Передовые методики по прототипированию алгоритмов MATLAB и Simulink на ПЛИС.

Стефан ван Бик, Судхир Шарма и Судипа Пракаш, Mathworks

Инженерам-проектировщикам микросхем часто приходится написать до 10 строчек кода для проверки каждой строки RTL кода, реализуемого в кристалле. Они могут потратить больше половины времени на задачи верификации. Несмотря на все эти усилия, почти 60% микросхем содержат функциональные недостатки и требуют повторного выпуска[1]. Так как одной только HDL симуляции недостаточно, чтобы выявить ошибки на системном уровне, в настоящее время разработчики активно используют ПЛИС для ускорения создания алгоритма и прототипа системы.

Использование ПЛИС для обработки больших объемов тестовых данных позволяет инженерам быстро оценить алгоритм и различные компромиссы построения архитектуры системы. Они также могут протестировать проект в реальных условиях, не тратя огромное количество времени на симуляцию HDL. Инструменты для проектирования и верификации на системном уровне, такие как MATLAB® и Simulink®, помогают инженерам реализовать все эти преимущества быстрого прототипирования алгоритмов на ПЛИС.

В данной статье описываются передовые практики по созданию прототипов на ПЛИС с MATLAB и Simulink. Они перечислены ниже и изображены на рисунке 1.

(1) Анализ эффектов квантования, возникающих при использовании арифметики с фиксированной точкой, на ранних стадиях проектирования и оптимизация длины слова для уменьшения занимаемой на кристалле площади и достижения более эффективных с точки зрения энергопотребления реализаций.

(2) Ускорение создания прототипов на ПЛИС благодаря использованию автоматической генерации HDL кода

(3) Повторное использование тестов в режиме ко-симуляции для анализа HDL реализации на системном уровне

(4) Ускорение верификации с технологией симуляции ПЛИС в контуре (FPGA-in-the-loop)

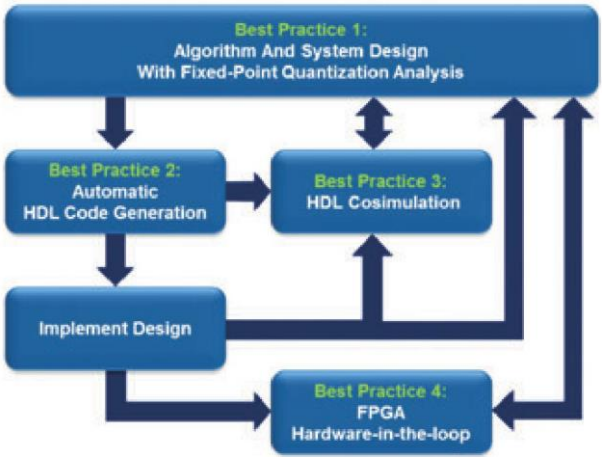

Рис. 1. Лучшие практики Модельно-Ориентированного Проектирования для создания прототипов на ПЛИС.

ПОЧЕМУ НУЖНО ПРОТОТИПИРОВАТЬ НА ПЛИС?

Прототипирование алгоритмов на ПЛИС дает инженерам больше уверенности в том, что поведение алгоритма в реальных условиях будет соответствовать ожиданиям. В дополнение к запуску тестовых векторов на высокой скорости разработчики могут использовать эти прототипы для проверки программного обеспечения и смежных функций системного уровня, таких как РЧ и аналоговые подсистемы. Кроме того, так как прототипы на ПЛИС работают быстрее, можно использовать гораздо большие массивы данных и находить ошибки, которые не могут быть обнаружены на имитационной модели.

Модельно-Ориентированное Проектирование (МОП) с использованием генерации HDL кода позволяет инженерам эффективно создавать прототипы на ПЛИС (Рис. 2). Этот рисунок иллюстрирует, что на практике инженеры часто сокращают стадию детальной проработки проекта в попытке поскорее начать разработку аппаратных средств, чтобы не выбиться из графика. Однако им все равно придется вернуться к этому этапу, когда обнаружится, что алгоритм в арифметике с фиксированной точкой не удовлетворяет системным требованиям. Такой откат способствует удлинению фазы создания HDL кода (показана длинной фиолетовой полосой) и может привести к таким последствиям, как использование связующих логических схем (glue logic) и дополнительных патчей к проекту.

Так как автоматическая генерация HDL кода быстрее, чем ручное кодирование, инженеры могут потратить часть сэкономленного времени на разработку более качественного алгоритма в арифметике с фиксированной точкой на этапе детализации проекта. Данный подход способствует более быстрому и качественному выпуску прототипов на ПЛИС, чем процесс ручного кодирования.

