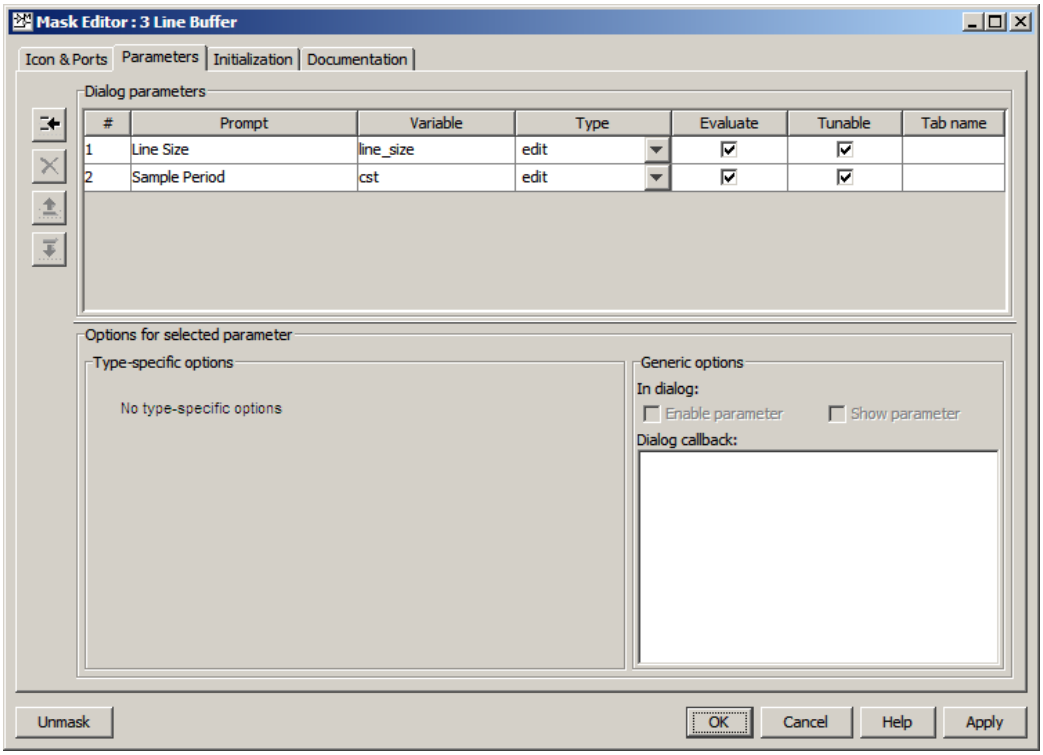

Рис. 13: Редактирование маски подсистемы $3LineBuffer$

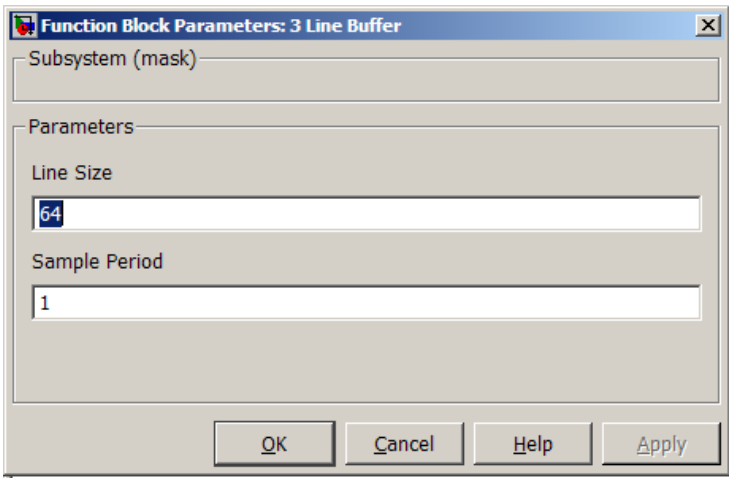

Рис. 14: Введение значения длины буффера

3.8 Система фильтра Лапласа

Фильтр Лапласа можно построить в виде композиции трёх КИХ—фильтров (фильтров с конечной импульсной характеристикой) порядка три. КИХ—фильтр, это линейный фильтр, который обычно реализуется с использованием трёх элементов: умножитель, сумматор и блок задержки.

