Qucs

Work Book

Thierry Scordilis
Mike Brinson
Gunther Kraut
Stefan Jahn
Chris Pitcher

Оглавление

Введение

Важные замечания и предупреждения

Вы должны принять во внимание тот факт, что книга написана «на ходу», так что возможны некоторые ошибки, и авторы не несут ответственности за любые последствия использования книги.

Данный документ предназначен стать рабочей книгой для разработчиков в области радио и микроволновой техники. В наши намерения не входит создать курс по радиотехнике, но лишь прикоснуться к ее основам. Задача, поставленная перед нами, наметить некоторые правила разработки и последовательность работы для тех, кто использует RF CAD программы. Эта последовательность работы будет поддерживаться в разных главах на достаточно разных предметах.

Содержание рабочей книги

В этой рабочей книге мы пройдемся по некоторым обычным задачам. Но с некоторым прогрессом в объяснениях, и, благодаря тому факту, что мы охватим обширное поле информации, некоторые ключевые моменты будут показаны только единожды, так что рекомендуем вернуться к чтению этих глав.

Эта рабочая книга включает:

последовательность работы: показан типовой процесс разработки проекта,

осмысление RF спецификации: обычная задача, которая может стать проклятой, может превратить проект в ночной кошмар,

BJT моделирование: после выбора устройства, мы всегда нуждаемся в использовании его в CAD, и обычно как раз этого устройства нет в CAD ... как создать его и проверить,

DC статика: поскольку все активные устройства должны смещаться ...

PA разработка: активный компонент найден, и маленький усилитель разработан без многочисленных ограничений,

LNA разработка: разработка более ограниченная большим количеством правил, стабильностью, шумами и т. д. ...

разработка осциллятора: процедура обычно из CAD вытекающая, поддержанная не обычной процедурой,

vco разработка: нормальная эволюция из осциллятора,

detector: разработка трудная в поддержке,

последует и больше . . .

1 Основная последовательность разработки

Подразумевая, что вы знакомы с обычной последовательностью разработки RF, HF, микроволновых схем или систем, остается только пояснить, как можно использовать QUCS для разработки такого типа цепей.

Как инженер-исследователь RF я еще и работаю со студентами. И всегда есть проблемы с новыми методами преподавания. Обычно студенты приходят с некоторыми познаниями в области CAD программ, но они не знают в действительности, как соразмерять свою разработку. Они используют только оптимизатор для размещения своих идей. Что обидно! Конечно, не все из них таковы, но это общая тенденция. На протяжении книги я хочу показать, что есть некоторые правила разработки, и разработка может просчитываться, и что она не будет работать по волшебству!!!

Для экспертов в этом нет ничего нового, но только некоторые постоянно используют QUCS. Хотя, собственно, правила разработки те же, что и при использовании на рабочем месте бумаги и ручки.

Автор.

Правильная организация документа

Постараемся сохранить единую организацию внутри разных глав, что будет, к слову:

Заглавная часть: которая осветит активное поле разработки, предполагаемое к использованию.
Блок спецификации: в порядке понимания, что мы должны сделать. Эта задача не будет объяснена на первый взгляд, поскольку не это цель данного документа (мы не знакомим со спецификацией системы. Это может появиться в том случае, когда компоненты, присутствующие в QUCS, будут расширены... но почему бы и нет в дальнейшем).

DC объяснение: если разработка включает DC часть, тогда мы должны предоставить DC демонстрацию, включая тепловой аспект, если нужно.
Функциональная разработка: в порядке пояснения, как это функционально разработано, либо в общем, либо в отношении QUCS. Второй аспект всегда должно держать в памяти. Все может не относится к другим CAD программам, и

следовательно остаться без внимания.

Надеюсь, что вышеизложенное поясняет цель создания этого документа.

2 Понимание RF спецификации

Параметры

. . . подготовлено Norman E. Dye из Motorola RF подразделения: AN 1107[1]. Так как это AN существенно для наших тем, не плохо бы дать небольшую ссылку на нее. Все AN от motorola ссылаются на это поле. Эта глава только выдержка, но главная цель выделена в дальнейшем . . .

Автор.

2.1 Введение

Справочные таблицы часто оказываются единственным источником информации о возможностях и характеристиках продукта. Это практически верно для уникальных RF полупроводниковых устройств, которые используются разработчиками оборудования во всем мире. Поскольку разработчик схемы подчас не может поговорить непосредственно с производителем, он полагается на информацию справочных данных его устройства. А для RF устройств многие спецификации уникальны сами по себе. Таким образом, важно, чтобы пользователь и производитель RF продукции говорили на одном языке, с тем чтобы то, что изготовитель полупроводников говорит о своем RF устройстве, было полностью понятно разработчику схемы.

В этой части рассматриваются параметры RF транзисторов и усилительных модулей из соображений максимальной оценки функциональных характеристик. Они разделены на пять базовых секций:

1. DC спецификации
2. мощные транзисторы
3. транзисторы малой мощности
4. мощные модули
5. линейные модули.

Комментарии сделаны для критических спецификаций – как определяются значения и в чем их смысл.

1 Это примечание может быть найдено в старых примечаниях справочных проспектов от motorola, если вы сохранили один из них, это истинное сокровище.

2.2 DC спецификация

В основном RF транзисторы характеризуются двумя типами параметров: DC и функциональными. «DC» спецификация состоит из предельного напряжения, тока утечки, hFE (DC β) и емкостей, тогда как функциональная спецификация покрывает усиление, наработку, шумовые параметры, искажения и т. д. . . . Тепловые характеристики, не попадающие в категорию теплового сопротивления и мощности рассеяния, могут быть либо DC, либо AC. Таким образом, мы будем обходиться со спецификацией теплового сопротивления, как с особой спецификацией, и дадим ей собственный заголовок, названный «тепловые характеристики».

3 DC анализ, развертка параметра и модели устройств

3.1 DC статические цепи

Любимый вопрос на курсах по электронике это:

«У вас двенадцать резисторов по одному ому. Вы соединяете их вместе так, что каждый резистор образует кромку куба. Каково сопротивление между противоположными углами куба?»

Намерение может быть в обучении паянию, чтобы более чем один студент спаял именно такой куб! Сегодня мы можем сделать это не прибегая к обучению пайке, мы симулируем цепь.

Вот моя попытка сделать куб в Qucs; любой желающий может попытаться улучшить его.

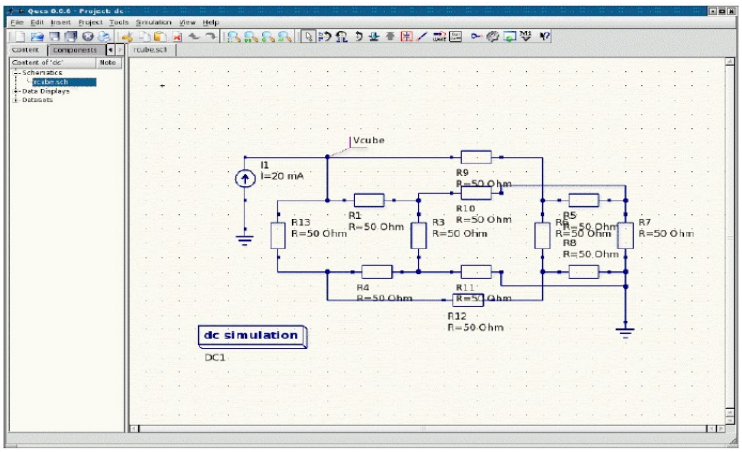


Рис. 3.1. Схема куба из резисторов

Все, что мне пришлось делать, это выбрать сопротивления слева в окне компонент и вставить их, повернув при необходимости, пока я ни получил двенадцать в схеме. Затем я соединил две сети по четыре в квадрат, а затем соединил оставшиеся четыре по углам квадратов. Что, я уверен, топологически и есть куб.

Все это может показаться тривиальным, но хорошо бы напомнить для начала, что мы создали виртуальное представление физической цепи. Временами мы должны крутить

и сжимать все, чтобы уложить в формат, который примет симулятор, и что вызывает сомнения, работаем ли мы с достаточно аккуратным представлением?
Есть правило: если мы можем привести в соответствие соединение наших компонент с компонентами реальной схемы, мы аккуратно представляем физическую цепь. И, я преувеличиваю, ВСЕГДА лучше проверить, что мы все сделали правильно – симулируйте неправильную цепь, и она расскажет вам, где вы солгали.

С моим кубом из резисторов, аккуратно нарисованным, я должен только нажать клавишу симуляции, и сведенные в таблицу результаты покажут мне напряжение в угловых узлах. Поскольку я пропускаю постоянный ток через куб, от одного угла к другому, закон Ома говорит мне, что напряжение между этими углами даст мне сопротивление. Если я использую ток в один ампер, выходной вольтаж будет эквивалентен резистору в ом[2].

Те, что внимательны к деталям, могут возразить сейчас, что я вовсе не решил действительной проблемы, поскольку в задаче были резисторы по одному ому, тогда как я использовал резисторы по пятьдесят ом. Хорошо, да так, я сжульничал. Что я часто делаю в симуляциях.

Для приведения всех резисторов к правильному значению, я должен был открыть окно редактора свойств двенадцать раз; вот как оно выглядит...

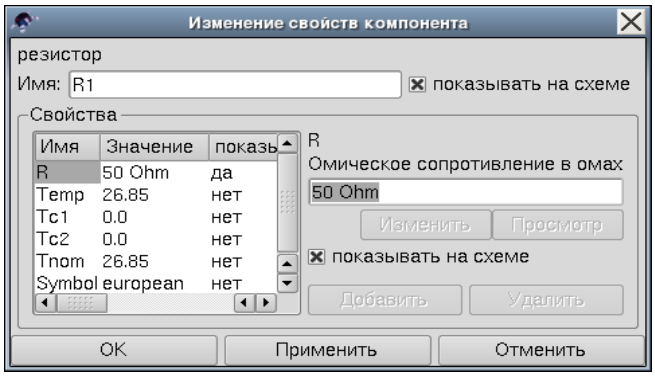


Рис. 3.2. Диалог свойств компонента

и подсвеченное значение приглашает меня ввести альтернативу. Я должен это сделать, но естественная лень берет верх. Я рассудил, что пятьдесят ом в пятьдесят раз больше, но если я уменьшу ток в один ампер до двадцати миллиампер, выходное

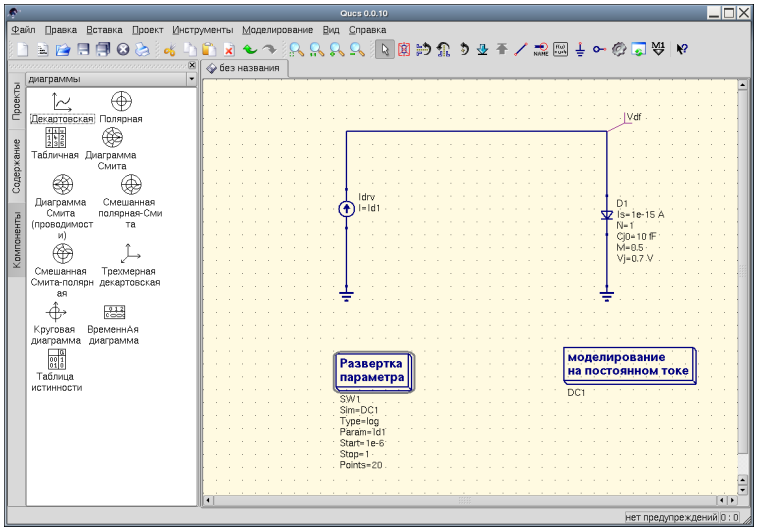
2 Я могу сказать вам результат полученный мной при симуляции, но почему я должен портить вам удовольствие... вернитесь к началу и запустите схему сами. Если вы действительно хотите основательно всем заняться, вы должны также строить схемы и получать результаты...

напряжение получится тем, что нужно. Вы найдете, что такая лень (или обостренное восприятие, зависящее от рассказчика!) может сохранить много времени и сил.

3.2 Когда вещи меняются

Все, что интересно, но не рядом с интересным, как тогда, когда мы начали менять питающее напряжение и наблюдали эффект, будет иметь место, когда мы введем нелинейные элементы.

Простейший нелинейный элемент – это диод, а вопрос, который мы задаем о диоде чаще всего: как диод меняет напряжение с изменением тока? Так что вернемся в Qucs и нарисуем такую цепь...

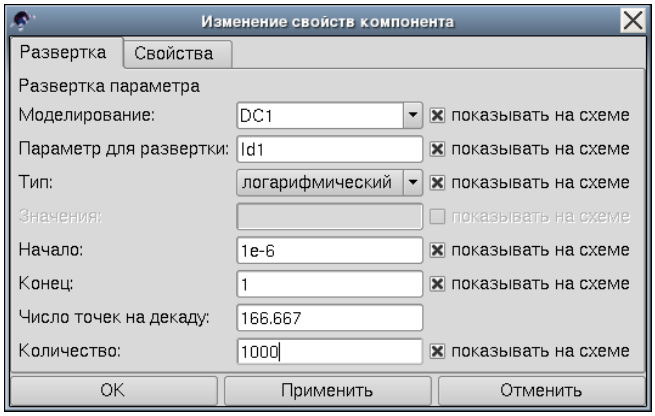


Эта цепь выглядит обманчиво просто, но вводит несколько больше возможностей Qucs, так что давайте пройдем по ним по порядку.

Компоненты вновь выбираются в окне слева и соединяются вместе. Затем выбираются два элемента из окна симуляции.

DC моделирование может прекрасным образом оставаться теперь, как есть, но обратите внимание на имя симуляции: DC1.

Диалоговое окно свойств развертки параметров выглядит похожим на это, когда открыто...

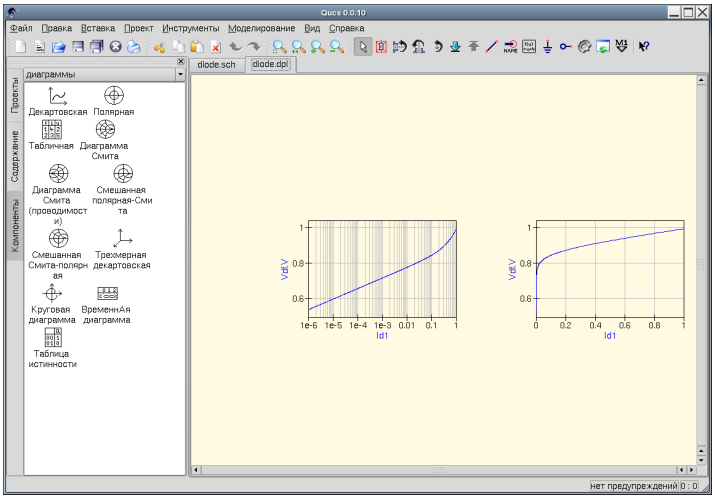


Первые два элемента, обратите внимание, ввод симуляции (здесь **DC1**, соответствующее имени выбранного элемента симуляции) и ввод параметра развертки, здесь обозначенный как **Id1**. Если вы взгляните на источник тока, питающий наш диод, вы увидите, что он, так уж получилось, маркирован как **Idrv**. В результате значение **Id1** свойства компонента источника тока **I** будет качаться в диапазоне значений, обозначенных нашей функцией развертки параметра, названной **SW1**[3].

Остальные вводы устанавливают тип развертки (здесь логарифмический) и диапазон значений, через которые проходит развертка. Вы можете попробовать разные значения в любом из них, чтобы увидеть эффект; одно из преимуществ симулятора перед физическим прототипом в том, что вы не можете разрушить ваш диод, пропустив слишком большой ток через него!

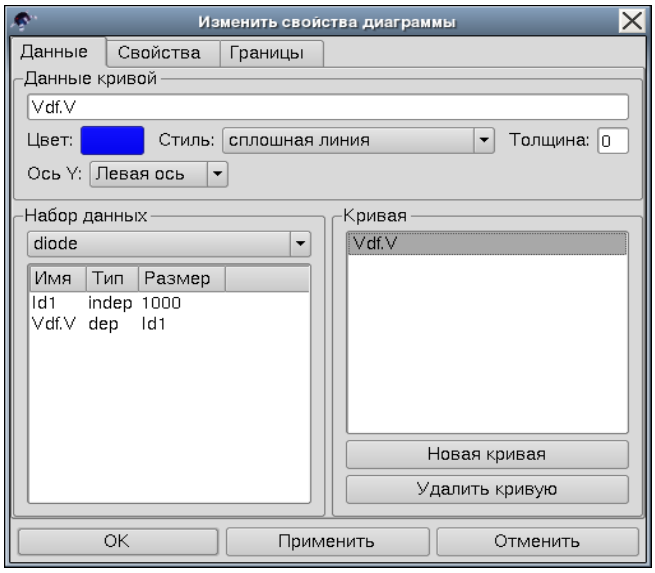
Так что я жму клавишу симуляции, и это переносит меня на страницу результатов, и я создаю пару выходных графиков. Вот как выглядит мой экран...

3 Вы можете изменить это имя, если хотите, в меню свойства окна редактора свойств.



В каждом случае я получаю график зависимости прямого напряжения на диоде (ось Y) от прямого тока (ось X). Левый график имеет логарифмическую шкалу для прямого тока, тогда как правый использует линейную шкалу тока. Как я сделал это? Хорошо, теперь-то вы уже знаете, что все это легко с Qucs!

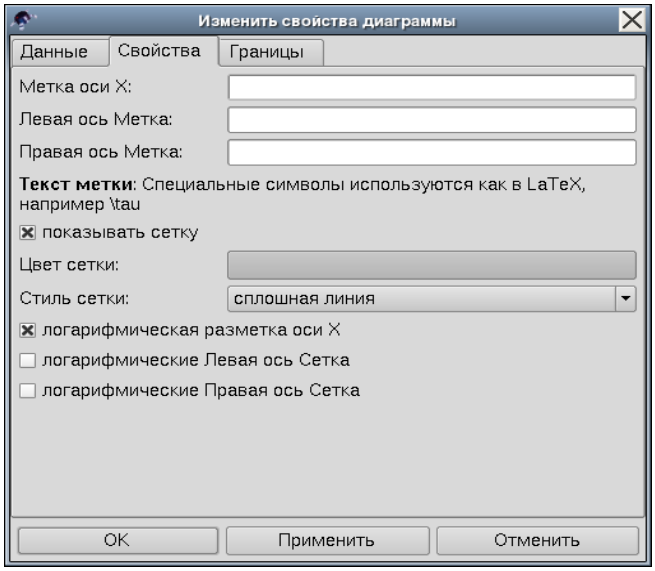
Когда вы выбираете тип графика в окне слева и перетаскиваете его в окно наблюдения, он создает график и открывает диалоговое окно, которое выглядит похожим на это



Левое окно показывает доступные переменные и будут ли они зависимыми или

независимыми. В данном случае ток **Id1** – независимая переменная, а прямое напряжение **Vdf.V** – зависимая. Двойной щелчок по вводу для **Vdf.V**, и он переносится в правую часть; нажмите OK и график будет нарисован.