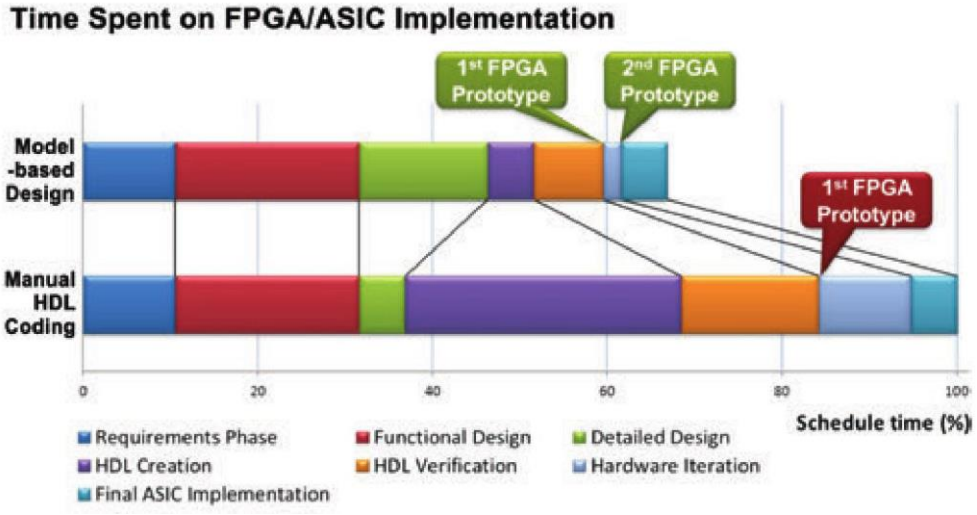

Рис. 2. Сравнение МОП и процесса ручного кодирования

ПРИМЕР ИЗ ПРАКТИКИ: ЦИФРОВОЙ ПОНИЖАЮЩИЙ ПРЕОБРАЗОВАТЕЛЬ.

Цифровой преобразователь частоты (digital down converter – DDC) служит полезным примером, иллюстрирующим передовой опыт использования МОП для создания прототипов на ПЛИС.

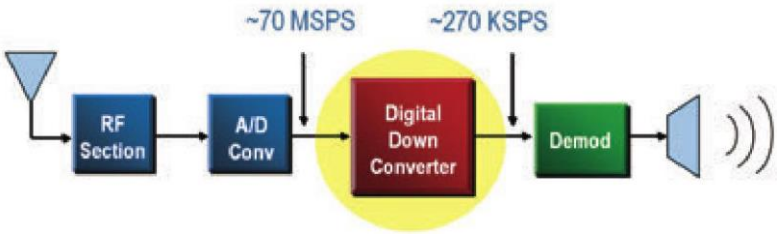

Рис. 3. Система связи с цифровым понижающим преобразователем.

Это устройство применяется во многих системах связи (см. рис. 3) для преобразования высокочастотного полосового сигнала в низкочастотный видеосигнал, который можно обрабатывать на низких частотах дискретизации. Это приводит к снижению энергопотребления и вычислительных затрат оборудования.

Основные компоненты DDC показаны на рисунке 4:

- Осциллятор с ЧПУ (NCO);
- Смеситель (Mixer);
- Цифровой фильтр.

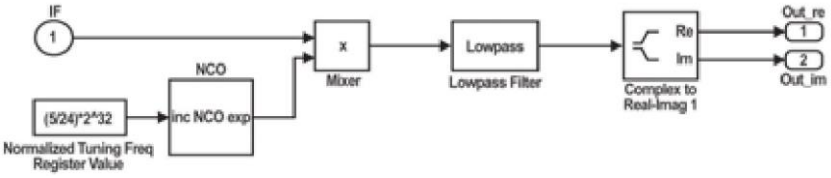

Рис. 4. Модель цифрового понижающего преобразователя частоты.

РЕКОМЕНДАЦИЯ №1 – АНАЛИЗИРУЙТЕ ЭФФЕКТЫ КВАНТОВАНИЯ НА РАННЕМ ЭТАПЕ ПРОЕКТИРОВАНИЯ

Инженеры, как правило, проверяют новые идеи и разрабатывают первоначальные алгоритмы, используя арифметику с плавающей точкой. Однако аппаратная реализация на ПЛИС и заказных микросхемах требует перехода к типам данных с фиксированной точкой, которая часто вносит ошибки квантования.