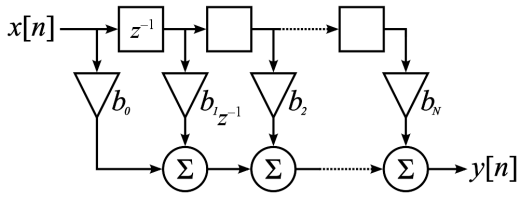

Рис. 15: Общий вид КИХ—фильтра

Уравнение описывающее связь между входными и выходными сигналами КИХ—фильтра имеет вид

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots b_N x[n-N],$$

где
$x[n]$— входной сигнал,
$y[n]$ — выходной сигнал,
N— порядок фильтра,
b_i — коэффициенты фильтра (элементы одной из строчек ядра).

В библиотеке Xilinx SG существует встроенный блок КИХ—фильтра, который будем использовать для построения схему фильтра Лапласа. Он построен следующим образом:

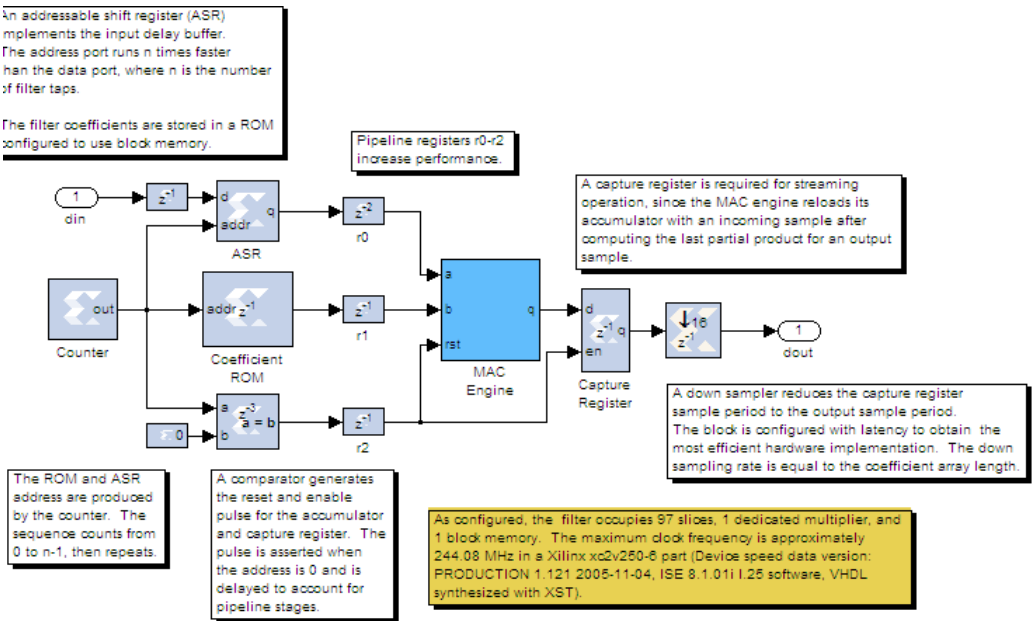

Рис. 16: Реализация КИХ—фильтра в Xilinx SG

В системе $3x3Filter$ используются три КИХ—фильтра соединени следующим образом:

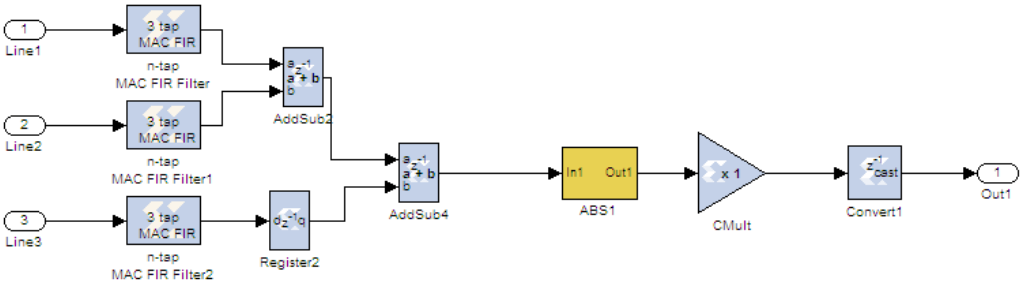

Рис. 17: Схема $3x3Filter$

Эту систему тоже можно замаскировать и включить в маске коэффициенты ядра с которыми будут работать КИХ—фильтры. В маске нужно включить и некоторое количество нужных параметров (точность представления целой части чисел, длина вектора коэффициентов, точность представления дробной части и т.д.), которые управляют с работой элементов подсистемы.

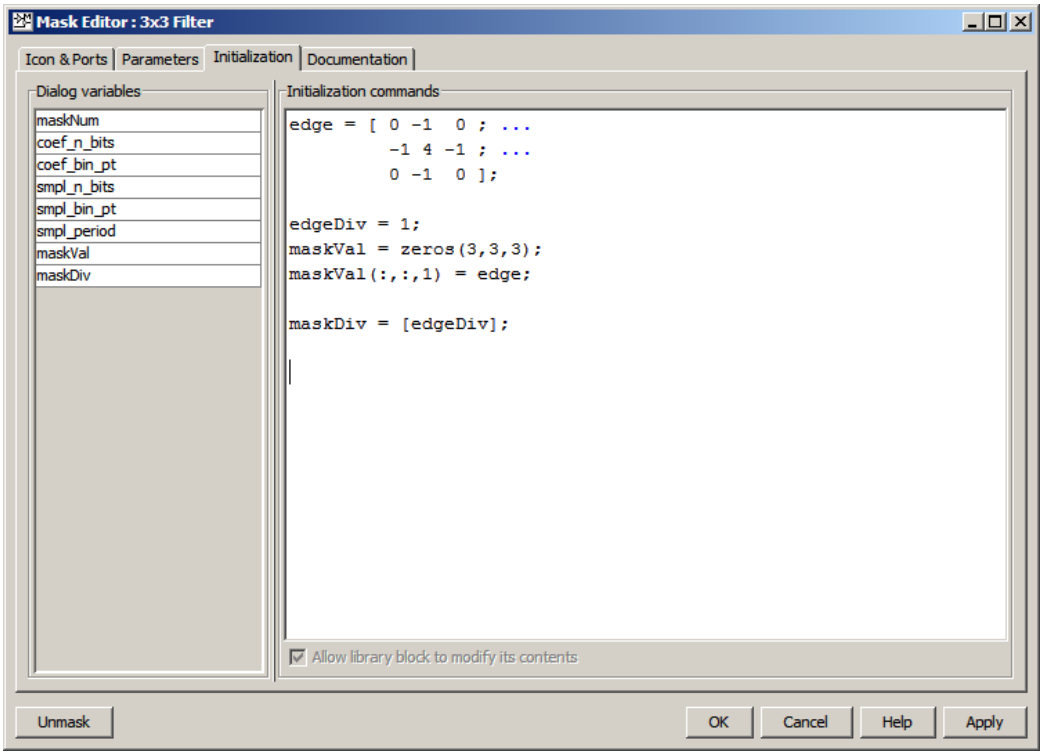

Рис. 18: Редактирование маски подсистемы $3x3Filter$

В дальнейшей работе, можно дополнить эту маску с любими новыми ядрами и выбрать любое из них через удобный интерфейс:

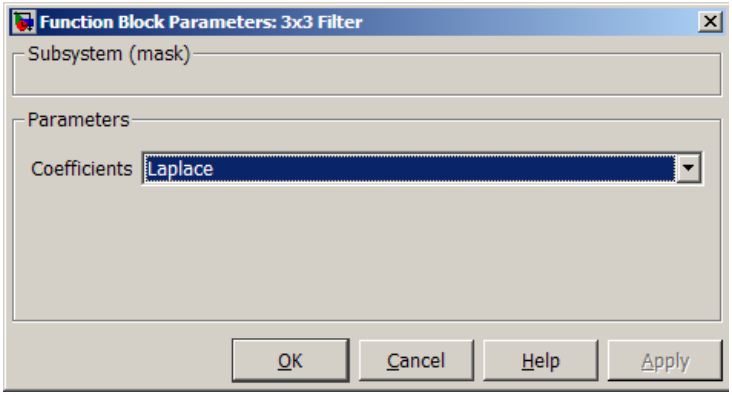

Рис. 19: Маска подсистемы $3x3Filter$

Каждый из фильтров в подсистеме вычисляет значения операции фильтрации для соответствующей строки ядра, а потом эти значения суммируются, чтобы получить значение единого нового пикселя.

Таким образом, последовательно вычисляются значения для всех строк изображения. В результате получаем некоторая последовательность битов в которой находится отфильтрированное изображение, и известное количество излишней информации, которая появилась в процессе заполнения и синхронизации буферов. Эти артефакты можно легко отстранить, так как они являются нулями или NaN (Not a number), и находятся в начале последовательности. Предполагаем, что первое ненулевое и не NaN значение является первым пикселем полученного изображения. Начиная с этого значения мы преобразуем эту последовательность в виде матрицы размера $Height \times Width \times 3$. На этом заканчивается этап реализации и симуляции алгоритма.

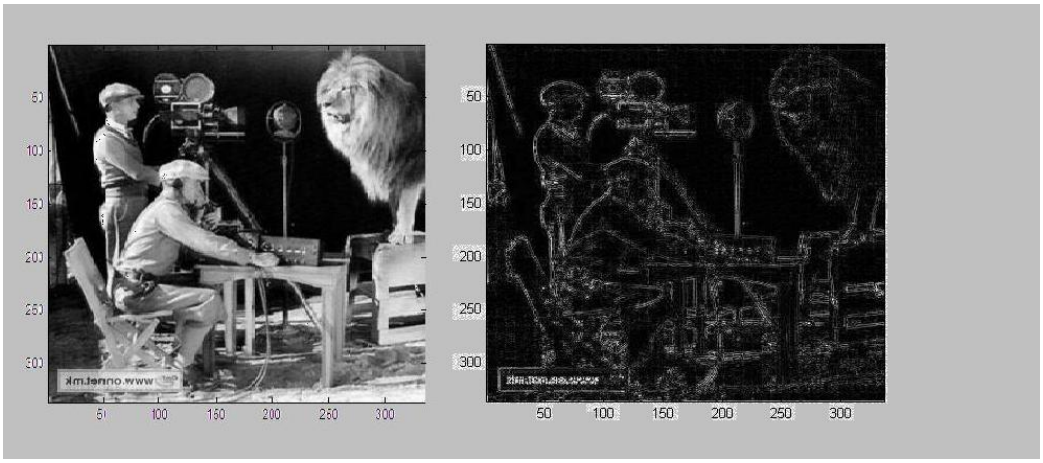

Рис. 20: Результат алгоритма выделения границ с помощью фильтра Лапласа

4 Генерирование списка соединений и совместная программно-аппаратная разработка

После построения модели и её симуляции на компьютере, можно перейти на следующий шаг — генерование HDL кода и списка соединений (netlist).

В данной реализации, загрузка списка соединений и соединение с устройством выполняется через Point-to-Point Ethernet связь. Поэтому в настройках блока System Generator, в качестве цели компиляции выбираем $ML605 \rightarrow Hardware\,Co-Simulation \rightarrow Point-to-Point\,Ethernet$.