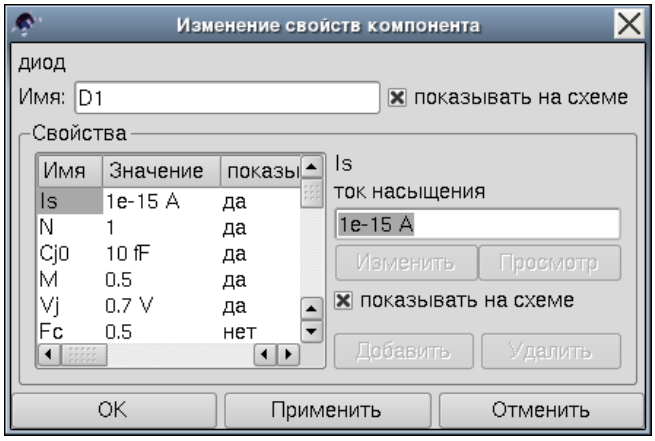
Это даст вам нечто похожее на правый график моего снимка экрана выше. Проделайте это все вновь, но на этот раз, прежде, чем нажимать OK, откройте окно свойств, которое выглядит похожим на это.



Здесь я выбрал логарифмическую ось X, что дает мне график слева. Я также передвинул их, изменив размер, чтобы они выглядели лучше; вы можете осуществить разного рода задумки, если хотите.

Я тайком устроил небольшую проверку, чтобы убедиться, что вы действительно все это проделали. Те, кто делал все, запустив симуляцию, вероятно удивились, почему ваш график выглядит несколько иначе, чем мой. Есть одна деталь, при больших токах на логарифмической шкале ваша кривая – прямая линия, тогда как моя тревожно поднимается вверх. Что случилось?

Все, что я сделал, открыл диалог свойств диода и установил некоторые параметры. Вот как выглядит окно диалога...



и каждый из этих вводов устанавливает один параметр виртуального компонента, который мы используем для модели диода.

Так что же это за параметры? Пришло время исследования одного из восхитительных аспектов компьютерной симуляции цепей, моделирование устройств...

3.3 Модели и параметры

Когда компьютер создает этот маленький кусочек виртуальной реальности, который представляет вашу физическую цепь, он использует заданные уравнения, которые описывают операцию с каждым устройством, включаемым в схему. Уравнение, которое относится к прямому DC напряжению на диоде, как функции тока, это

$$Id = Is \cdot (e^{\frac{Vd}{n \cdot Vt}} - 1)$$

где Vt – это прямое падение напряжения при 25 градусах C, при идеальном соединении, также задаваемом

$$Vt = \frac{Kb \cdot T}{q}$$

где

Kb = постоянная Больцмана
T = температура в градусах по Кельвину
q = заряд электрона

большинство из них – это константы, которые уже знакомы программе. Те, что нам нужно задать, это те, что в списке в окне редактора свойств. Для DC характеристик, чаще всего, есть только несколько, о которых стоит беспокоиться – это Is, ток насыщения, и T, температура. Если мы собираемся пропустить относительно большой ток через диод, мы можем также включить последовательный резистор Rs; если мы беспокоимся о поведении при малых токах, тогда нам нужно добавить параметр обратного тока Isr.

Как нам узнать, какое значение вставить? Большая часть должна быть описана в разделе о моделировании устройств. В действительности, мы как всегда имеем две возможности для выбора: использовать значение, взятое откуда-нибудь еще, или найти наше собственное значение, обычно методом проб и ошибок.

Есть великое множество моделей, подходящих к разным программам симуляции. Возможно, больше всего из доступного и бесплатного подходит для **spice**, и это многое можно загрузить с сайтов компаний, производящих полупроводниковые изделия. Вот, например, типичная spice модель для диода[4] 1N4148:

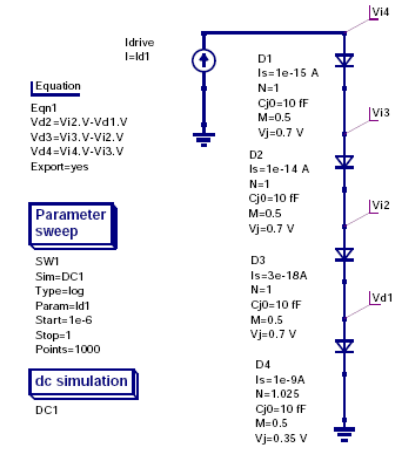
```
.model 1N4148 D(Is=0.1p Rs=16 CJO=2p Tt=12n Bv=100 Ibv=0.1p) 85-??-??
Original library
```

Любые не предоставленные значения подразумеваются значениями по умолчанию.

Другой путь – создать ваши собственные параметры устройства, которые схожи с червяками, которыми надо запастись пред рыбалкой. Вставляйте значения, рисуйте результирующие характеристики, смотрите как они согласуются со значениями опубликованных данных, возвращайтесь назад и подстраивайте значения; продолжайте этот процесс, пока результаты вас не удовлетворят или не измучают.

Вот, например, цепь для быстрого сравнения прямых характеристик диодов с различными значениями параметров.

4 Я не знаю, откуда это появилось, так что не знаю автора. Большинство библиотек с авторскими правами, даже если они в свободном распространении.



А вот графический вывод...

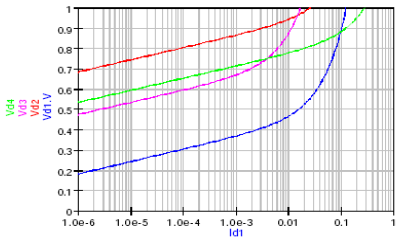


Рис. 3.3. Прямое напряжение на диоде

Зеленая и пурпурная кривые типичны для устройств 1N4148 и 1N4448; другие – это средние и низко-барьерные устройства Шотки. Я сделал первый проход сравнения со справочным листком, но не могу гарантировать, что эти кривые нечто большее, чем мои наилучшие приближения[5].

Если вы хотите получить больше информации о том, что делает каждый параметр, было нечто, написанное ранее, особенно для spice, по предмету; google search быстро обнаружит большую часть из этого. Qucs приходит с документом, который содержит детали по его моделям, и, будучи open source, всегда с, собственно, кодом.

Большинство из нас занимают позицию принятия на веру больших обсуждений, и совпадение кривых со справочными данными – это наилучшее, что мы можем. Это еще

5 Я полагаю, вы сейчас озадачены снимками с экрана, но я только распечатал схему и отобразил файлы из Qucs; вы можете найти раздел печати в меню файл, и, если вы хорошо попросите, он распечатает вам postscript файл.

один образец фундамента в инженерии, принцип утки (Duck Principle[6]): если вы не можете обнаружить разницу между поведением вашей модели и физического устройства, тогда они, для инженерных целей, одинаковы. Подойдите к этому с другой стороны, когда разница между моделью и реальным устройством падает ниже уровня неопределенности измерений, это уже не имеет значения.

В любом случае, разброс компонент в реальном мире имеет тенденцию делать погрешности тонких деталей модели чем-то академическим, как мы увидим при моделировании более сложных устройств.

6 Обычно звучит так: если это выглядит как утка, ходит как утка, крякает как утка и имеет вкус утки, тогда, для любой практической цели – это утка.

4 Начало работы с цифровыми цепями

Симуляция

4.1 Введение

21 Января 2006 Qucs 0.0.8 был выпущен группой разработчиков Qucs. Это первая версия пакета, включающего симуляцию цифровых цепей, базируемую на VHDL. FreeHDL[7] было выбрано в качестве «движка» VHDL. В период, последовавший за выходом 0.0.8, имела место значительная активность, сконцентрированная вокруг поиска и исправления ошибок в цифровом коде симуляции Qucs. Многие из обнаруженных, сегодня включены в последний CVS код и будут, в конечном счете, устранены в следующих выпусках Qucs (сегодня есть версия 0.0.10). Примечания в этом руководстве относятся к моим контактам с другими пользователями Qucs, к некоторым подспудным идеям, касающимся возможностей и ограничений текущего состояния Qucs VHDL симуляции. Значительная часть информации, приведенная здесь, была ассемблирована автором, при участии Michael Margraf в деле тестирования и отладки VHDL кода, генерируемого Qucs. В дальнейшем, если есть большой интерес в этих замечаниях, или действительный к Qucs VHDL симуляции в целом, я буду обновлять все по мере улучшения возможностей Qucs по цифровой симуляции.

Цифровая симуляция Qucs следует за сложным набором шагов, которые более всего ясны для пользователей программ. На первом шаге схема, представляющая цифровую цепь для тестирования, рисуется. Эта схема содержит не соединенную группу цифровых компонент Qucs, кто-то (или многие пользователи) определяет цифровые подсхемы (если требуется), и копий иконок цифровой симуляции с временным, или таблиц истинности, набором параметров. На втором шаге информация, записанная на схеме цепи, конвертируется в текстовый файл, содержащий VHDL установки. Этим описываются компоненты схемы, их соединение и тестовые метки для представления симуляции схемы. Далее FreeHDL запускается из Qucs для конвертирования файла кода VHDL в C++ исходную программу. Это компиляция для формирования выполняемого машинного кода симуляции оригинальной цепи. И, наконец, Qucs запускает эту программу, собирает данные сигналов, как события цифровых сигналов, которые имеют место, и отображает графики сигналов, как функцию времени или цифровых данных, в формате таблицы истинности.

Код VHDL, генерируемый Qucs 0.0.8, ограничен в его рамках следующими

7 FreeHDL Project, http://www.freehdl.seul.org/.

факторами:

- Цифровые вентили описаны потоком данных совпадающих установок.
- Триггеры (Flip-flop) и генераторы цифровых сигналов описаны установками процесса.
- Провода, соединяющие компоненты (сигналы), могут быть только типа бит, как определено в стандарте VHDL библиотеки[8].
- Структуры цифровых шин не допустимы в данном выпуске пакета Qucs.
- Цифровые подсхемы могут быть нарисованы, как схемы, и ассоциированы с символами на манер того, что делается с аналоговыми подсхемами.
- Выводы цифровых подсхем могут иметь тип in, out, inout или analog. Qucs обходится с выводами типа analog также, как с VHDL выводами типа inout.
- Будучи определены, цифровые подсхемы могут размещаться и соединяться с другими компонентами схем.
- Множественные копии тех же цифровых подсхем допустимы на единственной схеме.
- Цифровые подсхемы могут быть также вложенными; вложения протестированы до глубины четыре.

4.2 Симуляция простых цифровых схем

Самая основная форма цифровой цепи, которая может быть симулирована, такая, которая всецело состоит из предопределенных цифровых компонент Qucs, нарисованных в схеме, имеющей только один уровень иерархии разработки. Таблица истинности для простой комбинационной цепи этого типа показана в таблице 7.1.

Выход F может быть выражен суммой произведений булевых форм, как

$$\bar{A}\bar{B} \quad \bar{A}\bar{C} \quad \bar{B}\bar{C}$$

8 Сигналы типа «только бит» определяют логические сигналы «0» и «1». Следует позаботится, чтобы соединения сигналов не обнаруживалось в процессе симуляции, поскольку результирующее логическое состояние не может быть моделировано с типом «бит». Соединение сигналов могут случиться, когда два или более цифровых устройства пытаются перевести один и тот же провод в сигнал логического «0» и логической «1» в одно и то же время. Более того, не возможно симулировать представление устройств с третьим состоянием, используя VHDL сигнал типа «бит».

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Таблица 4.1. Таблица истинности для логической цепи со входами A, B, C и выходом F.

$$F = A.B.C + A.B.C + A.B.C + A.B.C$$

После минимизации, используя булеву алгебру или карты Карно, выход F становится

$$F = A.C + B.\overline{C}$$

Схема примера 1 показана на рис. 4.1. Эта диаграмма была построена с использованием той же техники, что применялась при рисовании аналоговых схем.

4.2.1 Примечания по рисованию цифровых схем

- Только предопределенные компоненты Qucs могут применяться при рисовании цифровых схем, это (1) цифровые компоненты перечисленные в иконках окна цифровых компонент, (2) символ земли и (3) иконка цифровой симуляции.
- Полезный прием при рисовании цифровых схем – это принимать матричное приближение, показанное на рис. 4.1. Входные сигналы следуют сверху вниз по схеме, а выходные сигналы позиционируются на правой стороне по горизонтальной линии. Это делает проверку цепей схемы более легкой, чем в случае, когда диаграммы имеют соединяющие компоненты провода в неструктурированном виде.
- Входным и выходным проводам (сигналам) должны быть присвоены имена, по именам цепей для симуляции A, B, C и F на рис. 4.1. Если сигнальные провода не именованы пользователем, Qucs будет размещать на них разные произвольные имена. Это может сделать идентификацию и выбор сигналов для отображения на графике выводного окна, и проверку и поиск ошибок в большой схеме, значительно более трудными, чем это действительно требуется.
- Отметьте на рис. 4.1. интернациональные символы для логических вентилей, показанных на схеме.

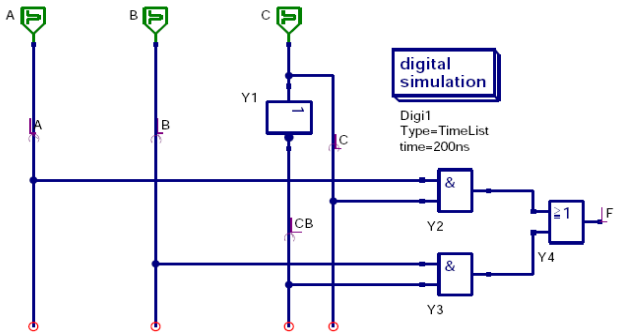


Рис. 4.1. Qucs схема для минимизации логической функции F

4.3 VHDL код, генерированный Qucs

Щелчок по клавише меню Qucs **Моделировать** (или нажатие на клавишу F2) начинает процесс симуляции. В начальной стадии этого процесса Qucs записывает текстовый файл на диск, файл содержащий код VHDL для симулируемой схемы. Этот файл может быть отображен щелчком по показанному в конце выпадающего меню netlist или нажатием клавиши F6. Код VHDL, произведенный Qucs для цепи, показанной на рис. 4.1, отображен в таблице 4.2.

Сигналы идентифицируемые как *nnnet0* и *nnnet1* в таблице были присвоены Qucs; *nnnet0* и *nnnet1* внутренние сигнальные сети, которые не именованы на схеме, показанной на рис. 4.1. Рис. 4.2 иллюстрирует начальную секцию типичного Qucs цифрового функционального графика. Этот стиль графика иллюстрирует событийные сигналы без задержек распространения компонент. Если требуется, задержки сигналов могут быть специфицированы для индивидуальных вентилей и других компонент (из меню редактора свойств компонент). VHDL код, генерированный для компонент с задержками, будет затем отображать подобные изменения, например, добавление 10 ns задержки сигналу CB в таблице 4.2 генерирует VHDL код

```
CB <= not C after 10 ns;
```

Читатели, возможно, окажутся достаточно наблюдательны, чтобы отметить – номер версии Qucs в таблице 4.2 листинга VHDL это 0.0.9. Это текущий номер версии CVS разработки. Qucs 0.0.9 включает некоторое количество важных зафиксированных ошибок. Оставшиеся примечания подразумевают, что читатели загрузили и

откомпилировали последний CVS код с Sourceforge.net[9].

```
   Qucs 0.0.9 tut1 ex1.sch
entity TestBench is
end entity ;
use work . all ;

architecture Arch TestBench of TestBench is
signal CB, A, B, F, C,
       nnnet0 ,
       nnnet1 : bit ;
begin
       nnnet0 <=C and A;
       nnnet1 <= CB and B;
       CB <= not C;

       A: process
       begin
              A <= ' 0 ' ; wait for 40 ns ;
              A <= ' 1 ' ; wait for 40 ns ;
       end process ;

       B: process
       begin
              B <= ' 0 ' ; wait for 20 ns ;
              B <= ' 1 ' ; wait for 20 ns ;
       end process ;

       F <= nnnet1 or nnnet0 ;

       C: process
       begin
              C <= ' 0 ' ; wait for 10 ns ;
              C <= ' 1 ' ; wait for 10 ns ;
       end process ;
end architecture ;
```

Таблица 4.2. VHDL код для схемы, показанной на рис. 4.1

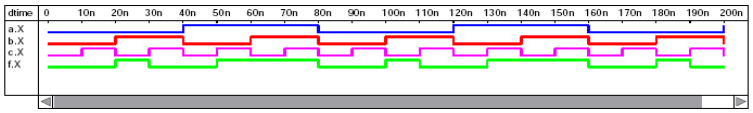


Рис. 4.2. Цифровая функциональная диаграмма для схемы на рис. 4.1

4.4 Таблицы истинности

Таблицы истинности – это один из самых фундаментальных и удобных способов отображения данных цифровых цепей. Qucs имеет встроенные средства, которые позволяют генерировать таблицы истинности из чертежа схемы. Это средство

9 Заметьте, что Qucs Linux выпуска 0.0.8 нормально симулирует цифровую схему с единственной иерархией без ошибок. Однако Qucs 0.0.8 аварийно завершает работу в фазе конверсии VHDL в C++, если схема имеет больше одного символа земли.

особенно полезно при поиске ошибок минимизации логической разработки. Давайте рассмотрим простой, но поучительный пример: логическая схема имеет четыре двоичных входа A, B, C и D, и один выход P. Выход P в логической «1», когда входы ABCD нумеруются в десятичной последовательности 3, 5, 7, 11 и 13. В булевой форме сумма произведений

$$P = \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.\overline{B}.C.D + A.B.\overline{C}.D$$

Это упрощается до

$$P = D.(A.B + B\ \ C)$$

Схема для суммы произведений уравнения P показана на рис. 4.3(a). Похожая на рис. 4.3(b) представляет схему для минимизированного уравнения P. Установка типа цифровой симуляции в виде таблицы истинности, а не в виде временного списка, заставляет Qucs по нажатию клавиши F2 генерировать таблицу истинности по информации, предоставляемой схемой. Число входов таблицы истинности и фактических выходов соответствует числу входов генераторов и числу именованных выходов. Таблица истинности для обоих схем дана в таблицах 4.3(a) и 4.3(b). Сравнение этих двух таблиц ясно показывает, что они не одинаковы и, более того, подтверждают, что минимизированное решение не корректно. Переделка процедуры минимизации указывает на ошибку – пропущенную инверсию сигнала. Корректное булево уравнение для P

$$P = D.(\overline{A}.B + B\ \ C)$$

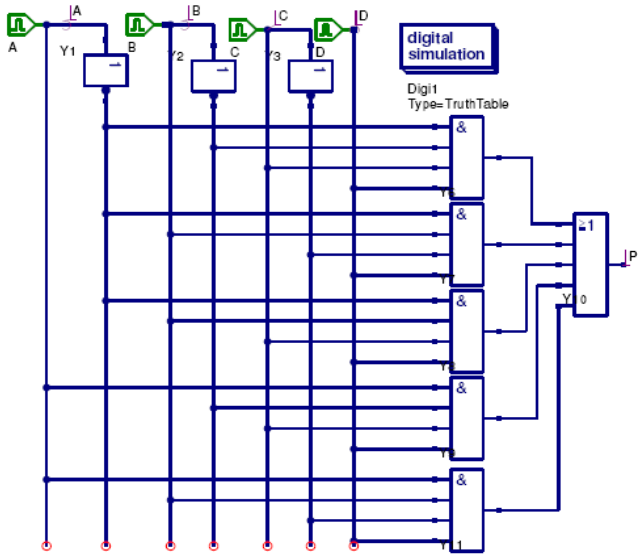


Рис. 4.3(a). Диаграмма для суммы произведений уравнения P

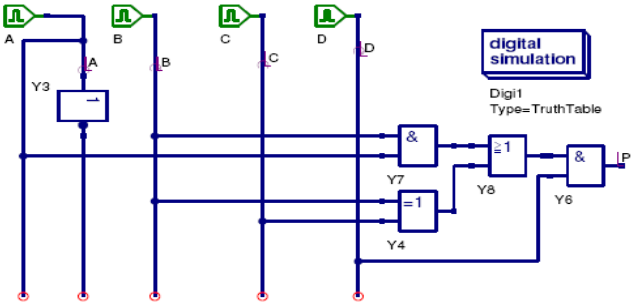


Рис. 4.3(b). Диаграмма для минимизированного уравнения P

	a.X	b.X	c.X	d.X	p.X
00000	0	0	0	0	0
00001	0	0	0	1	0
00010	0	0	1	0	0
00011	0	0	1	1	1
00100	0	1	0	0	0
00101	0	1	0	1	1
00110	0	1	1	0	0
00111	0	1	1	1	1
01000	1	0	0	0	0
01001	1	0	0	1	0
01010	1	0	1	0	0
01011	1	0	1	1	1
01100	1	1	0	0	0
01101	1	1	0	1	1
01110	1	1	1	0	0
01111	1	1	1	1	0

Таблица 4.3(a). Таблица истинности для суммы произведений уравнения P

	a.X	b.X	c.X	d.X	p.X
00000	0	0	0	0	0
00001	0	0	0	1	0
00010	0	0	1	0	0
00011	0	0	1	1	1
00100	0	1	0	0	0
00101	0	1	0	1	1
00110	0	1	1	0	0
00111	0	1	1	1	0
01000	1	0	0	0	0
01001	1	0	0	1	0
01010	1	0	1	0	0
01011	1	0	1	1	1
01100	1	1	0	0	0
01101	1	1	0	1	1
01110	1	1	1	0	0
01111	1	1	1	1	1

Таблица 4.3(b). Таблица истинности для минимизированного уравнения P

4.5 Цифровые подсхемы

Хотя возможно рисовать сложные схемы, используя только предопределенные цифровые компоненты, полученные с Qucs, эта техника может быть крайне скучна, и, конечно, склонна к ошибкам. При черчении больших схем мы требуем процедуры разработки, которая естественно подразделяет группы цифровых компонент на самостоятельные единицы. Последние могут быть позже обрабатываться так же, как и базовые цифровые компоненты, когда размещаются и соединяются в чертеже схемы. В мире разработки аналоговых и цифровых цепей такие единицы часто называются подсхемами[10]. Подсхема определена тремя главными атрибутами плюс некоторое количество других свойств. Главные атрибуты – это, в первую очередь, цифровая цепь, которая определяет цифровую функцию, затем, символ цепи, который изображает цепь на верхнем уровне иерархии разработки, и, наконец, подсхема выводов ввода/вывода, показанные на символе подсхемы. Другие свойства включают, например, задержки сигналов. Процесс генерации цифровых подсхем идентичен процессу, используемому для аналоговых подсхем. Это лучше продемонстрировать на примере. Рис. 4.4 показывает схему для четырех-входовой комбинационной цепи.