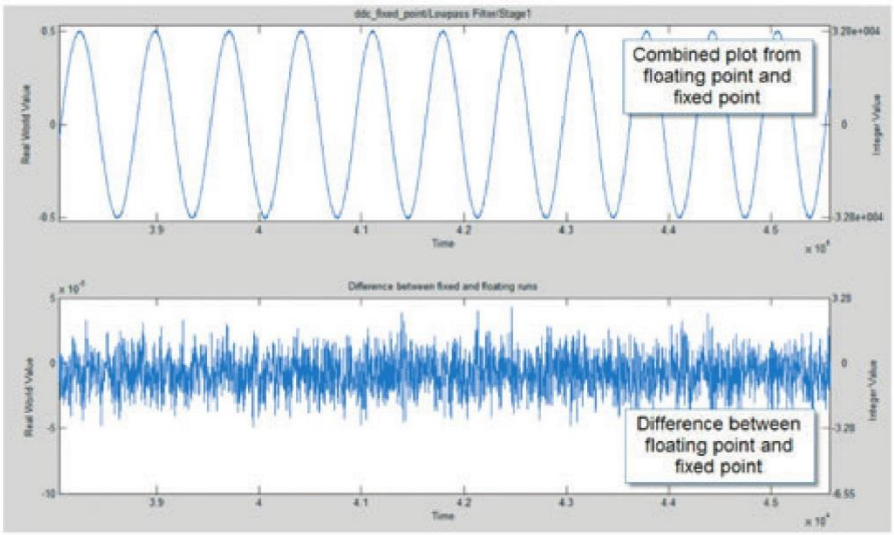

Рис. 5. Количественная оценка эффекта квантования с помощью Fixed-Point Designer.

При стандартном рабочем процессе переход к арифметике с фиксированной точкой обычно выполняется во время написания HDL кода. При этом у инженеров нет возможности оценить эффекты квантования, сравнив представление в фиксированной точке с эталонной моделью в плавающей точке. Также непросто проанализировать и HDL реализацию на переполнения.

Чтобы принять разумное решение о требуемой длине дробной части, инженерам нужно уметь сравнивать результаты симуляции в плавающей и фиксированной точке до начала кодирования на HDL. Увеличение длины дробной части уменьшает ошибки квантования, но приводит к увеличению длины слова и, как следствие, к большей площади на кристалле и потребляемой мощности.

Например, рисунок 5 показывает различия между результатами симуляции первого этапа низкочастотной фильтрации из цепи фильтров, входящих в состав DDC, в плавающей и фиксированной точке. На верхнем графике выводятся выходы из обеих реализаций, а на нижнем ошибка квантования для каждого отсчета. В зависимости от требований к проекту, инженерам может понадобиться увеличить длину дробной части, чтобы уменьшить значения ошибок.

Кроме выбора длины дробной части, инженеры должны оптимизировать длину слова для достижения низкого энергопотребления и эффективного использования логики кристалла.

В данном примере с помощью инструмента Fixed-Point Designer инженерам удалось уменьшить длину слова в некоторых элементах фильтра на целых 8 бит (см. рис. 6).

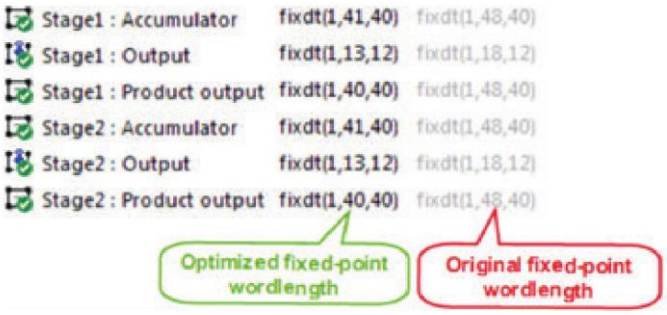

Рис. 6. Оптимизация применяемой арифметики с фиксированной точкой с помощью Fixed-Point Designer.

РЕКОМЕНДАЦИЯ №2 – ИСПОЛЬЗУЙТЕ АВТОМАТИЧЕСКУЮ ГЕНЕРАЦИЮ HDL КОДА, ЧТОБЫ БЫСТРЕЕ ВЫПУСКАТЬ ПРОТОТИПЫ НА ПЛИС.