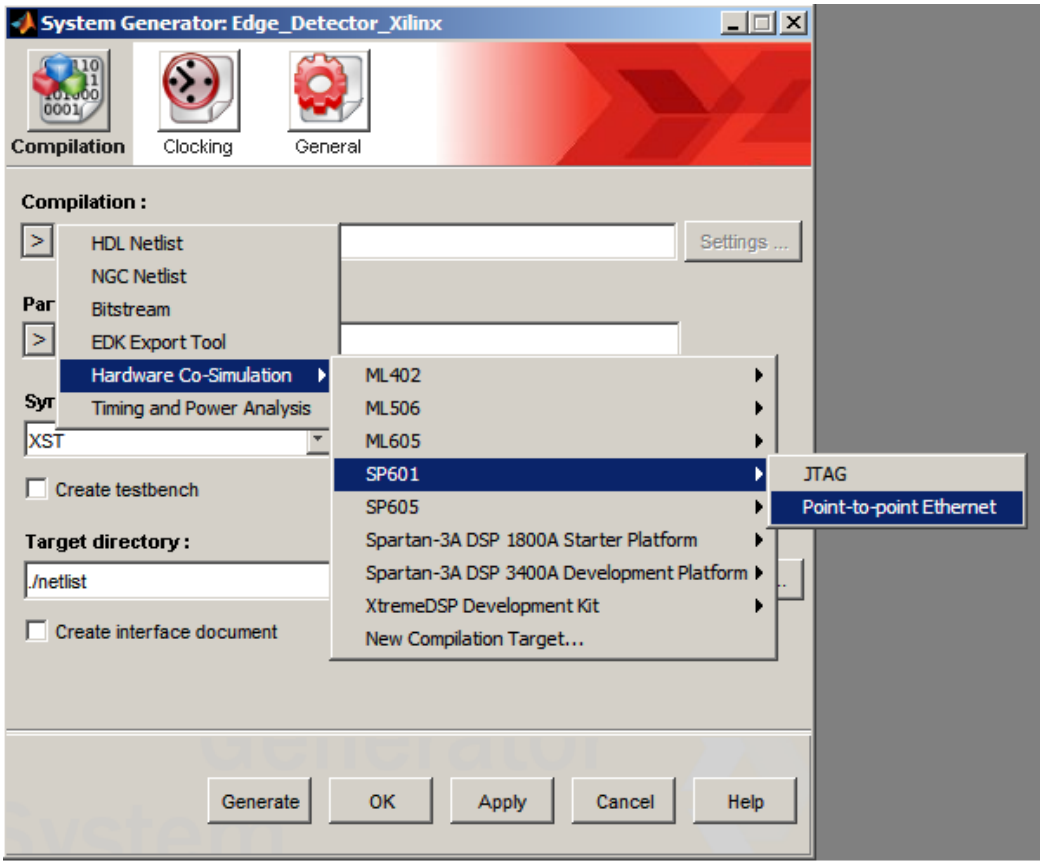

Рис. 21: Выбор типа компиляции

Дальше настраиваем частоту процессора устройства и период модели. Нажатием кнопки *Generate* можно начать процесс автоматического генерирования HDL кода и списка соединений.