После вычерчивания подсхемы выводы входа и выхода[11] прикрепляются к сигнальным портам. Выводы входного порта типа in показаны на диаграмме цепи, как зеленый символ – сигналы W, X, Y и Z на рис. 4.4. Вывод выходного порта типа out окрашен в красный цвет, сигнал G на рис. 4.4. Прохождение сигнала через порт обозначено стрелкой у символа порта. Сигналы входа/выхода и многие другие сигналы, которые нужно легко идентифицировать, имеют имена. Как только подсхема завершена, нажмите клавишу F3, чтобы Qucs генерировал символ подсхемы. Инструмент рисования, введенный как иконка в окне рисования Qucs, может быть использован для редактирования сгенерированных символов подсхем. Выводы порта ввода/вывода на символе подсхемы имеют тот же тип и имя, как и те, что на оригинальной подсхеме. Рис. 4.5 показывает окончательный символ для подсхемы COMB1. В этих заметках контур символа показан нарисованным в соответствии с международным кодом для логических символов[12]. Для проверки нашей новой подсхемы мы поместим ее символ на чистый лист и используем тестовые сигналы для выводов входа, затем посмотрим сигналы на выходном выводе. Рис. 4.6 показывает типичную тестовую цепь. Подсхема Gen4bit генерирует 4x-битовый тестовый шаблон, синхронизированный со входом

10 Симулятор схем SPICE – это хорошо известный пример повсеместного использования CAD программы, широко использующей подсхемы при разработке цепей.

11 Qucs 0.0.8 имеет ошибку, которая приводит к ошибке VHDL компилятора, когда выводы подсхемы специфицированы, как выводы типа out. Для обхода проблемы следует задать выходные выводы, как выводы типа analog. Подпрограммы Qucs, генерирующие VHDL код цепи, конвертируют вывод типа analog в VHDL тип inout. FreeHDL затем получает возможность компилировать сгенерированный VHDL код без ошибки. Эта ошибка была скорректирована в Qucs 0.0.9.

12 Ian Kampel, практическое введение в новые логические символы (A practical introduction to the new logic symbols), Butterworths, 1985, ISBN 0-408-01461-X.

тактового генератора. Спецификация для Gen4bit дана в следующем разделе этих заметок[13]. Графическое изображение тестового шаблона и выходного сигнала G показаны, как функция времени, на рис. 4.7.

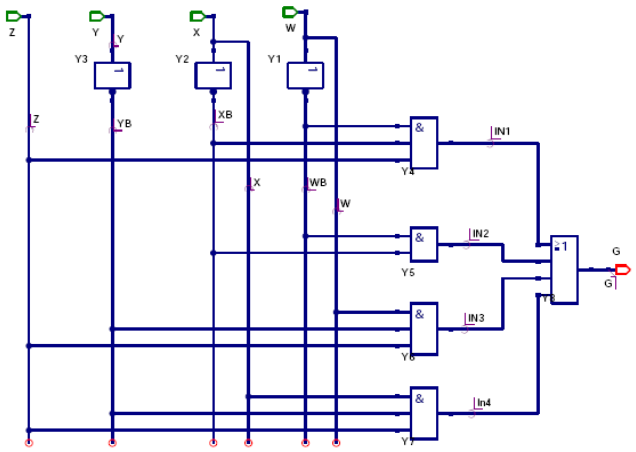


Рис. 4.4. Комбинационная логическая цепь со входами W, X, Y, Z и выходом G

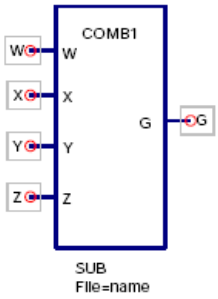


Рис. 4.5. Символ Qucs для логической цепи со входами W, X, Y, Z и выходом G

13 Подсхема Gen4bit включает другие вложенные подсхемы. Qucs 0.0.8 имеет ошибку, приводящую к ошибке VHDL компилятора с некоторыми конфигурациями вложенных подсхем. Это было устранено в версии 0.0.9.

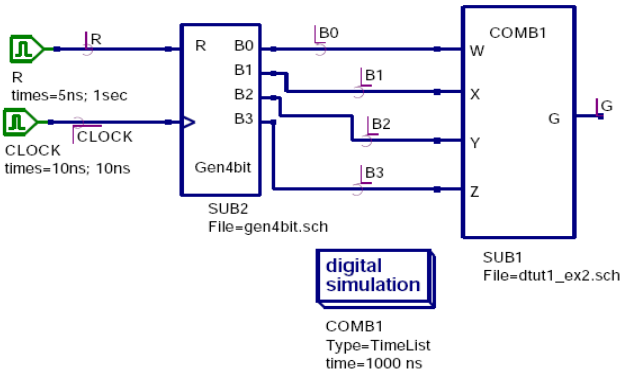


Рис. 4.6. Тестовая схема для логической цепи со входами W, X, Y, Z и выходом G

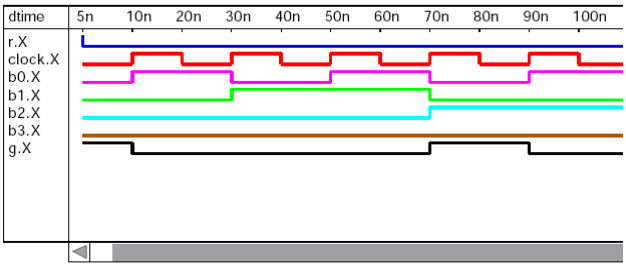


Рис. 4.7. Функциональная диаграмма для логической цепи со входами W, X, Y, Z и выходом G

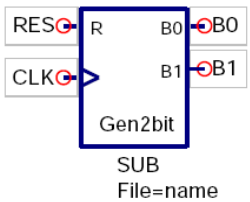
4.6 Построение цифровой библиотеки компонент

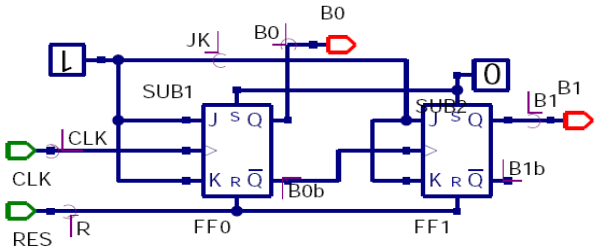
Графический интерфейс пользователя Qucs включает хорошие средства поддержки проекта. Комбинация этих возможностей с возможностями подсхем Qucs предоставляет все инструменты, требуемые для разработки библиотеки общих цифровых компонент. Такая библиотека может быть сохранена в мастер-проекте, а файлы индивидуальных компонент импортированы в другие проекты, когда это требуется. Вот несколько компонент, которые я разработал в процессе недавней серии целевых тестов по выявлению ошибок в VHDL коде, генерируемом Qucs.

4.6.1 Логический нуль

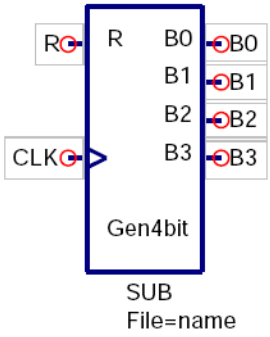
4.6.2 Логическая единица

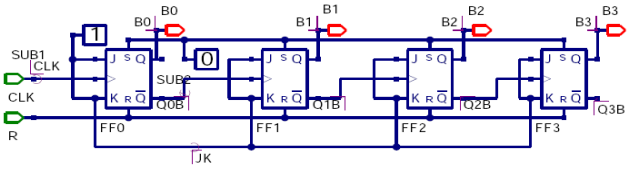
4.6.3 G2bit – 2х-битовый генератор шаблона




B0
JK
B0
1
SUB1
0
SUB2
B1
B1
CLK
J S Q
CLK
K R Q
B1b
B0b
RES
R
FF0
FF1

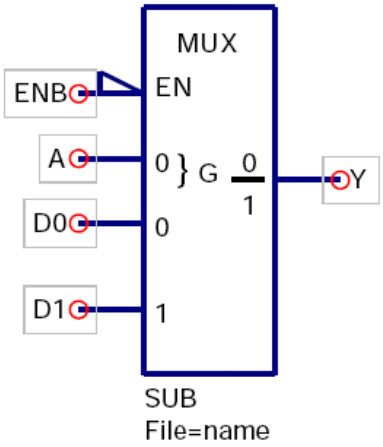
4.6.4 G4bit – 4-битовый генератор шаблона

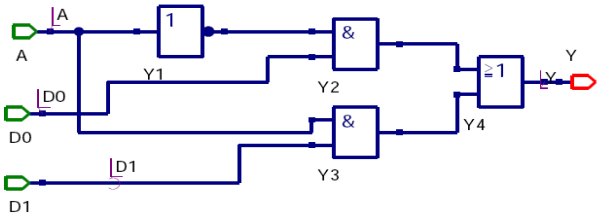




4.6.5 MUX2to1 – 2 входа на 1 мультиплексор

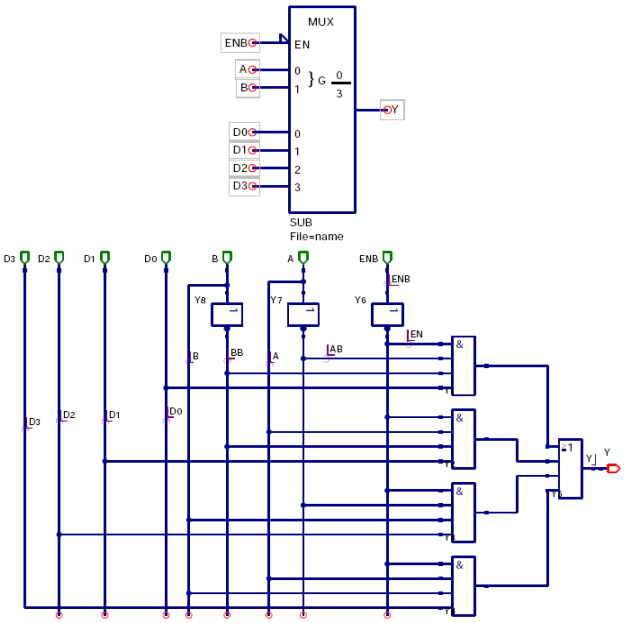
EN	A	Y
1	X	L
0	0	D0
0	1	D1



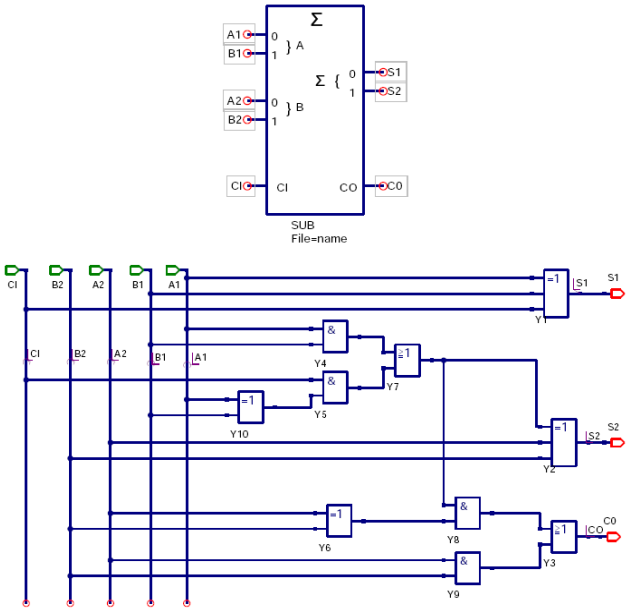


4.6.6 MUX4to1 – 4 входа на 1 мультиплексор

B	A	EN	Y
X	X	1	0
0	0	0	D0
0	1	0	D1
1	0	0	D2
1	1	0	D3



4.6.7 2-х битовый сумматор



4.7 Код подсхемы ***VHDL,*** генерированный ***Qucs***

Qucs генерирует отдельные объектно-архитектурные модели для каждой подсхемы. Определения этого компонента компилируются в рабочую библиотеку FreeHDL. Вот код VHDL из двух предыдущих примеров.

4.7.1 Gen2bit

```
entity Sub_gen2bit is
        port (CLK: in bit ;
                R: in bit ;
                nnout_B0 : out bit ;
                nnout_B1 : out bit );
end entity ;
use work . all ;
architecture Arch_Sub_gen2bit of Sub_gen2bit is
        signal B0b ,
                B1b ,
                JK,
```

```
                nnnet0 ,
                B0 ,
                B1 : bit;
begin
        FF0 : process (nnnet0 , R, CLK)
        begin
                if (R= '1 ') then B0 <= '0 ';
                elsif ( nnnet0= '1 ') then B0 <= '1 ';
                elsif (CLK= '1 ' and CLK' event ) then
                        B0 <= (JK and not B0) or ( not JK and B0 ) ;
                end if ;
        end process ;
        B0b <= not B0 ;


        FF1 : process (nnnet0 , R, B0b)
        begin
                if (R= '1 ') then B1 <= '0 ';
                elsif ( nnnet0= '1 ') then B1 <= '1 ';
                elsif (B0b= '1 ' and B0b' event) then
                        B1 <= (JK and not B1) or ( not JK and B1 ) ;
                end if ;
        end process ;
        B1b <= not B1 ;

        SUB2 : entity Sub_logic_zero port map (nnnet0 );
        nnout_B0 <= B0 or '0 ';
        nnout_B1 <= B1 or '0 ';
        SUB1 : entity Sub_Logic_one port map (JK ) ;
end architecture ;
```

4.7.2 2-битовый сумматор

```
entity Sub_fadd_2bit is
        port (A1 : in bit ;
        B1 : in bit ;
        A2 : in bit ;
        B2 : in bit ;
        CI : in bit ;
        nnout_S1 : out bit ;
        nnout_S2 : out bit ;
        nnout_CO : out bit );
end entity ;

use work . all ;
architecture Arch_Sub_fadd_2bit of Sub_fadd_2bit is
        signal nnnet0 ,
                nnnet1 ,
                nnnet2 ,
                nnnet3 ,
                nnnet4 ,
                nnnet5 ,
                nnnet6 ,
                S2 ,
                CO,
                S1 : bit;
```

```
begin
        S1 <= CI xor B1 xor A1 ;
        nnnet0 <= B2 xor A2 ;
        nnnet1 <= nnnet0 and nnnet2 ;
        nnnet3 <= B2 and A2 ;
        nnnet2 <= nnnet4 or nnnet5 ;
        nnnet4 <= nnnet6 and CI ;
        nnnet5 <= B1 and A1 ;
        S2 <= B2 xor A2 xor nnnet2 ;
        CO <= nnnet3 or nnnet1 ;
        nnnet6 <= B1 xor A1 ;
        nnout S2 <= S2 or '0 ';
        nnout CO <= CO or '0 ';
        nnout S1 <= S1 or '0 ';
end architecture ;
```

4.7.3 Замечания по генерации VHDL подсхем

- Qucs предопределяет генерацию цифровых компонент, совпадающую с потоком данных состояний сигналов или состояний процесса.
- Ранее определенные символы подсхем генерируют VHDL карту состояний портов.
- Сигналы объекта порт типа out защищены от чтения, как входные сигналы, маскированием каждого выходного сигнала с использованием логической функции signal-name OR '0'[14].
- VHDL

  ```
  use work . all ;
  ```

  состояние включено перед каждым определением архитектуры подсхемы для обеспечения того, чтобы FreeHDL могло найти любую вложенную подсхему[15].
- Полный файл кода VHDL для цифровой разработки составлен из выходных тестовых стендов объектно-архитектурной модели плюс объектно-архитектурные модели для каждой подсхемы, специфицированной в разработке.

4.8 Вложение подсхем: более сложный пример разработки

Теоретически нет ограничений на глубину вложения подсхем, допускаемой в Qucs. На

14 Попытка прочитать сигналы объекта порта типа out вызывают в VHDL ошибку компиляции.

15 Строго говоря, это не необходимо – задавать состояние использования рабочей библиотеки, поскольку эта библиотека обычно видима в любое время, когда компилируются объектно-архитектурные модели. Однако при настоящем состоянии разработки FreeHDL выяснилось, что это необходимо при использовании предопределенной FreeHDL VHDL карты библиотеки.

практике большинство цифровых схем могут быть сконструированы с максимальным числом уровней иерархии разработки равным четырем или пяти. Рис. 4.8 показывает пример, который использовался для тестирования Qucs на предмет вложения подсхем. Разработка – простая RTL функция, которая использует мультиплексор для для передачи данных от одного из двух входных регистров к единственному выходному регистру. Следующий раздел этих заметок обрисовывает в деталях спецификацию подсхем, необходимых для построения RTL разработки. Набор графиков симуляции, показывающий операцию передачи в регистр, на рис. 4.9.