Для создания прототипа на ПЛИС необходим HDL код. Инженеры пишут Verilog и VHDL код вручную. В качестве альтернативы этому выступает автоматическая генерация кода с помощью HDL Coder, которая дает важные преимущества. Инженеры могут:

- Быстро оценить может ли алгоритм быть реализован на оборудовании;
- Моментально проверить различные реализации алгоритма и выбрать лучшую из них;
- Быстрее прототипировать алгоритмы на ПЛИС.

```
BEGIN
  -- Count limited, Unsigned Counter
  --  initial value   = 0
  --  step value      = 1
  --  count to value  = 119
  --
  -- <S22>/Counter Limited
  Counter_Limited_process : PROCESS (clk)
  BEGIN
    IF clk'EVENT AND clk = '1' THEN
      IF reset = '1' THEN
        Counter_Limited_count <= to_unsigned(0, 8);
      ELSIF enb = '1' THEN
        IF Counter_Limited_count = 119 THEN
          Counter_Limited_count <= to_unsigned(0, 8);
        ELSE
          Counter_Limited_count <= Counter_Limited_count + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS Counter_Limited_process;

  Counter_Limited_out1 <= Counter_Limited_count;

  -- <S22>/1-D Lookup Table

  alpha1_D_Lookup_Table_k <= to_signed(0, 31) WHEN Counter_Limited_out1 <= 0 ELSE
      to_signed(119, 31) WHEN Counter_Limited_out1 >= 119 ELSE
      signed(resize(Counter_Limited_out1, 31));
  alpha1_D_Lookup_Table_out1_re <= table_data_re(to_integer(alpha1_D_Lookup_Table_k));
  alpha1_D_Lookup_Table_out1_im <= table_data_im(to_integer(alpha1_D_Lookup_Table_k));
```

Рис. 7. Пример HDL кода, полученного из модели Simulink.

При создании DDC мы сгенерировали 5780 строк HDL кода за 55 секунд. Инженеры могут прочесть и понять этот код (рис. 7)

Автоматическая генерация кода позволяет вносить изменения в модель системного уровня и тут же получать обновленную HDL реализацию, повторно сгенерировав код.

РЕКОМЕНДАЦИЯ №3 – ИСПОЛЬЗУЙТЕ ОДНУ И ТУ ЖЕ ИСПЫТАТЕЛЬНУЮ СРЕДУ В РЕЖИМЕ КО-СИМУЛЯЦИЯ ДЛЯ ВЕРИФИКАЦИИ HDL КОДА.

Функциональная верификация.

Ко-симуляция HDL позволяет инженерам повторно использовать модели Simulink для подачи сигналов на вход HDL симулятора и интерактивно выполнять анализ выходных сигналов на системном уровне (Рис. 8).

В то время как симуляция HDL обеспечивает только выход цифровых сигналов, режим ко-симуляции вооружает разработчика всем арсеналом инструментов системного анализа из Simulink, а также дает взгляд на всю систему в целом. Когда инженеры наблюдают различия между ожидаемыми результатами и симуляцией HDL кода, ко-симуляция помогает лучше понять влияние данных несоответствий на все изделие.

К примеру, отображение спектра сигнала (рис. 9) позволяет инженерам принять обоснованное решение об игнорировании несоответствия между исходным проектом и HDL реализацией, потому что эти расхождения лежат ниже полосы подавления. В то время как одни только флаги о расхождениях в результатах моделирования укажут лишь на наличие ошибки, а не на результаты этой ошибки. В конечном итоге инженер

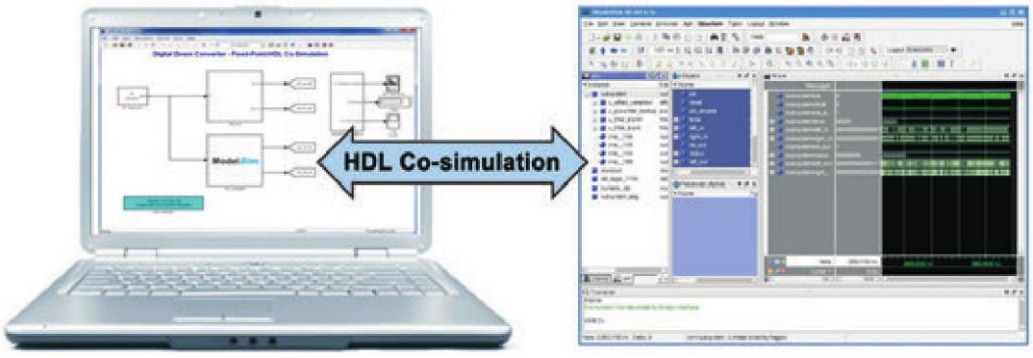

Рис. 8.

Ко-симуляция HDL кода между Questa/ModelSim и Simulink

может прийти к тому же решению, но подобный анализ займет у него гораздо больше времени.

Покрытие модели тестами.