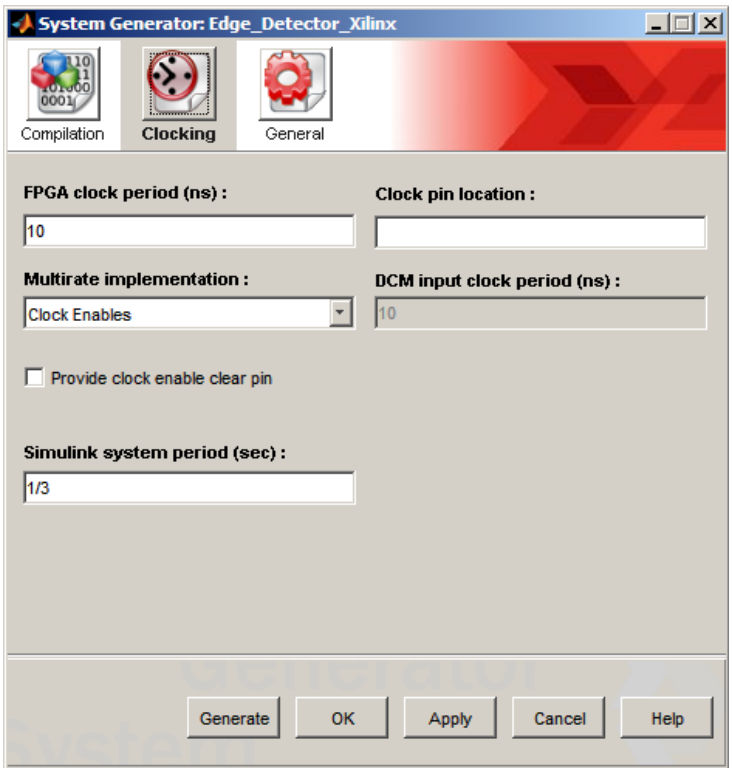

Рис. 22: Настройка частоты процессора

Нужно отметить, что этот процесс, даже для такой простой модели может длиться больше 40 минут. Тем не менее, этот шаг чаще всего выполняется однажды, в конце реализации. Это позволяет сэкономить большое количество времени, так как проверку симуляции и точности работы модели можно выполнять на компьютере после любого изменения.

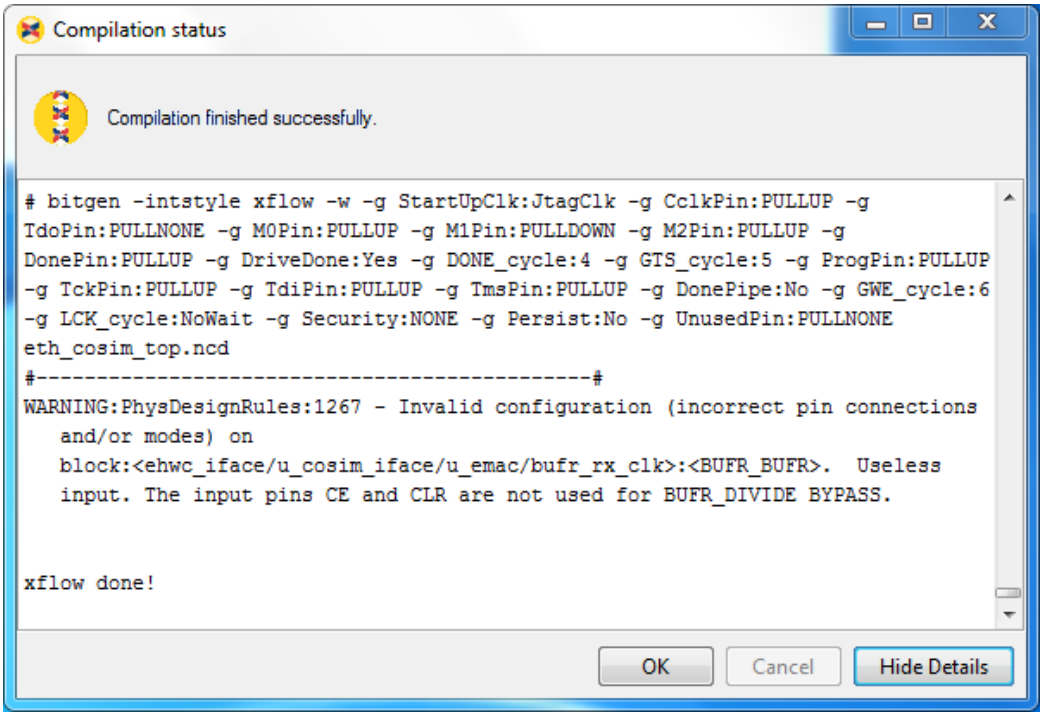

Рис. 23: Генерирование HDL и netlist

В результате генерирования, получется блок, который в своём поведении и форме соответствует данной модели. Разница только в том, что с помощью этого блока, можно автоматически загружать список соединений на устройство и выполнять совместную аппаратно-программную ко-симуляцию.