4.8.1 4х-битовая разработка RTL

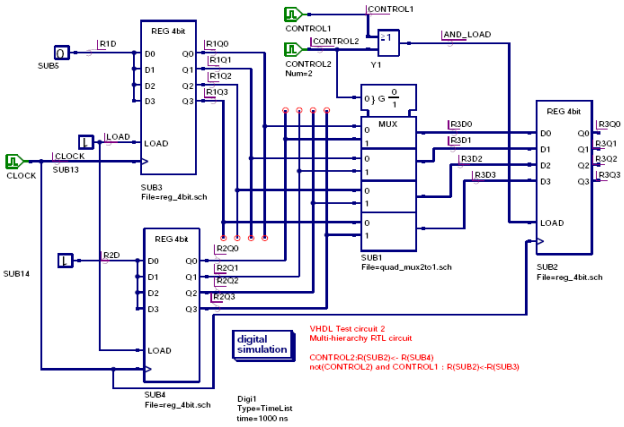
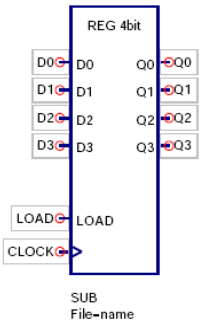
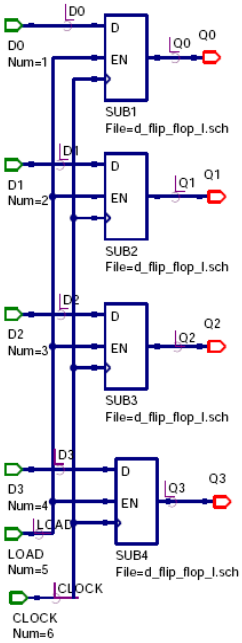


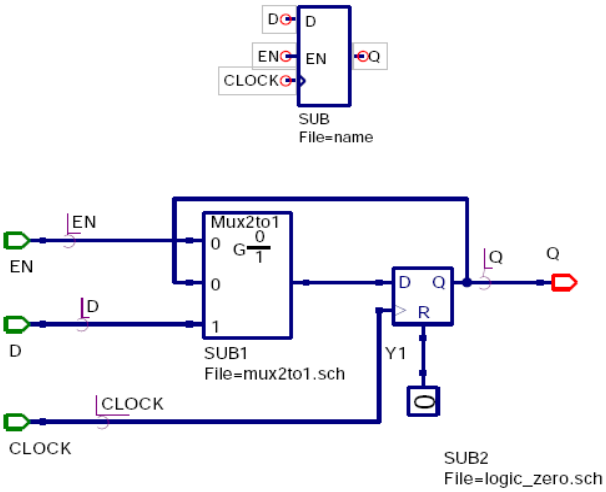
Рис. 4.8. Схема верхнего уровня

Reg4bit

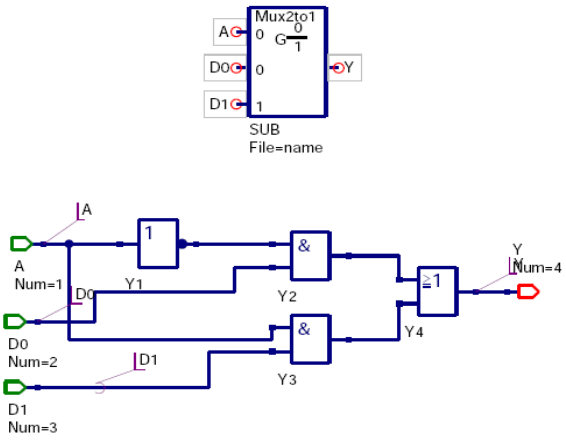

REG 4bit
D0
D1
D2
D3
Q0
Q1
Q2
Q3
LOAD
CLOCK
SUB
File=name


D0
Num=1
D
EN
Q0
SUB1
File=d_flip_flop_l.sch
D1
Num=2
Q1
SUB2
File=d_flip_flop_l.sch
D2
Num=3
Q2
SUB3
File=d_flip_flop_l.sch
D3
Num=4
LOAD
Num=5
Q3
SUB4
File=d_flip_flop_l.sch
CLOCK
Num=6

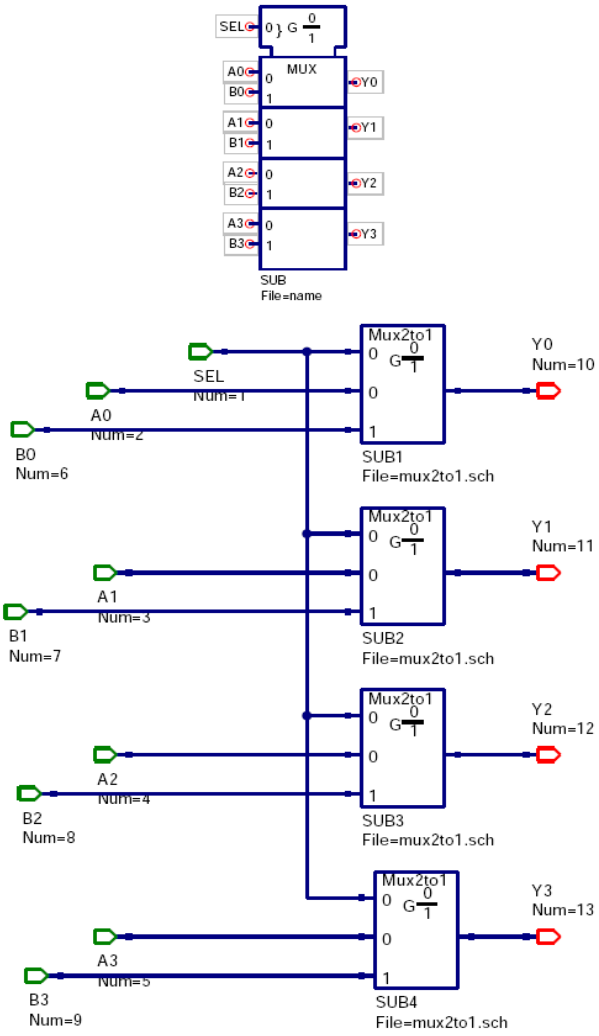
D flip-flop (триггер) с возможностью загрузки



Mux2to1



QuadMux


SEL
A0
B0
A1
B1
A2
B2
A3
B3
MUX
Y0
Y1
Y2
Y3
SUB
File=name
Mux2to1
SUB1
File=mux2to1.sch
SUB2
SUB3
SUB4
Num=1
Num=2
Num=3
Num=4
Num=5
Num=6
Num=7
Num=8
Num=9
Num=10
Num=11
Num=12
Num=13

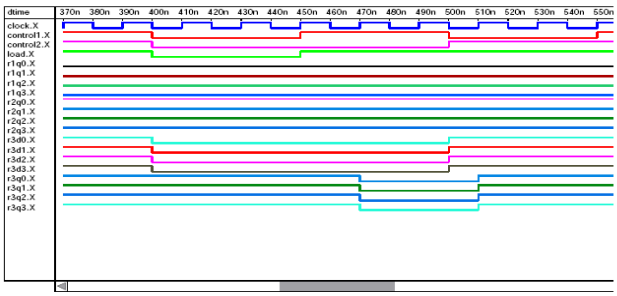


Рис. 4.9. Образец графика симуляции для RTL разработки

4.9 Обновление номер один: Май 2006

Хотя прошло совсем немного времени с момента публикации первой версии заметок к этому цифровому руководству на Qucs Sourceforge Web-сайте, произошло многое в мире цифровой симуляции Qucs. Ошибки в коде Qucs были найдены и исправлены, и ряд новых возможностей добавлено в программу. Это расширяет мощь цифровой симуляции Qucs, и дает пользователям некоторое представление о дальнейшей эволюции пакета. Цель данных заметок: во-первых, оповестить читателей о тех новинках, что появились в цифровой симуляции Qucs, и, во-вторых, пояснить, как эти новинки использовать. Заметьте, однако, заметки не преследуют цели обучить читателей, как программировать, используя VHDL[16].

4.9.1 Ошибки, коррекция и небольшие изменения в коде цифровой симуляции Qucs

Все ошибки, о которых сообщалось в первой версии этих заметок, были скорректированы в последнем Qucs CVS коде. Эти коррективы, конечно, также включены в выпуск Qucs 0.0.9. В процессе тестирования некоторое количество других мелких, но существенных, ошибок было также найдено и устранено, это включает

- Много-входовые вентили (три или более входов) типа nand и nor приводили к ошибке FreeHDL компиляции из-за ошибки в VHDL коде, генерируемом Qucs.
- Имена сигналов и, например, имена компонент из одной буквы, которые были

16 Хорошее введение в язык VHDL и его приложение к разработке цифровых систем может быть найдено в Digital System Design using VHDL by Charles H. Roth, Jr, PWS Publishing Company, 1997, ISBN 0-534-95099-X.

аббревиатурой физических устройств вызывали сбой компиляции.
- Изменение времени задержки цифровых компонент приводило к тому, что соединение компонента в схеме удалялось.
- GUI проблемы приводили к ошибкам в коде поворотов и отражений символов.
- Ошибки Qucsconv кода преобразования приводили к тому, что цикл цифровой симуляции Qucs давал сбой до вывода графика TimeList.

Было сделано некоторое количество изменений либо VHDL кода, генерируемого Qucs, либо ввода схемы GUI, что включает

- VHDL код, генерируемый Qucs для символа земли был изменен
```
gnd <= gnd and '0 ' ;
to
gnd <= '0 ' ;
```
- Символ для порта цифрового ввода был изменен с символа аналогового вывода на такой, что содержит цифровые in и out выводы, нарисованные бок-о-бок. Это отражает двунаправленное состояние порта inout.

Более полный список всех коррекций ошибок и других программных модификаций может быть найден в лог-файлах изменений Qucs.

4.9.2 Новые возможности цифровой симуляции

Диаграмма потока, иллюстрированного на рис. 4.10, показывает некоторое количество разных трассировок симуляции для тестируемой цифровой цепи. Возможности цифровой симуляции Qucs были улучшены, чтобы включить прямую симуляцию VHDL испытательных стендов и симуляции схем, которые включают цифровые компоненты, специфицированные VHDL объектно-архитектурными моделями. Различные комбинации, которые пользователи могли адаптировать к Qucs элементам цифровых цепей, это:

1. Элементы цепей схемы используют предопределенные символы цифровых компонент, генерация подсхем использует те же самые символы и копирует иконки цифровой симуляции; это приближение описано в первой версии этих заметок.
2. Элемент цепи, идентичный символам 1 плюс для цифровых компонент, специфицированных VHDL объектно-архитектурных моделей.
3. Элемент цепи, использующий Qucs VHDL редактор кода. Введенный текст описывает и тестируемую цепь, и векторы тестирования нужные для оживления входов цепи в процессе симуляции.

Когда тестируемая цепь была введена в Qucs, щелкните по клавише меню Моделировать или нажмите клавишу F2, что запустит процесс цифровой симуляции Qucs.

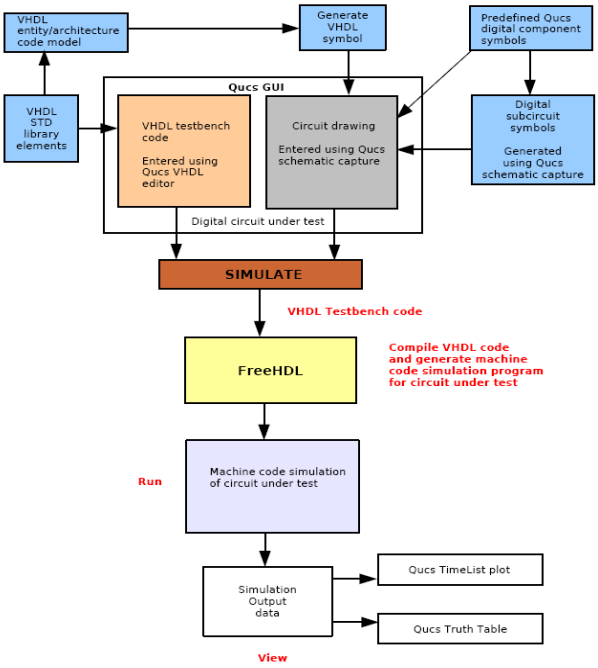


Рис. 4.10. Диаграмма трассировки потока Qucs

4.9.3 Ограничения

Прежде, чем приступить к описанию новых возможностей цифровой симуляции, очень важно, чтобы читатели поняли ограничения, которые присущи различным трассировкам цифровой симуляции, описанным в последнем разделе и иллюстрированным в диаграмме потока, показанной на рис. 4.10. Ввод схемы Qucs позволяет пользователям рисовать цепи, состоящие из символов предопределенных компонент и символов подсхем. На этой стадии разработки GUI цифровые сигналы должны быть типа «bit» (как определено в стандартной библиотеке VHDL – библиотеке STD в пакете FreeHDL), где индивидуальные сигналы проходят через единственный провод. Черчение шинных структур в Qucs типа VHDL bit-vector, например, не реализовано совсем. Это подразумевает, что выводы порта символа устройства должны представлять единственные сигналы. Похожим образом сети, соединяющие выводы на более, чем одном устройстве, могут быть только единичными сигналами сетей, а не шинными структурами. И не похоже, чтобы это было изменено в последующих выпусках Qucs.

Хотя текущий выпуск FreeHDL – это 0.0.1, пакет снабжен существенной подсетью объектов языка VHDL[17]. Основные возможности не поддержанные выпуском 0.0.1:

- Совместные переменные.
- Следующие атрибуты: transaction, quiet, stable и delayed.
- Атрибуты, определенные пользователем.
- Группы.
- Охраняемое задание сигналов.
- Текущие драйверы не могут выключаться.

Программа рисования Qucs TimeList использует вывод данных сигнала машинного кода программы симуляции, генерируемого пакетом FreeHDL[18]. Текущее ограничение программы черчения TimeList в том, что она может отображать только сигналы типа «bit». Графический вид шинных сигналов не может быть отображен.

Выше приведены ограничения, а это значит, что писать VHDL код, который будет компилироваться FreeHDL, возможно, но возможны и проблемы либо с чертежами схем, либо с выводом графиков состояния в цикле симуляции Qucs. Поскольку Qucs развивается, следует ожидать, что эти ограничения будут сняты. По вопросу ограничений последнее замечание: FreeHDL может симулировать цепи, описанные типами данных и средствами, находящимися в

17 Полное описание спецификаций 1987 и 1993 языка VHDL может быть найдено в The Designer's Guide to VHDL by Peter J. Ashenden, second edition 2002, Morgan Kaufmann Publishers, ISBN 1-55860-674-2.

18 Машинный код программы симуляции выводит данные сигналов в VCD формате. Затем это конвертируется в формат данных Qucs TimeList с помощью утилиты qucsconv.

```
IEEE.std_logic_1164
```

библиотеке и других предопределенных библиотеках. Однако в данный момент разработки программного обеспечения Qucs может быть использована только стандартная библиотека VHDL, подразумевая, что следует использовать данные типа «bit» для представления логических сигналов.

4.9.4 Использование Qucs VHDL редактора

Qucs выпуска 0.0.9 включает VHDL текстовый редактор[19], который имеет все обычные средства редактирования плюс цветовое кодирование разных установок языка VHDL. Одно необычное свойство этого редактора – управление масштабом, что позволяет увеличивать или уменьшать размер текста, как это обычно реализовано для чертежей схем. Редактор VHDL включен в пакет Qucs с двумя основными целями, во-первых, для чисто текстового файла VHDL симуляции[20], и во-вторых, для разработки VHDL объектно-архитектурных моделей, которые могут быть связаны с символами ввода схемы. Последнее – это существенное увеличение возможностей программного обеспечения Qucs в появившейся возможности создания библиотек моделей устройств, изготовленных вручную. Эти новые библиотечные устройства, данные в поддержке основного сообщества пользователей Qucs, значительно расширяют потенциал использования пакета Qucs. В этом разделе использование текстового редактора VHDL продемонстрировано серией примеров цифровых цепей. Включенные VHDL листинги показывают типичное использование Qucs базовых типов данных VHDL. Текст также обрисовывает любые ограничения, накладываемые Qucs.

- Пример 1: Сумма произведений (SOP) комбинационной цифровой цепи. Булево уравнение[21] для SOP комбинационной цепи:

  $f = \overline{W}.X.\overline{Y}.\overline{Z} + \overline{W}.\overline{X}.\overline{Y}.\overline{Z} + W.\overline{Y}.\overline{Z} + W.X.Y.Z$

  VHDL код для структурной модели этой комбинационной цепи и ее ассоциированных испытательных стендов дан в следующем листинге.

19 Для запуска нового VHDL редактора щелкните по второй иконке слева на инструментальном меню Qucs. Он также может быть активирован использованием последовательности нажатия клавиш Ctrl+Shift+V.

20 Это метод, еще предпочитаемый многими опытными пользователями VHDL. Однако подход через черчение схем, похоже, становится все популярнее.

21 Булево уравнение для функции f не было минимизировано. Оно в форме производной непосредственно из таблицы истинности, и представлено чисто примером, демонстрирующим использование Qucs VHDL редактора.

```
entity test_vector is    test_vector generator.
       port ( z, y, x, w: out bit);
end entity test_vector ;

architecture behavioural of test_vector is
begin
pz : process is
       begin
              z <= '0' ; wait for 20 ns;
              z <= '1' ; wait for 20 ns;
       end process pz ;
py : process is
       begin
              y <= '0' ; wait for 40 ns;
              y <= '1' ; wait for 40 ns;
       end process py ;
px : process is
       begin
              x <= '0' ; wait for 80 ns;
              x <= '1' ; wait for 80 ns;
       end process px ;
pw : process is
       begin
              w <= '0' ; wait for 160 ns;
              w <= '1' ; wait for 160 ns;
       end process pw ;
end architecture behavioural ;

entity and4 is    4 input and gate.
       port ( in1, in2, in3, in4 : in bit ;
       out1 : out bit );
end entity and4 ;

architecture dataflow of and4 is
       begin
              out1 <= in1 and in2 and in3 and in4 ;
end architecture dataflow ;

entity and3 is    3 input and gate .
       port ( in1, in2, in3 : in bit ;
       out1 : out bit );
end entity and3 ;

architecture dataflow of and3 is
       begin
              out1 <= in1 and in2 and in3 ;
end architecture dataflow ;

entity or4 is    4 input or gate.
       port ( in1, in2, in3, in4 : in bit ;
       out1 : out bit);
end entity or4 ;

architecture dataflow of or4 is
       begin
              out1 <= in1 or in2 or in3 or in4 ;
end architecture dataflow ;

entity inv is    Inverter.
       port ( in1 : in bit ;
```

```
       out1 : out bit);
end entity inv ;

architecture dataflow of inv is
       begin
              out1 <= not in1 ;
end architecture dataflow ;


entity testbench is    Test bench outer entity wrapper.
end entity testbench ;


library work ;
use work . all ;


architecture structural of testbench is    Testbench architecture.
signal b0, b1, b2, b3, zb, yb, xb, wb,a, b, c, d, f : bit;
begin
       d1 : entity test_vector port map(b0, b1, b2, b3);
       d2 : entity inv port map(b0, wb);
       d3 : entity inv port map(b1, xb);
       d4 : entity inv port map(b2, yb);
       d5 : entity inv port map(b3, zb);
       d6 : entity and4 port map(zb, yb, b1, wb, a);
       d7 : entity and4 port map(zb, yb, xb, wb, b);
       d8 : entity and3 port map(zb, yb, b0, c);
       d9 : entity and4 port map(b0, b1, b2, b3, d);
       d10 : entity or4 port map(a, b, c, d, f);
end architecture structural ;
```

При вводе этого кода в текстовый редактор Qucs VHDL текст будет выделен цветом. К сожалению, цветовое кодирование теряется при печати или переносе в текстовый процессор, или компоновщик текста, подобный LaTeX. Структура листинга VHDL следует нормальным соглашениям для базируемой на тексте VHDL симуляции. Все компоненты объектно-архитектурных моделей должны быть определены до ссылок на них из других моделей компонент. Симуляция испытательных стендов должна быть последней объектно-архитектурной моделью в листинге VHDL. В течение VHDL фазы компиляции FreeHDL компилирует компонент объектно-архитектурных моделей в рабочую библиотеку[22]. Эти скомпилированные модели затем становятся доступны для испытательных стендов симуляции через VHDL установку `use`, вставляемую в листинг до установки архитектуры испытательного стенда (testbench architecture statement). Когда VHDL листинг для симуляции был введен в Qucs VHDL редактор кода, нажатием клавиши F2 запускается процесс симуляции. Продолжительность симуляции может быть задана использованием Настроек Документа в выпадающем меню Файл (или нажатием клавиш Ctrl+.). Любые VHDL синтаксические ошибки, или действительные опечатки, записываются в файл и могут быть просмотрены нажатием клавиши F5. Очевидно, что если

22 В большинстве реализаций VHDL рабочая библиотека всегда видна, и не требуется делать ее видимой, используя установки библиотеки и использования. Однако FreeHDL, похоже, нуждается в этих установках в фазе линковки, иначе компилятор VHDL дает сбой.