Можно использовать HDL Verifier, Simulink design Verifier и Questa/ModelSim для автоматизации анализа покрытия кода. При таком подходе Simulink Design Verifier создает наборы тестовых воздействий, а HDL Verifier запускает Questa/ModelSim для сбора данных о покрытии кода тестами для полноценного анализа сгенерированного кода.

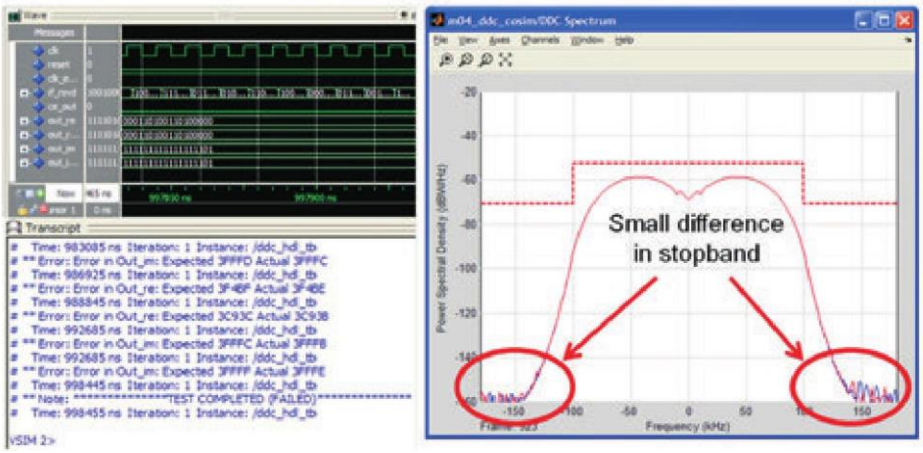

Рис. 9. Анализ показателей системы с помощью спектроскопа и доступ к поведению HDL реализации.

РЕКОМЕНДАЦИЯ №4 – УСКОРЯЙТЕ ВЕРИФИКАЦИЮ ЗА СЧЕТ СИМУЛЯЦИИ В РЕЖИМЕ FPGA-IN-THE-LOOP.

Имея алгоритм понижающего преобразователя, верифицированный на системном уровне и с помощью ко-симуляции HDL, теперь можно развернуть его на целевой платформе ПЛИС.

Верификация алгоритма с помощью ПЛИС, также называемая ПЛИС в контуре (FPGA-in-the-loop) дает больше уверенности, что алгоритм будет верно работать в реальных условиях. Она позволяет инженерам выполнять тестовые сценарии быстрее, чем ко-симуляция на компьютере.

Модель Simulink при этом используется для управления входами ПЛИС и сбора результатов ее работы (рис. 10). Как и в режиме ко-симуляции, результаты всегда доступны для анализа в Simulink.

В таблице 1 сравниваются два метода верификации (ко-симуляция HDL и технология FPGA-in-the-loop), используемые при разработке цифрового понижающего преобразователя. В этом случае симуляция FPGA-in-the-loop была в 23 раза быстрее. Такое увеличение скорости позволяет инженерам при испытаниях прогонять большие объемы данных и выполнять регрессионное тестирование проектов. Это позволяет выявлять потенциальные проблемы, требующие дополнительной проработки.

В то же время, несмотря на скорость работы, ко-симуляция дает большую видимость внутри HDL кода. Поэтому она больше подходит для детального анализа проблемных мест, найденных во время симуляции FPGA-in-the-loop.

Таблица 1. Сравнение возможностей верификации с помощью ко-симуляции HDL и режима FPGA-in-the-loop

Метод верификации	Скорость симуляции	Видимость в HDL коде	Возможность системного анализа
HDL Ко-симуляция	46 сек	Да	Да
FPGA-in-the-loop	2 сек	Нет	Да

Рис. 10. Симуляция в режиме FPGA-in-the-loop с помощью Simulink и платы с ПЛИС.

ИТОГИ.

Описанные в данной статье четыре лучших практики позволяют инженерам разрабатывать прототипы на ПЛИС гораздо быстрее и качественнее, чем при традиционном рукописном рабочем процессе. Кроме того инженеры могут продолжать совершенствовать свои модели на протяжении всей разработки и моментально получать код для реализации на ПЛИС. Эта возможность способствует существенному сокращению времени на итерации над проектом.

Чтобы получить больше информации о данном рабочем процессе, посетите вебсайт:

www.mathworks.com/programs/techkits/techkit_asic_response.html

ПРИМЕЧАНИЕ

[1] Аппаратная/программная ко-верификация, Dr.Jack Horgan.