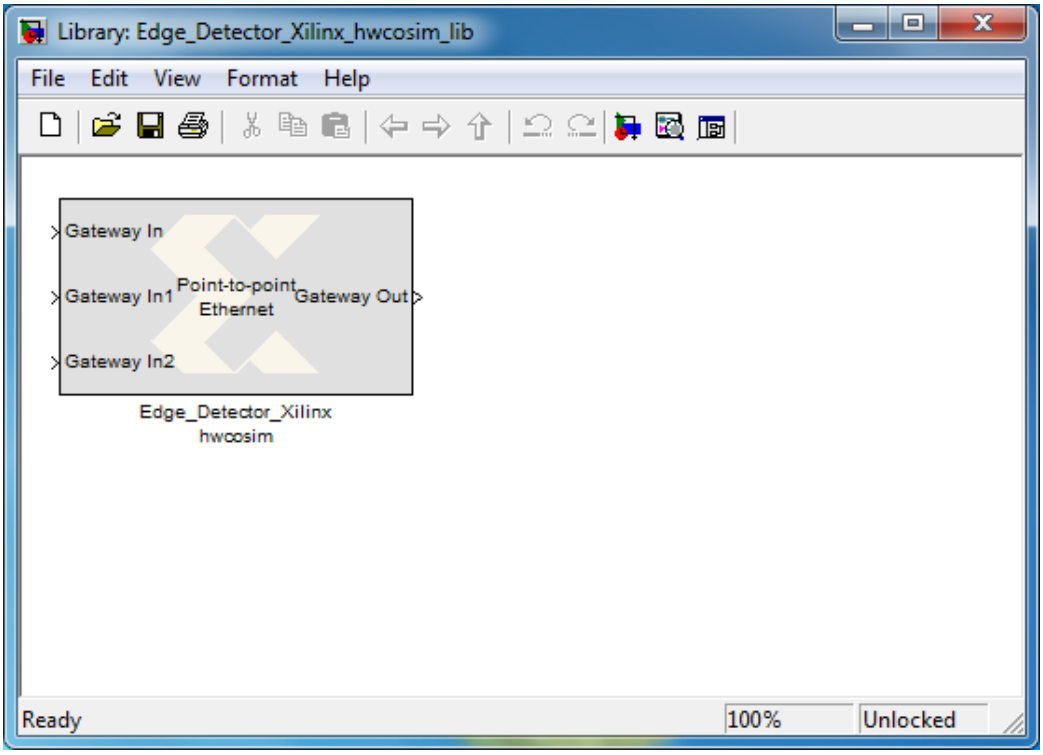

Рис. 24: Блок «чёрная коробка» который выполняет идентичную с системой работу

5 Заключение

В процессе работы были постигнуты следующие результаты:

- изучен процесс проектирования ППВП с помощью Simulink/System Generator
- построена модель симуляции и тестирования для фильтра Лапласа

Процесс аппаратного устроения или оптимизации алгоритмов состоит из следующих этапов:

- ознакомление с процессом САП разработки
- анализ и тестирование разработанных алгоритмов
- оптимизирование

В данной курсовой работе, успешно закончен первый этап, а выполнение последующих этапов предусмотрено в дальнейшей работе.

Список литературы

[1] Сергиенко А. Б. Цифровая обработка сигналов (второе издание). — СПб.: Питер, 2006. - 751 с.: ил.

[2] System Generator for DSP Getting Started Guide

[3] System Generator for DSP Reference Guide

[4] http://www.xilinx.com/