есть сообщение об ошибке, значит необходимо ее исправить, используя VHDL редактор текста, и перезапустить симуляцию. Типичный TimeList выход для примера 1 показан на рис. 4.11.

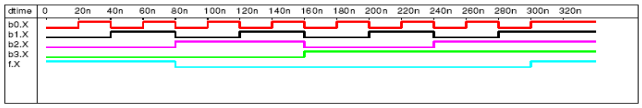


Рис. 4.11. Образец графики симуляции для редактора VHDL примера 1.

- Пример 2: Пример 1VHDL редактора моделировал, используя поток данных VHDL установок. VHDL код второго примера дан в следующем листинге. VHDL стиль, выбранный для моделирования схемы, базируется на потоке данных VHDL, совпадающем с назначенными сигналами. Входные текстовые векторы сгенерированы с использованием простого механизма состояний, а не отдельных установок процесса. Спецификация тест-вектора порта генератора использует в точности один сигнал типа «bit», и может легко вписаться без проблем в другие компоненты, соединенные в диаграмму схемы Qucs. Процедура ввода символов компонент в схему из объектно-архитектурных моделей представлена в следующем разделе этих заметок. В этом примере также продемонстрировано использование бит-векторной конструкции шины. Qucs позволяет использовать бит-векторы, как сигналы или переменные в VHDL моделях, предоставляя все сигналы в установках порта объявления объекта только типа «bit»[23]. Типичный вывод TimeList для примера 2 показан на рис. 4.12.

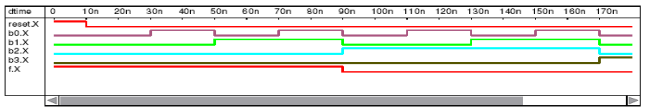


Рис. 4.12. Образец симуляции сигналов для редактора VHDL в примере 2.

23 Это ограничение Qucs 0.0.9 и будет отменено в последующих выпусках пакета. Также заметьте, сигналы типа бит-вектора, которые декларируются в определении архитектуры, перечислены в диалоге графика сигналов TimeList. Однако текстовое сообщение говорит о том, что нет данных результатов, если сделана попытка отобразить их. И еще раз, это ограничение будет устранено в последующих выпусках Qucs.

```
entity test_vector_a is
         port ( RESET, CLOCK : in bit ;
         B0, B1, B2, B3 : out bit);
end entity test_vector_a ;

architecture behavioural of test_vector_a is
signal present_state , next_state : bit_vector (3 downto 0):= ”1111 ”;
begin

p1 : process (CLOCK ) is
       begin
              if (CLOCK’ event and CLOCK= ’1 ’) then
                     present_state <= next_state ;
              end if ;
       end process p1 ;


p2 : process (RESET, present_state ) is
       begin
              if (RESET = ’1’ ) then next_state <= ”1111”;
              end if ;
       case present_state is
              when ”0000” => next_state <= ”0001”;
              when ”0001” => next_state <= ”0010”;
              when ”0010” => next_state <= ”0011”;
              when ”0011” => next_state <= ”0100”;
              when ”0100” => next_state <= ”0101”;
              when ”0101” => next_state <= ”0110”;
              when ”0110” => next_state <= ”0111”;
              when ”0111” => next_state <= ”1000”;
              when ”1000” => next_state <= ”1001”;
              when ”1001” => next_state <= ”1010”;
              when ”1010” => next_state <= ”1011”;
              when ”1011” => next_state <= ”1100”;
              when ”1100” => next_state <= ”1101”;
              when ”1101” => next_state <= ”1110”;
              when ”1110” => next_state <= ”1111”;
              when ”1111” => next_state <= ”0000”;
       end case ;
       B3 <= next_state (3); B2 <= next_state (2);
       B1 <= next_state (1); B0 <= next_state (0);
       end process p2 ;
end architecture behavioural ;


library work ;
use work . all ;


entity testbench is
end entity testbench ;


architecture dataflow of testbench is
signal reset, clk, b0, b1, b2, b3, zb : bit;
signal yb, xb, wb,a, b, c, d, f : bit;
begin
p1 : process is
       begin
```

```
                clk <= '0 '; wait for 10 ns;
                clk <= '1 '; wait for 10 ns;


end process p1 ;


p2 : process is
        begin
                reset <= '1 '; wait for 10 ns;
                reset <= '0 '; wait for 2000 ns ;

end process p2 ;


d1 : entity test_vector_a port map(reset , clk , b0, b1, b2, b3);

   Data flow model of combinational circuit

        wb <= not b0; xb <= not b1; yb <= not b2; zb <= not b3 ;
        a <= (wb and b1 ) and ( yb and zb );
        b <= (wb and xb ) and ( yb and zb );
        c <= b0 and ( yb and zb );
        d <= (b0 and b1 ) and ( b2 and b3 );
        f <=a or b or c or d;
end architecture dataflow ;
```

- Пример 3: Пример 1 VHDL редактора моделирует, используя установки и переменные VHDL процесса.
  VHDL код для третьего примера дан в листинге в конце этого параграфа. В этом примере иллюстрируется использование переменных VHDL. VHDL код для генератора вектора слегка необычен в том, что используется не традиционная разработка двух-процессорных служебных сигналов, предпринимающая и калькуляцию следующих данных состояния, и передачу информации о следующем состоянии в существующее состояние. Это приближение необходимо, поскольку FreeHDL не позволяет использовать разделяемые переменные. И еще раз, в этом примере только одно-битовые данные проходят через установки объекта к тестируемому устройству. Тестируемое устройство представлено таблицей истинности, кодированной в установки процесса. Это не самый элегантный код, но он вполне обеспечивает демонстрацию использования различных VHDL конструкций и типов данных в Qucs цифровой симуляции. Типичный TimeList график для редактора VHDL в примере 3 показан на рис. 4.13. Сравнение трех выходных графиков в примерах для редактора VHDL показывает, что все результаты симуляции очень похожи с некоторыми небольшими отличиями в начальной фазе, следующей за импульсом RESET, изменяющимся с логической «1» на логический «0». Этот, возможно, эффект возникает вследствие разной последовательности инициализации для каждой из тест-векторных моделей.

  Qucs VHDL editor Пример *3*

```
entity test_vector_b is
port ( RESET, CLOCK : in bit ;
         B0, B1, B2, B3 : out bit);
end entity test_vector_b ;


architecture behavioural of test_vector_b is
begin
p1 : process (RESET, CLOCK) is
       variable present_state , next_state :
              bit_vector (3 downto 0):= ”0000 ”;
       begin
              if (RESET = ’1’ ) then next_state := ”0000”;
              elsif (CLOCK’ event and CLOCK= ’1 ’) then
                     present_state := next_state ;
              case present_state is
                     when ”0000 ” => next_state := ”0001”;
                     when ”0001 ” => next_state := ”0010”;
                     when ”0010 ” => next_state := ”0011”;
                     when ”0011 ” => next_state := ”0100”;
                     when ”0100 ” => next_state := ”0101”;
                     when ”0101 ” => next_state := ”0110”;
                     when ”0110 ” => next_state := ”0111”;
                     when ”0111 ” => next_state := ”1000”;
                     when ”1000 ” => next_state := ”1001”;
                     when ”1001 ” => next_state := ”1010”;
                     when ”1010 ” => next_state := ”1011”;
                     when ”1011 ” => next_state := ”1100”;
                     when ”1100 ” => next_state := ”1101”;
                     when ”1101 ” => next_state := ”1110”;
                     when ”1110 ” => next_state := ”1111”;
                     when ”1111 ” => next_state := ”0000”;
              end case ;
              end if ;
       B3 <= next_state (3); B2 <= next_state (2);
       B1 <= next_state (1); B0 <= next_state (0);
end process p1 ;
end architecture behavioural ;


library work ;
use work . all ;


entity testbench is
end entity testbench ;


architecture dataflow of testbench is
signal reset, clk, b0, b1, b2, b3, f : bit;
begin
p1 : process is
       begin
              clk <= ’0 ’; wait for 10 ns;
              clk <= ’1 ’; wait for 10 ns;
end process p1 ;


p2 : process is
begin
       reset <= ’1 ’; wait for 10 ns;
       reset <= ’0 ’; wait for 2000 ns ;
end process p2 ;
```

```
d1 : entity test_vector_b port map(reset , clk , b0, b1, b2, b3);

   Behavioural model of combinational circuit


p3 : process (b3, b2, b1, b0) is
variable SEL : bit_vector (3 downto 0);
begin
       SEL := b3&b2&b1&b0 ;
       if (SEL = "0010 ") then f <= '1 ';
       elsif (SEL = "0000 ") then f <= '1 ';
       elsif (SEL = "1111 ") then f <= '1 ';
       elsif (SEL = "0001 ") then f <= '1 ';
       elsif (SEL = "0011 ") then f <= '1 ';
       else f <= '0 ';
       end if ;
end process p3 ;
end architecture dataflow ;
```

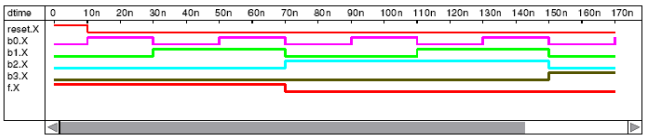


Рис. 4.13. Образец графика симуляции редактора VHDL в примере 3.

4.9.5 Линковка VHDL объектно-архитектурных моделей с символами устройств Qucs схемы

VHDL был изначально разработан, как язык описания оборудования для спецификации цифровых систем. И действительно, многие инженеры все еще предпочитают описывать цифровые системы полностью формулировками VHDL, а не использовать черчение схем. Когда он написан, код VHDL сохраняется как текстовый файл и становится входными данными для пакета VHDL компилятора/симулятора. Через общераспространенные запросы многие цифровые синтезаторы/симуляторы средств CAD[24] начинают включать способность линковки VHDL модельного кода с символами ввода схем. Затем это, конечно, можно использовать в диаграммах схем, как основной вход среды[25] разработки и симуляции цифровых цепей. Qucs выпуска 0.0.9 имеет такую возможность, позволяющую соединять VHDL код моделей с символами схемы. При черчении цифровых схем эти определенные пользователем символы могут смешиваться с предопределенными цифровыми символами Qucs и

24 См., например, XILINX, WebPACK программное обеспечение на http//www.xilinx.com/ise/logic_design_prod/webpack.htm.

25 Пожалуйста, заметьте, что при начале процесса симуляции VHDL чертежи схем конвертируются в VHDL текстовый файл.

другими символами подсхем, определенными пользователем. Процесс линковки кода VHDL с чертежными символами схем Qucs непосредственный и будет проиллюстрирован в этих заметках двумя примерами.

- Пример 4: 4х-битовый тестовый векторный генератор шаблона. Показанное в таблице 4.4 – это листинг VHDL объектно-архитектурной модели 4х-битового двоичного генератора шаблона. VHDL код идентичен тестовому коду вектора, представленному в третьем примере редактора VHDL. После введения VHDL объектно-архитектурного кода модели, используя редактор Qucs для VHDL, окончательный текст сохраняется в файле с подходящим именем и расширением файла vhdl. Qucs затем вписывает модель под категорией проекта VHDL. Просто щелкните по имени модели в категории VHDL левой клавишей мышки, а затем перенесите указатель мышки в подходящее место на схеме, что приведет к тому, что Qucs переместит символ, представляющий модель, на лист с чертежом схемы. Размещение символа в позиции указываемой курсором достигается щелчком левой клавиши мышки. Процедура схожая с той, что используется для выбора и размещения предопределенных символов Qucs на чертеже схемы. Qucs автоматически генерирует символ прямоугольника с именем названной VHDL, имеющий то же самое количество выводов, что и установки порта в установках объекта VHDL модели. Каждый из выводов получает имя, которое соответствует имени в объектных установках. Qucs фиксирует порядок выводов на сгенерированном символе. Похоже, что нет возможности редактировать этот символ. Однако in, out или inout порты символов подсхемы могут быть прикреплены к символам VHDL и редактированный пользователем символ генерируется. Рис. 4.14 показывает сгенерированный Qucs символ VHDL с прикрепленными портами для модели, записанной в таблице 4.4. Редактированный символ для 4х-битового двоичного генератора шаблона показан на рис. 4.15. Заметьте, что на рис. 4.15 порядок расположения выводов был изменен для отражения естественного порядка у устройства с его входными выводами слева и выходными справа. Символы VHDL модели могут также быть генерированы размещением компонента файла VHDL, что находится в обзорном листе в цифровых компонентах на схеме. При редактировании свойства имени файла VHDL для этого устройства на соответствие файлу объектно-архитектурной модели VHDL, Qucs автоматически генерирует символ VHDL. Определите ваш собственный символ, затем продолжите на тот же манер, что описано выше.

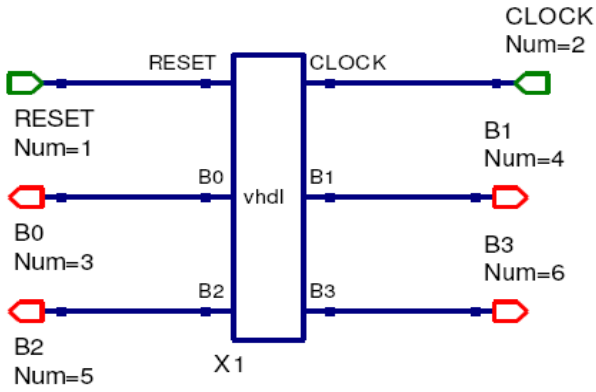


Рис. 4.14. Сгенерированный Qucs символ VHDL с портами подсхемы для тестового генератора шаблона

```
entity patgen_4bit is
port ( RESET, CLOCK : in bit ;
         B0, B1, B2, B3 : out bit);
end entity patgen_4bit ;


architecture behavioural of patgen_4bit is
begin
p1 : process (RESET, CLOCK) is
       variable present_state , next_state :
              bit_vector (3 downto 0):= ”0000”;

       begin
              if (RESET = ’1’ ) then next_state := ”0000”;
              elsif (CLOCK’ event and CLOCK= ’1 ’) then
                     present_state := next_state ;
              case present_state is
                     when ”0000 ” => next_state := ”0001”;
                     when ”0001 ” => next_state := ”0010”;
                     when ”0010 ” => next_state := ”0011”;
                     when ”0011 ” => next_state := ”0100”;
                     when ”0100 ” => next_state := ”0101”;
                     when ”0101 ” => next_state := ”0110”;
                     when ”0110 ” => next_state := ”0111”;
                     when ”0111 ” => next_state := ”1000”;
                     when ”1000 ” => next_state := ”1001”;
                     when ”1001 ” => next_state := ”1010”;
                     when ”1010 ” => next_state := ”1011”;
                     when ”1011 ” => next_state := ”1100”;
                     when ”1100 ” => next_state := ”1101”;
                     when ”1101 ” => next_state := ”1110”;
                     when ”1110 ” => next_state := ”1111”;
                     when ”1111 ” => next_state := ”0000”;
              end case ;
              end if ;
              B3 <= next_state (3); B2 <= next_state (2);
              B1 <= next_state (1); B0 <= next_state (0);
       end process p1 ;
end architecture behavioural ;
```

Таблица 4.4. VHDL код для 4х-битового генератора шаблона

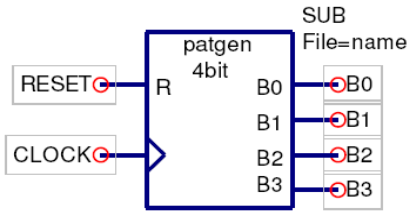


Рис. 4.15. Определенный пользователем символ 4х-битового генератора шаблона

Full adder 1 bit

```
entity fulladder is
port (a, b, cin : in bit ;
         sum, cout : out bit);
end entity fulladder ;
```

```
architecture dataflow of fulladder is

begin
       sum <= (a xor b) xor cin ;
       cout <= (a and b) or (a and cin) or (b and cin );
end architecture dataflow ;
```

Таблица 4.5. VHDL код для 1-битового полного сумматора

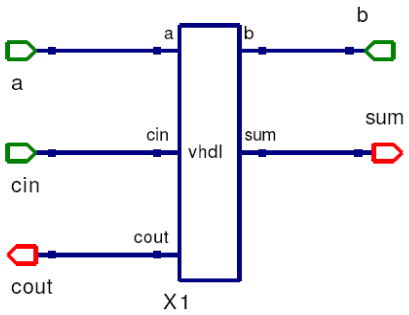


Рис. 4.16. Сгенерированный Qucs символ VHDL с портами подсхемы для одно-битового полного сумматора

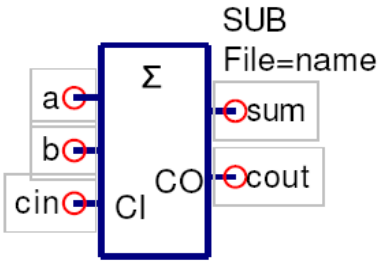


Рис. 4.17. Определенный пользователем символ одно-битового полного сумматора

- Пример 5: 4х-битовый полный сумматор.
  Символы VHDL модели могут комбинироваться либо с символами предопределенных цифровых компонентов Qucs, либо с другими символами подсхем. В данном примере VHDL модель для простого одно-битового полного сумматора соединяется четыре раза последовательно, формируя 4х-битовый полный сумматор. Код VHDL модели для простого одно-битового полного сумматора дан в таблице 4.5. Диаграмма ассоциированного символа для одно-битового полного сумматора показана на рис. 4.16 и 4.17.

  Рис. 4.18 показывает схему простого 4-х битового сумматора. Соответствующий определенный пользователем символ 4-х битового полного сумматора дан на рис. 4.19.

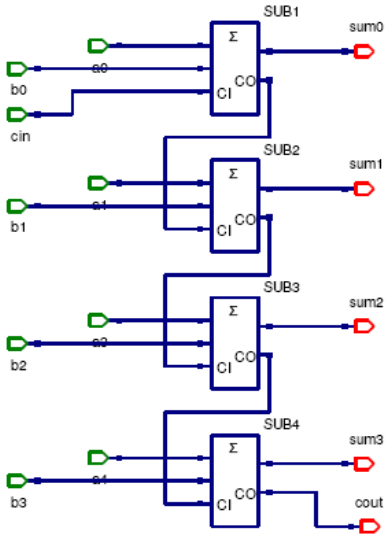


Рис. 4.18. Схема 4х-битового полного сумматора

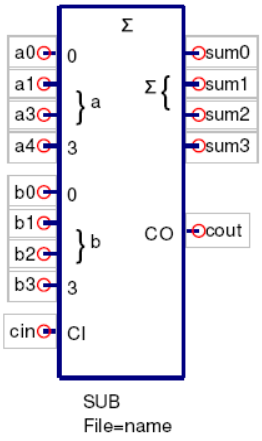


Рис. 4.19. Определенный пользователем символ 4х-битового полного сумматора

4.9.6 Генерация кода VHDL из схемы Qucs

Нажатие клавиши F2 приведет к симуляции Qucs разработки введенной пользователем Qucs. Входные данные для симуляции – это либо текстовый файл VHDL, сохраненный из текстового редактора VHDL, либо файл кода VHDL, сгенерированный Qucs, использовавшим информацию, заложенную в чертеж схемы. В этом разделе заметок руководства представлена большая разработка, а

результирующий код VHDL и результаты симуляции обсуждаются. Пример, выбранный для этой цели – это 4 на 4 бита комбинационный цифровой множитель. Оба, 4-х битовый генератор шаблона и 4-х битовый полный сумматор, обсуждавшиеся в последнем разделе, формируют часть разработки ядра 4-х битового множителя и его ассоциированного испытательного стенда. Таблица 4.6 показывает таблицу умножения для 4 бита на 4 бита комбинационного двоичного множителя. Входы устройства – это двоичные биты a3 a2 a1 a0 и b3 b2 b1 b0. Устройство множителя 4 на 4 требует 16 вентилей and (для генерации производимых множителем термов), трех 4х-битовых полных сумматора (для суммирования выходных r термов) и двух 4х-битовых генераторов шаблонов для проверки 256 возможных состояний входов. Выход множителя представлен в таблице 4.6 как r7 r6 r5 r4 r3 r2 r1 и r0. Схема для множителя 4 на 4 и испытательный стенд даны на рис. 4.20.

				b3	b2	b1	b0
				a3	a2	a1	a0
				a0b3	a0b2	a0b1	a0b0
			a1b3	a1b2	a1b1	a1b0	
		a2b3	a2b2	a2b1	a2b0		
	a3b3	a3b2	a3b1	a3b0			
r7	r6	r5	r4	r3	r2	r1	r0

Таблица 4.6. Таблица для 4 на 4 комбинационного множителя

Код VHDL для этого примера представлен в следующем листинге. Этот листинг был сгенерирован Qucs[26]. Небольшая часть графика TimeList для цифрового множителя показана на рис. 4.21. В момент 1.74 микросекунд вход a – «0101», вход b – «0111» и выход r – «00100011», что соответствует десятичному 35. Несколько произвольных проверок результатов симуляции показали, что множитель 4 на 4 работает правильно. Заметьте, что код VHDL, сгенерированный Qucs для 4х-битового множителя, не содержит никаких временных задержек распространения данных. Это должно быть добавлено в вентили and, если необходимо. Однако на данном этапе разработки цифровой симуляции Qucs передача временных данных, и других параметров, от символов устройств, генерируемых из моделей VHDL, совершенно не реализован. Использование родовых типов (generics) VHDL очевидный путь, как это должно быть сделано. Родовые типы допустимы, конечно, в базируемых на тексте VHDL симуляциях.

26 Некоторые читатели отметят, что именование схем для внутренних сетей сигналов отличается в листинге множителя VHDL по сравнению с листингом VHDL в первой версии этих заметок. В конечной фазе разработки 0.0.9 соглашение об именовании, применяемое для Qucs, было изменено, чтобы придать больше гибкости структуре.

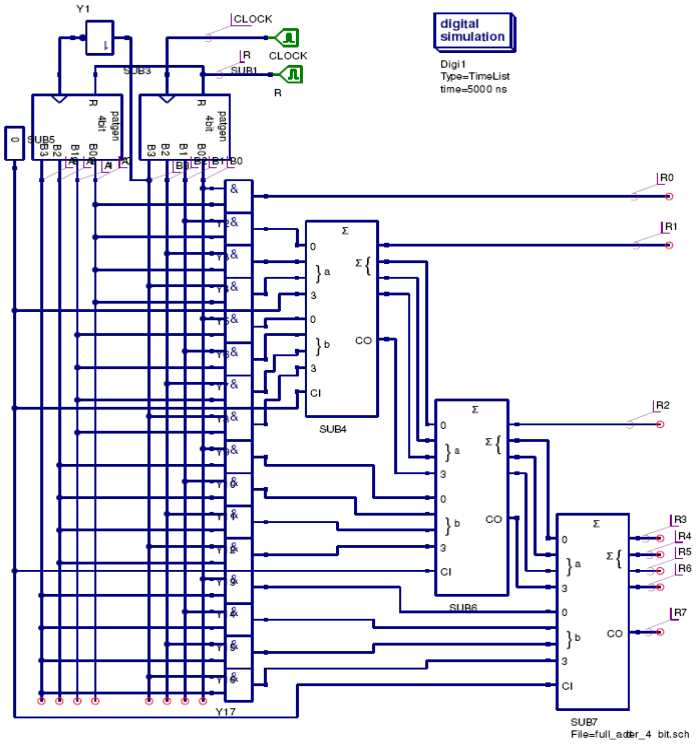


Рис. 4.20. 4 на 4 бита комбинационный цифровой множитель

```
   Qucs 0.0.9
   /mnt/hda2/vhdl_comp_lib_prj/multiplier_4bx4bit.sch

entity patgen_4bit is
       port ( RESET, CLOCK : in bit ;
                B0, B1, B2, B3 : out bit);
end entity patgen_4bit ;


architecture behavioural of patgen_4bit is
begin
p1 : process (RESET, CLOCK) is
       variable present_state , next_state :
              bit_vector (3 downto 0) := ”0000”;
       begin
              if (RESET = ’1’ ) then next_state := ”0000”;
              elsif (CLOCK’ event and CLOCK= ’1’) then
                     present_state := next_state ;
              case present_state is
                     when ”0000” => next_state := ”0001”;
              when ”0001” => next_state := ”0010”;
              when ”0010” => next_state := ”0011”;
              when ”0011” => next_state := ”0100”;
              when ”0100” => next_state := ”0101”;
              when ”0101” => next_state := ”0110”;
              when ”0110” => next_state := ”0111”;
```

```
                when ”0111” => next_state := ”1000”;
                when ”1000” => next_state := ”1001”;
                when ”1001” => next_state := ”1010”;
                when ”1010” => next_state := ”1011”;
                when ”1011” => next_state := ”1100”;
                when ”1100” => next_state := ”1101”;
                when ”1101” => next_state := ”1110”;
                when ”1110” => next_state := ”1111”;
                when ”1111” => next_state := ”0000”;
        end case ;
        end if ;
        B3 <= next_state (3); B2 <= next_state (2);
        B1 <= next_state (1); B0 <= next_state (0);
end process p1 ;
end architecture behavioural ;

entity Sub_patgen_4bit is
        port (net_net0 : in bit ;
                net_net5 : in bit ;
                net_outnet_net1 : out bit ;
                net_outnet_net3 : out bit ;
                net_outnet_net2 : out bit ;
                net_outnet_net4 : out bit );
end entity ;
use work . all ;
architecture Arch_Sub_patgen_4bit of Sub_patgen_4bit is
        signal net_net1 ,
                net_net2 ,
                net_net3 ,
                net_net4 : bit ;
begin
        net_outnet_net1 <= net_net1 or ’0 ’;
        net_outnet_net2 <= net_net2 or ’0 ’;
        net_outnet_net3 <= net_net3 or ’0 ’;
        net_outnet_net4 <= net_net4 or ’0 ’;
        X1 : entity patgen 4bit port map ( net_net0 , net_net5 ,
                net_net1 , net_net3 , net_net2 , net_net4 );
end architecture ;
```

logic_zero.vhdl

```
entity logic_zero is
  port (Y : out bit);
end entity logic_zero ;


architecture dataflow of logic_zero is
begin
        Y <= ’0 ’;
end architecture dataflow ;

entity Sub_logic_zero is
        port ( net_outnetY : out bit );
end entity ;
use work . all ;
architecture Arch_Sub_logic_zero of Sub_logic_zero is
        signal netY : bit;
begin
        X1 : entity logic_zero port map ( netY ) ;
        net_outnetY <= netY or ’0 ’;
end architecture ;
```

```
   Full adder   1 bit

entity fulladder is
       port (a, b, cin : in bit ;
                sum, cout : out bit);
end entity fulladder ;


architecture dataflow of fulladder is
begin
       sum <= (a xor b) xor cin ;
       cout <= (a and b) or (a and cin) or (b and cin );
end architecture dataflow ;

entity Sub_full_adder_1bit is
       port (net_net0 : in bit ;
       net_net1 : in bit ;
       net_net2 : in bit ;
       net_outnet_net3 : out bit ;
       net_outnet_net4 : out bit );
end entity ;
use work . all ;
architecture Arch Sub_full_adder_1bit of Sub_full_adder_1bit is
       signal net_net3 ,
       net_net4 : bit ;
begin
       X1 : entity fulladder port map ( net_net0 , net_net1 ,
       net_net2 , net_net3 , net_net4 );
       net_outnet_net3 <= net_net3 or '0 ';
       net_outnet_net4 <= net_net4 or '0 ';
end architecture ;

entity Sub_full_adder_4bit is
       port (net_net0 : in bit ;
       net_net1 : in bit ;
       net_net2 : in bit ;
       net_net3 : in bit ;
       net_net4 : in bit ;
       net_net5 : in bit ;
       net_net6 : in bit ;
       net_net13 : in bit ;
       net_net7 : in bit ;
       net_outnet_net8 : out bit ;
       net_outnet_net9 : out bit ;
       net_outnet_net10 : out bit ;
       net_outnet_net11 : out bit ;
       net_outnet_net12 : out bit );
end entity ;

use work . all ;
architecture Arch_Sub_full_adder_4bit of Sub_full_adder_4bit is
       signal net_net14 ,
       net_net15 ,
       net_net16 ,
       net_net8 ,
       net_net9 ,
       net_net10 ,
       net_net11 ,
       net_net12 : bit ;
begin
       net_outnet_net8 <= net_net8 or '0 ' ;
       net_outnet_net9 <= net_net9 or '0 ' ;
       net_outnet_net10 <= net_net10 or '0' ;
```

```
       net_outnet_net11 <= net_net11 or '0' ;
       net_outnet_net12 <= net_net12 or '0' ;
SUB4 : entity Sub_full_adder_1bit port map ( net_net3 , net_net13 ,
       net_net14 , net_net11 , net_net12 ) ;
SUB3 : entity Sub_full_adder_1bit port map ( net_net2 , net_net6 ,
       net_net15 , net_net10 , net_net14 ) ;
SUB2 : entity Sub_full_adder_1bit port map ( net_net1 , net_net5 ,
       net_net16 , net_net9 , net_net15 ) ;
SUB1 : entity Sub_full_adder_1bit port map ( net_net0 , net_net4 ,
       net_net7 , net_net8 , net_net16 ) ;
end architecture ;


entity TestBench is
end entity ;
use work . all ;
architecture Arch_TestBench of TestBench is
  signal netA0 , netA1 , netA2 , netA3 , netR , netB0 ,
       netB1 , netB2 , netB3 , netR0 , netR1 , netR2 ,
       netR3 , netR4 , netR5 , netR6 , netR7 , netCLOCK,
       net_net0 , net_net1 , net_net2 , net_net3 , net_net4 ,
       net_net5 , net_net6 , net_net7 , net_net8 , net_net9 ,
       net_net10 , net_net11 , net_net12 , net_net13 , net_net14 ,
       net_net15 , net_net16 , net_net17 , net_net18 , net_net19 ,
       net_net20 , net_net21 , net_net22 , net_net23 ,
       net_net24 : bit ;
begin
SUB3 : entity Sub_patgen_4bit port map (netR , net_net0 ,
                                        netA0, netA1, netA2, netA3);
SUB1 : entity Sub_patgen_4bit port map (netR , netCLOCK,
                                        netB0, netB1, netB2, netB3);

R: process
begin
       netR <= '1 '; wait for 10 ns;
       netR <= '0 '; wait for 2000 ns ;
end process ;

CLOCK: process
begin
       netCLOCK <= '0 '; wait for 10 ns;
       netCLOCK <= '1 '; wait for 10 ns;
end process ;

net_net0 <= not netB3 ;
netR0 <= netA0 and netB0 ;
net_net1 <= netA0 and netB1 ;
net_net2 <= netA0 and netB2 ;
net_net3 <= netA0 and netB3 ;
SUB5 : entity Sub_logic_zero port map (net_net4 );
net_net5 <= netA1 and netB0 ;
net_net6 <= netA1 and netB1 ;
net_net7 <= netA1 and netB2 ;
net_net8 <= netA1 and netB3 ;
net_net9 <= netA2 and netB0 ;
net_net10 <= netA2 and netB1 ;
net_net11 <= netA2 and netB2 ;
net_net12 <= netA2 and netB3 ;
SUB4 : entity Sub_full_adder_4bit port map ( net_net1 , net_net2 ,
              net_net3 , net_net4 , net_net5 , net_net6 , net_net7 ,
              net_net8 , net_net4 , netR1 , net_net13 , net_net14 ,
              net_net15 , net_net16 );

SUB6 : entity Sub_full_adder_4bit port map ( net_net13 , net_net14 ,
```

```
        net_net15 , net_net16 , net_net9 , net_net10 , net_net11 ,
        net_net12 , net_net4 , netR2 , net_net17 , net_net18 ,
        net_net19 , net_net20 );
net_net21 <= netA3 and netB0 ;
net_net22 <= netA3 and netB1 ;
net_net23 <= netA3 and netB2 ;
net_net24 <= netA3 and netB3 ;
SUB7 : entity Sub_full_adder_4bit port map ( net_net17 , net_net18 ,
               net_net19 , net_net20 , net_net21 , net_net22 ,
               net_net23 , net_net24 , net_net4 , netR3 , netR4 ,
               netR5 , netR6 , netR7 ) ;
end architecture ;
```

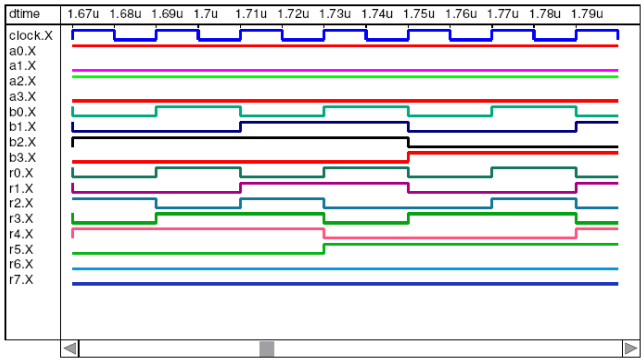


Рис. 4.21. Часть графика вывода TimeList 4 на 4 бита комбинационного цифрового множителя

4.10 Обновление номер два: Сентябрь 2006

Обновление номер два в этом руководстве последовательно сообщает о главных изменениях, которые имели место в Qucs цифровой симуляции с момента первого обновления размещенного на Qucs Web-сайте около трех месяцев назад. За этот период было реализовано некоторое количество важных, и очень критических, расширений. Предыдущие выпуски концентрировались на стабильности фундаментальной базы для цифровой симуляции цепей, используя язык VHDL. Первым переносчиком для представления сигналов цепей были VHDL типы сигналов bit и bit-vector. Следующий выпуск Qucs (версия 0.0.10) и FreeHDL (версия 0.0.3) расширили допустимые типы сигналов включением IEEE std_logic_1164 девяти-уровневой логики, целых и действительных. Читатели оценят эти изменения – результат большого кропотливого труда группы Qucs. Они должны рассматриваться как очень большой прогресс, поскольку не все возможности, предлагаемые реализацией FreeHDL языка VHDL, в настоящий момент доступны через ввод схем в Qucs и текстовый файл VHDL подпрограмм симуляции. Хотя значительное количество тестов имело место, похоже, что программные ошибки выйдут на свет, как только больше пользователей Qucs попробует новые возможности. Если вы обнаружите ошибки, пожалуйста, сообщите о них, разместив замечания на Qucs Web-

сайте. Дополнение новых типов сигналов в цифровой симулятор Qucs воздействовало на все этапы пути симуляции, от ввода схемы до графического вывода и табулирования входных и выходных сигналов. Следовательно, хотя это может показаться совершенно неправильно при первой реализации, необходимые изменения для приспосабливания новых типов сигналов есть в отчетах состояния дел по результатам симуляции пакета Qucs. В выпуске 0.0.10 не было сделано попыток добавить новые типы сигналов к части ввода схемы пакета Qucs[27]. Новая работа по цифровым разделам пакета Qucs была сконцентрирована на (1) усовершенствовании ввода в язык VHDL, используя Qucs с цветовым маркированием VHDL редактор текста[28], (2) модификациях к FreeHDL, что позволяет очистить интерфейс между Qucs и FreeHDL, (3) поднять конверсию данных результатов симуляции от формата дампа изменяемых значений FreeHDL до естественного формата Qucs, и (4) главные изменения в подпрограммах сообщения результатов, которые доступны из Qucs диалога иконки диаграмм. Детальный список программных изменений и зафиксированных ошибок может быть найден в лог-файлах изменений Qucs и FreeHDL.

4.10.1 Симуляция кода VHDL с использованием Qucs и FreeHDL.

Диаграмма потока на рис. 4.10 показывает отношение между Qucs и FreeHDL, и последовательность, которая имеет место в процессе симуляции цифровой цепи. Эта потоковая диаграмма, однако, не охватывает деталей этапа конвертирования (1) VHDL кода цепи в машинный код программы симуляции, и (2) выходных результатов симуляции в формат, который может быть нарисован или изображен в табличном виде Qucs. Это проиллюстрировано в диаграмме потоков, представленной на рис. 4.22. Скрипт командного языка управляет каждой стадией qucsdigi в этой последовательности. Базовое понимание процесса использования Qucs и FreeHDL необходимо, если пользователи программного обеспечения способны написать значимый код VHDL и симулировать его, используя два пакета. Код VHDL либо сгенерированный из диаграммы схемы автоматически Qucs, либо при использовании тестового редактора Qucs VHDL. Использование программы ввода схемы было описано в обновлении один этих заметок к руководству. Однако некоторые из читателей, возможно, обнаружат, что включенный в VHDL код, сгенерированный Qucs, ссылается на библиотеки VHDL. Язык VHDL использует библиотеки для предоставления возможностей, которые не специфицированы в базовом определении языка, но широко используются всеми системами обработки языка; две такие библиотеки – это STD и IEEE. Когда симулируются цифровые цепи, базовые знания

27 Дополнительные новые типы сигналов для ввода схемы в Qucs есть в списке to-do.

28 Было устранено некоторое количество ошибок редактора, и теперь пользователи могут сами определить цветовую схему для различных классов зарезервированных слов и типов данных VHDL.

структуры задачи симуляции, и как это использует VHDL библиотеки – существенно. Это подразумевает, что пользователи программного обеспечения Qucs/FreeHDL должны понимать значение того, как система компилирует и симулирует задачу VHDL симуляции цепи. Когда код VHDL симуляции был введен через текстовый редактор VHDL, щелчок по клавише симуляции Qucs запускает командный скрипт (shell script) qucsdigi, выполняющий последовательность, показанную на рис. 4.22[29]. Программа freeehdl-v2cc конвертирует код VHDL в функции C++. Последние затем компилируются вместе с основной функцией C++. Следующий этап в последовательности – это линковка скомпилированного объектного кода с объектным кодом из любой ссылки на элемент в предопределенных VHDL библиотеках для создания выполняемой программы цифровой симуляции. Она же, в свою очередь, запускается Qucs, выводя набор результатов симуляции в формате дампа изменения значений (VCD)[30]. Окончательно, программа, называемая qucsconv, конвертирует VCD результаты симуляции в формат естественных данных Qucs готовыми для пост-процесса в виде графической или табличной диаграммы Qucs.

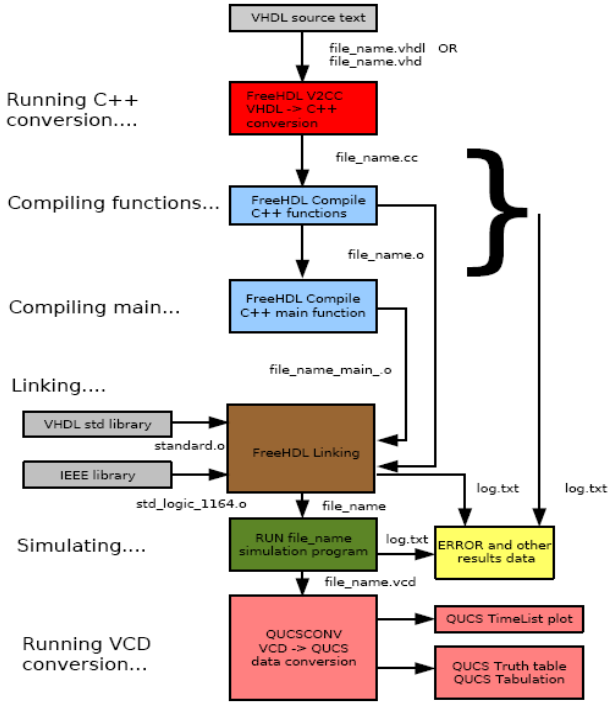


Рис. 4.22. Детальная диаграмма потока, показывающая процесс компиляции и симуляции результатов VHDL кода.

29 Для корректной работы пакета FreeHDL директория, где установлено программное обеспечение, должна быть включена в командный PATH, из которого запускается Qucs.

30 Язык дампа измененных значений был первоначально разработан как формат обмена графикой симуляции для Verilog HDL. Спецификация формата VCD может быть найдена на http://wwwee.eng.hawaii.edu/msmith/ASICs/HTML/Verilog/LRM/HTML/15/ch15.2.htm

4.10.2 VHDL предопределенные пакеты и библиотеки.

Все системы обработки языка VHDL предоставляют предопределенный пакет VHDL, называемый стандартным. Этот пакет определяет множество фундаментальных типов данных VHDL, например, bit, character, integer и real. Предопределенные типы, подтипы и другие функции в стандартном пакете сохраняются в библиотеке, называемой STD. FreeHDL версия библиотеки STD включает дополнительный VHDL пакет, названный textio, который используется для ввода и вывода сигнальных данных из файла и в файл. Вторая библиотека, названная IEEE, определяет (1) многозначные логические сигналы, определенные девятью по-разному кодированными значениями, делающими их удобными для моделирования цифровых цепей, которые составлены из компонент разных технологий, (2) подтипы логических сигналов и (3) большой набор полезных функций, процедур и перезагружаемых операторов. FreeHDL версия библиотеки IEEE состоит из следующих пакетов:

1. std_logic_1164
2. numeric_bit
3. math_real
4. numeric_std
5. std_logic_arith
6. std_logic_unsigned
7. vital_timing

Еще одна библиотека всегда определена системой обработки кода VHDL и называется рабочей (work library). Эта библиотека поддерживает компилированные пользователем VHDL единицы разработки объектов/архитектур.

4.10.3 Структура кода симуляции VHDL.

В большинстве базовых форм VHDL код симуляции цепи структурирован, как объектно-архитектурный испытательный стенд (test bench), который включает тестовую входную информацию сигнала[31]. Пример контура базового формата

```
entity testbench is
   entity body statements
end entity testbench ;


architecture behavioural of testbench is
   architecture body statements
end architecture behavioural ;
```

31 Тестовые сигналы часто называют test_vectors.

Типы данных VHDL, функции и операторы в стандартном пакете всегда видимы для VHDL test bench кода, и добавлять явные ссылки на их использование не нужно. Однако, если объектно-архитектурный испытательный стенд использует типы данных или другие элементы, определенные в других библиотеках, например, std_logic тип в библиотеке IEEE, тогда ссылка на него должна быть добавлена перед каждой объектно-архитектурной парой, где он используется. Ссылки на библиотеки используются VHDL library и use установками. Пример, показывающий, как эти установки применяются, обрисован в следующем сегменте кода VHDL:

```
library ieee ;
use ieee . std_logic_1164 . all ;


entity testbench is
   entity body statements
end entity testbench ;


architecture behavioural of testbench is
   architecture body statements
end architecture behavioural ;
```

Здесь слова кода VHDL означают, что все элементы в специфицированной библиотеке должны быть доступны для использования в последующих объект/архитектура парах, testbench в примере выше. Если должно быть использовано более одной библиотеки, тогда установка library/use нужна для каждой библиотечной ссылки. Более полная программа симуляции цепи VHDL состоит более, чем из одной пары объект/архитектура. В подобных случаях испытательный стенд цепи, с его сигналами тест-векторов, должен быть последним вводом в программу. Пример более сложной структуры программы VHDL

```
library ieee ;
use ieee . std_logic_1164 . all ;

entity comp1 is
   entity body statements
end entity comp1 ;


architecture behavioural of comp1 is
   architecture body statements
end architecture behavioural ;

library ieee ;
use ieee . std_logic_1164 . all ;

entity comp2 is
   entity body statements
end entity comp2 ;


architecture behavioural of comp2 is
```

```
   architecture body statements
end architecture behavioural ;


library ieee ;
use ieee . std_logic_1164 . all ;

use work . all ;


entity testbench is
   entity body statements
end entity testbench ;


architecture behavioural of testbench is
   architecture body statements
end architecture behavioural ;
```

В процессе конвертации кода VHDL в машинный код программы симуляции каждая пара объект/архитектура перед финальным вводом испытательного стенда, компилируется как отдельная единица разработки и сохраняется в рабочей библиотеке[32]. На скомпилированные единицы разработки, поддерживаемые в рабочей библиотеке, могут быть ссылки в других моделях объект/архитектура, задаваемые VHDL установкой `use work.all`[33], вставляемой в код симуляции VHDL перед каждой установкой объект/архитектура, где на них ссылаются.

4.10.4 VHDL типы данных.

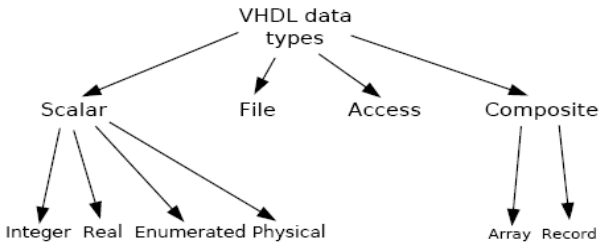


Рис. 4.23. VHDL типы данных

Диаграмма на рис. 4.23 показывает разные типы данных, доступные в языке VHDL. FreeHDL реализует все эти типы данных. В практической симуляции цепей разные типы данных VHDL обычно используются для спецификации (1) сигналов, (2)

32 Пара объект/архитектура testbench также, конечно, компилируется, но эта единица разработки единственная, которая запускается как выполняемая программа симуляции.

33 Ссылки на индивидуальные элементы также допустимы вставкой, например, `use.work.comb1; use.work.comb2;` в код VHDL.

переменных и (3) констант[34]. В процессе симуляции Qucs/FreeHDL автоматически сохраняют эти значения целых, действительных и перечисляемых бит сигналов, как развитие симуляции во времени. Более того, `bit_vector` и типы сигналов IEEE, включающие `std_logic_vector`, также сохраняются. Сигналы этих типов затем становятся доступны для вывода и табулирования с использованием Timing (временных), Truth table (таблиц истинности), Tabular (табличных) и Cartesian (декартовых) выходных диаграмм. Выделенные элементы в определенных пользователем составных сигналах, те что сохраняются в массивах, например[35], могут быть назначены базовым типам сигналов, а затем отображены[36]. Пример того, как это делается, дан в следующих разделах этих обновленных заметок руководства. Отметьте, что значения переменных и констант не записываются во время симуляции.

4.10.5 Пример VHDL симуляции, использующей сигналы integer.

Следующий код VHDL демонстрирует, как данные целого типа могут быть использованы для представления сигналов. В этом примере сигналы A, B изменяют состояние фронта тактовых импульсов `clk`. Код дополнительно проверяет целые сигналы и константы, используя арифметические операторы, определенные в библиотеке STD[37]. Результаты этой симуляции показаны на рис. 4.24.

```
   A very basic test of data type integer
entity testbench is
end entity testbench ;

architecture behavioural of testbench is
signal A, B, C : integer := 0;
signal clk : bit;
begin
p0 : process is    Generate clock signal
       begin
              clk <= '0 '; wait for 10 ns;
              clk <= '1 '; wait for 10 ns;
       end process p0 ;

p1 : process (clk) is
       begin
              if ( clk ' event and clk = '1 ') then
```

34 Тип файла, конечно, разный, в зависимости от того, используется ли он для сохранения тест-векторов, данных компонент, как содержимое ROM, и выходных результатов симуляции.

35 Пожалуйста, заметьте, что эти типы сигналов, базируемые на составных типах записей, будут, возможно, вызывать сбой цикла симуляции Qucs, работа с этими типами данных была добавлена в список to-do.

36 Qucs/FreeHDL также автоматически собирает данные графики для составных сигналов, базируемых на массивах бит и IEEE типах сигналов. Однако в случае больших массивов нужно при выводе или табулировании делать это непосредственно, поскольку содержимое массива выводится каждый раз при отображении сигнала.

37 Спецификация FreeHDL библиотеки STD может быть найдена в текстовом файле freehdl-0.0.3/std/standard.vhdl.

```
                    A <=A+ 1;
                    B <=B+ 2;
                end if ;
        end process p1 ;
        C <=A+B ;
end architecture behavioural ;
```

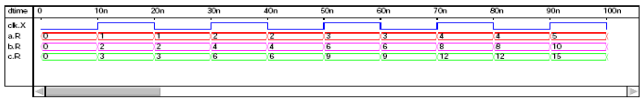


Рис. 4.24. Выходные результаты простого примера test bench, использующего сигналы integer

4.10.6 Многозначная логика.

Хотя сигналы типов bit и bit-vector широко применяются при симулировании цифровых систем, одним из самых слабых их мест является факт, что трудно представлять системы сигнальных шин, используя только кодирование сигналов логическим «0» и логической «1». Более того, схемы, где обнаруживаются соединения сигналов в шину, часто приводят к сбою симуляции. Пакет `IEEE std_logic_1164` устраняет эти ограничения посредством введения системы многозначной логики, которая определяет девять разных логических значений для представления типов сигналов и силы сигналов. Не только проблема шинных соединений разрешается с помощью логических функций, но система многозначной логики позволяет также конструировать устройства из разных производственных технологий, симулируемые одновременно, обеспечивая отражение процессом симуляции реальной практики разработки схем. Следующие два примера симуляции вводят систему девяти логических значений и демонстрируют ее использование в разработке цифровой шинной системы. Сигналы типа real также введены для показа их представления в Qucs. Приведенное ниже – это код VHDL для базовой симуляции, которая генерирует набор IEEE std_logic, integer и real сигналов. Рис. 4.25 иллюстрирует, как Qucs Timing диаграмма отображает разные типы сигналов. Секция табличных результатов также дана на рис. 4.26.

```
library ieee ;
use ieee . std_logic_1164 . all ;

entity testbench is
end entity testbench ;


architecture behavioural of testbench is
signal clk : bit;
signal bv1 : bit_vector (8 downto 0);
signal stdl1 : std_logic vector (8 downto 0);
signal INT1 : integer := 0;
signal INT2 : integer := 99;
signal R1 : real := 0.33;
signal R2 : real := 99.0;
signal R3 : real := 0.0;
```

```
signal R4 : real := 0.0;
begin
p0 : process is
        begin
                clk <= '0 '; wait for 10 ns;
                clk <= '1 '; wait for 10 ns;
        end process p0 ;

p1 : process (clk) is
        variable v1 : integer := 0;
        begin
        if ( clk ' event and clk = '1') then
                v1 := v1+1;
                case v1 is
                when 1=> bv1 <= "000000000"; stdl1 <= "000000000";
                when 2=> bv1 <= "000000001"; stdl1 <= "000000001";
                when 3=> bv1 <= "000000011"; stdl1 <= "00000001X";
                when 4=> bv1 <= "000000111"; stdl1 <= "0000001XZ";
                when 5=> bv1 <= "000001111"; stdl1 <= "000001XZU";
                when 6=> bv1 <= "000011111"; stdl1 <= "00001XZUW";
                when 7=> bv1 <= "000111111"; stdl1 <= "0001XZUWL";
                when 8=> bv1 <= "001111111"; stdl1 <= "001XZUWLH";
                when 9=> bv1 <= "111111111"; stdl1 <= "01XZUWLH ";
                when others => v1 := 0;
                end case ;
        end if ;
end process p1 ;
p3 : process (clk) is
        begin
                if ( clk ' event and clk ='1') then
                        INT1 <= INT1 + 1;
                        INT2 <= INT2  20;
                end if ;


        if (INT1 >= 9) then
                INT1 <= 0;
                INT2 <= 99;
        end if ;
end process p3 ;


p4 : process (clk) is
        Variable V2 : real;
        begin
                if ( clk ' event and clk ='1') then
                        R1 <= R1 + 1.0;
                        R2 <= R2  20.0;
                        R3 <= R1#R2 ;
                        R4 <= R2/(R1+0.0001);
                end if ;


        if (R1 >= 20.0) then
                R1 <= 0.0;
                R2 <= 99.0;
        end if ;
end process p4 ;
end architecture behavioural ;
```

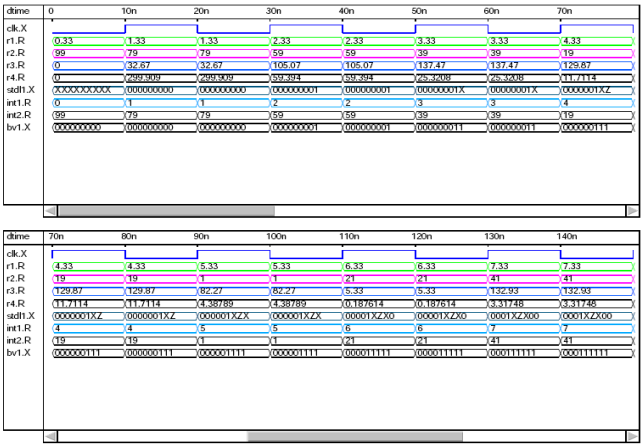


Рис. 4.25. Выходные результаты, иллюстрирующие TimeList представление сигналов

VCD вывод графики (waveform) заменяет стандартную кодировку цифровых сигналов на четыре логических уровня. Это «0», «1», «Z» (высокий импеданс) и «X» (неизвестный). Таблица 4.7 перечисляет, как девять `ieee.std_logic` уровней сигнала представлены в формате VCD. Пока VCD стандарт проверяется, пакет Qucs/FreeHDL ограничивает отображение выходных данных симуляции базовыми «0», «1», «Z» и «X» сигналами. Следующий пример показывает, как IEEE `std_logic` тип сигнала может быть использован для симуляции шинной логики. Демонстрация была сделана простой, чтобы сократить код VHDL. Фрагмент кода симулирует два три-стабильных буфера, которые передают выходы на шинные драйверы, чьи выходы соединены в общую сигнальную шину. Шинные драйверы обеспечивают разделение выходов три-стабильных буферов перед комбинированием на общей линии шины. Это позволяет выходным сигналам буферов и комбинированному сигналу вычерчиваться раздельно. Результирующие графики ясно показывают решение функции `std_logic` в операции, см. рис. 4.27. Обратите внимание

VHDL signal levels VCD

’0’ Форсированный логический 0 ’0’
’1’ Форсированная логическая 1 ’1’
’X’ Форсированный неизвестный ’X’
’Z’ Высокий импеданс ’Z’
’U’ Неинициализированный ’X’
’W’ Слабый неизвестный ’0’
’L’ Слабый логический 0 ’0’
’H” Слабая логическая 1 ’1’
’-’ Неважно ’X’

Таблица 4.7. IEEE многозначная логика и VCD представление

на эффект 7 ns задержки на рисунке графика и использование VHDL общей установки для задания инверсного значения задержки устройства.

```
         Demonstration of a simple bus structure using
         the IEEE std_logic data type.
      library ieee ;
      use ieee . std_logic_1164 . all ;

      entity buf is
      generic (delay : time := 0 ns );
      port (in1 , control : in std_logic ;
              out1 : out std_logic );
      end entity buf ;
      architecture behavioural of buf is
      begin
      p0 : process (in1 , control ) is
             begin
                    if ( control = '1 ') then out1 <= in1 after delay ;
                    else out1 <= 'Z';

                    end if ;
             end process p0 ;
      end architecture behavioural ;


      library ieee ;
      use ieee . std_logic_1164 . all ;

      entity invert is
      generic (delay : time := 0 ns );
port ( in1 : in std_logic ;
              out1 : out std_logic);
      end entity invert ;


      architecture behavioural of invert is
      begin
             out1 <= not in1 after delay ;
      end architecture behavioural ;


      library ieee ;
      use ieee . std_logic_1164 . all ;


      entity buf2 is
      port ( in1 : in std_logic ;
              out1 : out std_logic);
      end entity buf2 ;


      architecture dataflow of buf2 is
      begin
             out1 <= in1 ;
      end architecture dataflow ;


      library ieee ;
      use ieee . std_logic_1164 . all ;

      use work . all ;
```

```
entity testbench is
end entity testbench ;


architecture structural of testbench is
signal data_in_1, data_in_2 : std_logic;
signal data_out_1 , data_out_2 : std_logic ;
signal data_control , control_buf1 : std_logic ;
signal result : std_logic ;


begin
p0 : process is
       begin
              data_in_1 <= '0 '; wait for 5 ns;
              data_in_1 <= '1 '; wait for 5 ns;
end process p0 ;

data_in_2 <= not data_in_1;

p1 : process is
       begin
              data_control <= '1 '; wait for 40 ns;
              data_control <= '0 '; wait for 40 ns;
end process p1 ;

c1g1 : entity buf port map( in1 => data_in_1, control => data_control,
       out1 => data_out_1 );
c1g2 : entity invert generic map ( delay => 7 ns)
       port map( in1 => data_control , out1 => control_buf1 );
c1g3 : entity buf port map( in1 => data_in_2, control => control_buf1,
       out1 => data_out_2 );
c1g4 : entity buf2 port map( in1 => data_out_1, out1 => result );
c1g5 : entity buf2 port map( in1 => data_out_2, out1 => result );


end architecture structural ;
```

dtime	clk.X	int1.R	int2.R	r1.R	r2.R	r3.R	r4.R	bv1.X	stdl1.X
0	0	0	99	0.33	99	0	0	000000000	XXXXXXXXX
1e-8	1	1	79	1.33	79	32.7	300	000000000	000000000
2e-8	0	1	79	1.33	79	32.7	300	000000000	000000000
3e-8	1	2	59	2.33	59	105	59.4	000000001	000000001
4e-8	0	2	59	2.33	59	105	59.4	000000001	000000001
5e-8	1	3	39	3.33	39	137	25.3	000000011	00000001X
6e-8	0	3	39	3.33	39	137	25.3	000000011	00000001X
7e-8	1	4	19	4.33	19	130	11.7	000000111	0000001XZ
8e-8	0	4	19	4.33	19	130	11.7	000000111	0000001XZ
9e-8	1	5	-1	5.33	-1	82.3	4.39	000001111	000001XZX
1e-7	0	5	-1	5.33	-1	82.3	4.39	000001111	000001XZX
1.1e-7	1	6	-21	6.33	-21	-5.33	-0.188	000011111	00001XZX0
1.2e-7	0	6	-21	6.33	-21	-5.33	-0.188	000011111	00001XZX0
1.3e-7	1	7	-41	7.33	-41	-133	-3.32	000111111	0001XZX00
1.4e-7	0	7	-41	7.33	-41	-133	-3.32	000111111	0001XZX00
1.5e-7	1	8	-61	8.33	-61	-301	-5.59	001111111	001XZX001
1.6e-7	0	8	-61	8.33	-61	-301	-5.59	001111111	001XZX001
1.7e-7	1	9	-81	9.33	-81	-508	-7.32	111111111	01XZX001X
1.8e-7	0	0	99	9.33	-81	-508	-7.32	111111111	01XZX001X
1.9e-7	1	1	79	10.3	-101	-756	-8.68	111111111	01XZX001X

	clk.X	int1.R	int2.R	r1.R	r2.R	r3.R	r4.R	bv1.X	stdl1.X
00000	0	0	99	0.33	99	0	0	000000000	XXXXXXXXX
00001	1	1	79	1.33	79	32.67	299.909	000000000	000000000
00010	0	1	79	1.33	79	32.67	299.909	000000000	000000000
00011	1	2	59	2.33	59	105.07	59.394	000000001	000000001
00100	0	2	59	2.33	59	105.07	59.394	000000001	000000001
00101	1	3	39	3.33	39	137.47	25.3208	000000011	00000001X
00110	0	3	39	3.33	39	137.47	25.3208	000000011	00000001X
00111	1	4	19	4.33	19	129.87	11.7114	000000111	0000001XZ
01000	0	4	19	4.33	19	129.87	11.7114	000000111	0000001XZ
01001	1	5	1	5.33	1	82.27	4.38789	000001111	000001XZX
01010	0	5	1	5.33	1	82.27	4.38789	000001111	000001XZX
01011	1	6	21	6.33	21	5.33	0.187614	000011111	00001XZX0
01100	0	6	21	6.33	21	5.33	0.187614	000011111	00001XZX0
01101	1	7	41	7.33	41	132.93	3.31748	000111111	0001XZX00
01110	0	7	41	7.33	41	132.93	3.31748	000111111	0001XZX00
01111	1	8	61	8.33	61	300.53	5.59338	001111111	001XZX001
10000	0	8	61	8.33	61	300.53	5.59338	001111111	001XZX001
10001	1	9	81	9.33	81	508.13	7.32284	111111111	01XZX001X
10010	0	0	99	9.33	81	508.13	7.32284	111111111	01XZX001X
10011	1	1	79	10.33	101	755.73	8.68158	111111111	01XZX001X
10100	0	1	79	10.33	101	755.73	8.68158	111111111	01XZX001X

Рис. 4.26. Выходные результаты, иллюстрирующие табличное представление сигналов

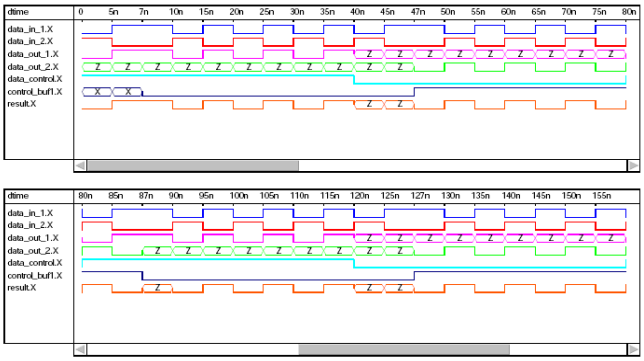


Рис. 4.27. Графика сигналов для простого примера шины

4.10.7 Запуск отладчика VHDL кода симуляции.

Язык VHDL имеет встроенные возможности, позволяющие отлаживать код VHDL во время симуляции. В этом разделе представлены зарезервированные слова утверждений, сообщений и предупреждений, и они используются как отладочная

помощь, объясняющая попутно детали примера разработки. В предыдущем обновлении цифрового руководства была представлена структурированная разработка 4х-битового цифрового перемножителя, как пример применения трассировки цифрового симулятора ввода схем в Qucs. Следующий пример расширяет предыдущий умножитель до 16 бит. Однако структурный уровень большего умножителя становится очень детальным и его разработка может быть склонна к ошибкам. Для демонстрации мощности VHDL 16 битовый перемножитель был реконструирован на функциональном уровне. Блок-диаграмма его симулирующего испытательного стенда дана на рис. 4.28: вначале тактовый генератор стробирует устройство генератора данных, которое генерирует последовательность целых чисел. Это все конвертируется в 16 bit_vectors и прикладывается к устройству 16-битового перемножителя, как входы x и y; потом 16-битовый умножитель при обнаружении изменений на входах x или y конвертирует эти сигналы из 16 bit_vectors в integer, перемножает их и, наконец, конвертирует целый результат в 32 bit_vector выход Res_bit. Хотя стандартная библиотека STD определяет арифметические операции для целых, она не предоставляет функции для конвертации целых в бит-векторы или обратных операций. Следующий VHDL листинг дает полную программу испытательного стенда симуляции для 16-битового перемножителя, включая необходимые функции преобразования данных. VHDL отладочный код или код сообщений использует зарезервированные слова утверждений, отчетов и предостережений, добавленных к коду архитектуры data_generator и functional_multiplier. В процессе симуляции эти тестовые строки и время симуляции, когда активируются, записываются в Qucs `log.txt` файл, давая запись трассировки активности симуляции. В случаях обнаружения ошибок на уровне предостерегающего сбоя, симуляция прерывается. FreeHDL позволяет VHDL сообщать установки без сопровождающих утверждающих установок[38]. Типичная диаграмма Timing для этой разработки показана на рис. 4.29.

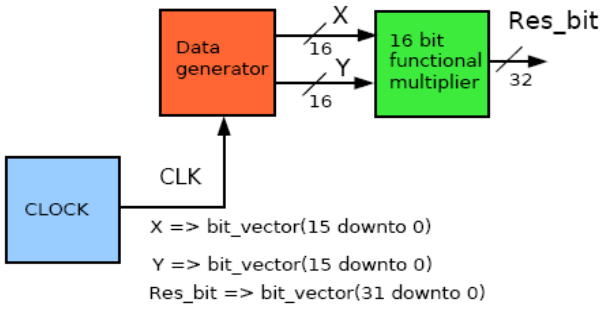


Рис. 4.28. Блок-диаграмма 16-битового функционального перемножителя

38 Одно из изменений в 1993 ревизии IEEE VHDL 1076-1987 стандарта было разрешение установок сообщений без предшествующей подтверждающей клаузулы утверждения. FreeHDL пытается подчиниться ревизии 1993.

16 bit digital multiplier Пример.
Simulation trace using assert , report and severity statements.

```
entity clock is
port ( clk : out bit );
end entity clock ;

architecture behavioural of clock is
begin
p0 : process is
       begin
              clk <= '0' ; wait for 10 ns;
              clk <= '1' ; wait for 10 ns;
       end process p0 ;
end architecture behavioural ;


entity data_generator is
port ( clk : in bit ;
         x, y : out bit_vector (15 downto 0));
end entity data_generator ;


architecture behavioural of data_generator is
type mem_array_16 is array (1 to 8) of integer ;
signal count : integer := 0;

function integer_to_vector_16(int no : integer) return bit_vector is
variable ni : integer ;
variable return_value : bit_vector (15 downto 0);
begin
 assert ( ni < 0)
  report "Function integer_to_vector_32 : integer number must be >= 0"
  severity failure ;
 ni := int no;
 for i in return_value ' Reverse_Range loop
       if ( (ni mod 2 )=1 ) then return_value(i) := '1';
       else return_value(i) := '0';
       end if ;
 ni := ni/2;
 end loop ;
 return return_value ;
 end integer_to_vector_16 ;


begin
p1 : process (clk) is
       variable xi : mem_array_16 := (1, 2, 3, 4, 5, 6, 7, 8);
       variable yi : mem_array_16 := (2, 4, 6, 8, 10, 12, 14, 16);
       variable xh, yh : integer ;
       variable counti : integer ;
       begin
              counti := count+1;
                     if ( counti > 8) then
                            counti := 1;
                     end if ;
              xh := xi(counti);
              yh := yi(counti);
              x <= integer_to_vector_16(xh);
              y <= integer_to_vector_16(yh);
              count <= counti ;
              report "In process p1.data_generator.";
```

```
       end process p1 ;
end architecture behavioural ;


entity functional_multiplier is
port ( x, y: in bit_vector (15 downto 0);
         res_bit : out bit_vector (31 downto 0));
end entity functional_multiplier ;


architecture behavioural of functional_multiplier is

function vector_to_integer (v1 : bit_vector ) return integer is
variable return_value : integer :=0;
alias v2 : bit_vector (v1 ' length  1 downto 0) is v1 ;
       begin
       for i in v2 ' high downto 1 loop
              if (v2(i) = '1') then
                     return_value := (return_value+1)#2;
              else
                     return_value := return_value   2;
              end if ;
       end loop ;
       if v2(0) = '1' then return_value:= return_value+1;
       end if ;
return return_value ;
end vector_to_integer ;


function integer_to_vector_32(int no : integer) return bit_vector is
variable ni : integer ;
variable value : bit_vector (31 downto 0);
begin
 assert ( ni < 0)
 report "Function integer_to_vector_32 : integer number must be >= 0"
 severity failure ;
 ni := int no;
 for i in 0 to 31 loop
       if ( (ni mod 2 )=1 ) then value(i) := '1';
       else value(i) := '0';
       end if ;
       if ni > 0 then ni := ni/2;
       else ni := (ni  1)/2;
       end if ;
 end loop ;
return value ;
end integer_to_vector_32 ;


begin
p0 : process (x,y) is
       variable xi , yi , prod mult : integer ;
       begin
              xi := vector_to_integer(x);
              yi := vector_to_integer(y);
              prod_mult := xi   yi ;
              res_bit <= integer_to_vector_32(prod mult);
              report "In process p1.functional multiplier";
       end process p0 ;
end architecture behavioural ;


entity test2_vhdl_1 is
end entity test2_vhdl_1 ;
```

```
architecture behavioural of test2_vhdl_1 is
signal clk : bit;
signal x, y : bit_vector (15 downto 0);
signal res_bit : bit_vector (31 downto 0);

begin
       d1 : entity work. clock port map ( clk );
       d2 : entity work.data_generator port map(clk , x, y);
       d3 : entity work. functional multiplier port map ( x, y,
res_bit );
end architecture behavioural ;
```

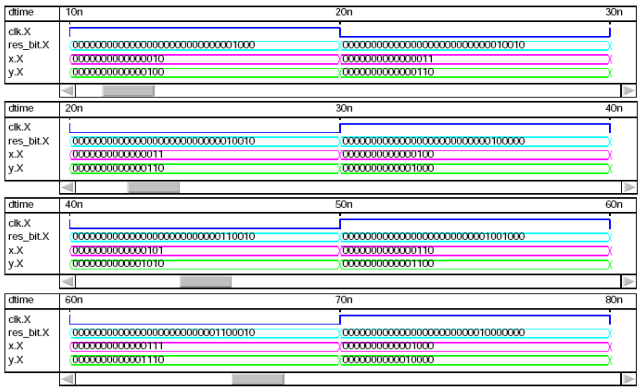


Рис. 4.29. Типичная временная диаграмма для 16-битового функционального умножителя

Более развитые выходные сообщения отладки или таблицы результатов могут быть записаны в файл сообщений Qucs `log.txt` с использованием поддержки предопределенных данных подпрограммами в STD библиотеке пакета textio[39]. Этот пакет содержит функции для чтения и записи STD типов данных из и в файлы[40]. Следующий сегмент кода VHDL показывает, как простая таблица результатов может быть записана в файл `log.txt`. Таблица результатов показана в таблице 4.8.

```
   Test textio package.

library STD;
use STD. textio . all ;


entity Qucs_write_test is
end entity Qucs_write_test ;
```

39 Спецификация для FreeHDL пакета textio может быть найдена в текстовом файле freehdl-0.0.3/std/textio.vhdl.

40 VHDL допускает чтение и запись данных в стандартные потоки ввода и вывода и в определенные пользователем файлы. В настоящее время только запись данных в файл `log.txt` и чтение данных из определенных пользователем файлов были проверены. Пожалуйста, отметьте, что использование пакета `textio` – это весьма рискованная возможность программного обеспечения Qucs/FreeHDL и, возможно, не свободная от ошибок.

```
architecture behavioural of Qucs_write_test is
begin
write_test : process is
        variable input_line , output_line : line ;
        variable int1 : integer := 10 ;
        begin
                write ( output_line , string ' (" ")) ;
                write line ( output , output_line) ;
                write (output_line , string ' (" String  > log.txt ")) ;
                write_line ( output , output_line) ;

        test_L1 : for i c in 1 to 5 loop
                int1 := int1 + 1 ;
                write (output_line , string ' ( " int1 = ")) ;
                write (output_line , int1) ;
                write (output_line , string ' ("int1 ^2 = ")) ;
                write (output_line , int1   int1) ;
                write_line ( output , output_line) ;
                end loop test_L1 ;
        report "Finished test for loop . " ;
        end process write_test ;
end architecture behavioural ;
```

```
Output:
__________
Starting new simulation on Thu 24. Aug 2006 at 13:10:56
running C++ conversion... done.
compiling functions... done.
compiling main... done.
linking... done.
simulating...
Output to STD output -> log.txt
int1 = 11 int1^2 = 121
int1 = 12 int1^2 = 144
int1 = 13 int1^2 = 169
int1 = 14 int1^2 = 196
int1 = 15 int1^2 = 225
0 fs + 0d: NOTE: Finished test for loop.
running VCD conversion... done.
Simulation ended on Thu 24. Aug 2006 at 13:10:57
Ready.
Errors:
________
```

Таблица 4.8: Log.txt файл, показывающий табличный вывод результатов

4.10.8 Тестирование цифровых систем с использованием **test_vector,** сохранных на диске.

В стремлении с моей стороны рассмотреть все новые возможности, представленные в предыдущих секциях этого обновления, заключительный пример, демонстрирующий то, как тест-векторы, сохраненные на диске в виде текстового файла, могут читаться программой симуляции, а затем передаваться на входы цифровой системы при тестировании. Код этого примера дан в следующем листинге:

```
   Testing digital circuits using test_vectors
   stored as a text file on disk.


entity comb1 is
port (a, b, c, d : in bit ;
       y: out bit);
end entity comb1 ;

architecture dataflow of comb1 is
begin
       y <= (a nand b) or (c and d);
end architecture dataflow ;


library STD;
use STD. textio . all ;

entity testbench is
end entity testbench ;


architecture behavioural of testbench is
signal clock : bit ;
signal v1, v2, v3, v4, y_out : bit;
type array_list is array (1 to 20) of bit ;
signal v1sd , v2sd , v3sd , v4sd : array_list ;


Procedure store_data ( variable number : out integer ) is
variable d1, d2, d3, d4 : bit;
variable in_line , out_line : line ;
variable i : integer ;
variable my_string : string(1 to 20):= cr & "Constrained string" & cr;
file infile : text open read_mode is "/mnt/hda2/qucs
                                          0.0.10f/test1_data";
       begin
              report my_string ;
              i := 1;
              while not ( endfile(infile) ) loop
              readline ( infile , in_line );
              read(in_line , d4);
              read(in_line ,d3);
              read(in_line ,d2);
              read(in_line ,d1);
              v1sd( i ) <= d1;
              v2sd( i ) <= d2;
```

```
                v3sd( i ) <= d3;
                v4sd( i ) <= d4;
                report "In file read loop.";
                i := i +1;
                if (i > 20) then exit ;
                end if ;
                number:= i ;
        end loop ;
end procedure store_data ;


begin
p0 : process is    Generate a clock signal .
        begin
                clock <= '1 '; wait for 10 ns;
                clock <= '0 '; wait for 10 ns;
        end process p0 ;


g0 : entity work . comb1 port map (v1, v2, v3, v4, y out);

p1 : process is    Read test_vectors from disk and
                   apply data to circuit inputs .
variable no reads : integer ;
variable in_line , out_line : line ;
        begin
                store_data(no_reads);
                write(out_line ,string '("count = ") );
                write(out_line , no_reads 1);
                writeline (output , out_line );

                for k in 1 to no_reads 1 loop    Count up .
                wait until ( clock ' event and clock ='1 ');
                v1 <= v1sd(k);
                v2 <= v2sd(k);
                v3 <= v3sd(k);
                v4 <= v4sd(k);
                write(out_line , string '("Time = "),left , 8 );
                write(out_line , now, right , 10);
                write(out_line , string '("test_vectors  > "),right,20);
                write(out_line , v4, left , 2 );
                write(out_line , v3, left , 2 );
                write(out_line , v2, left , 2);
                write(out_line , v1, left , 2);
                write(out_line , string '("k = "), right , 10 );
                write(out_line ,k);
                writeline (output , out_line );
                wait until ( clock ' event and clock ='0 ');
                end loop ;

                for k in no_reads 1 downto 1 loop    Count down .
                wait until ( clock ' event and clock ='1 ');
                v1 <= v1sd(k);
                v2 <= v2sd(k);
                v3 <= v3sd(k);
                v4 <= v4sd(k);
                write(out_line , string '("Time = "),left , 8 );
                write(out_line , now, right , 10);
                write(out_line , string '("test_vectors  > "),right,20);
                write(out_line , v4, left , 2 );
                write(out_line , v3, left , 2 );
                write(out_line , v2, left , 2);
                write(out_line , v1, left , 2);
                write(out_line , string '("k = "), right , 10 );
```

```
                write(out_line ,k);
                writeline (output , out_line );
                wait until ( clock ' event and clock ='0 ');
                end loop ;
        wait ;
end process p1 ;
end architecture behavioural ;
```

Хотя листинг выше относительно короткий, тщательное изучение его содержания должно позволить читателю идентифицировать множество новых возможностей Qucs/FreeHDL, представленных ранее. Кроме того, в некоторых разделах код иллюстрирует дополнительные возможности, которые подойдут пользователям Qucs/FreeHDL уже хорошо знакомым с языком VHDL. Они перечислены ниже:

- VHDL код симулирует представление простой цепи комбинационной логики, названной comb1: есть четыре входа (a, b, c, d) типа bit и один выход (y) типа bit[41].
- Симулируемый `testbench` состоит из двух процессов: `process p0` генерирует тактовые сигналы с периодом в 20 ns; `process p1` вводит тестовые данные, содержащиеся в файле `test1_data` [42], и сохраняется в четырех сигнальных массивах (v1sd, v2sd, v3sd и v4sd), используя эти данные для входов цепи при тестировании по фронту тактовых импульсов. Заметьте, что `process p1` выполняется только однажды за счет установки `wait` в конце.
- Реализация компонента `comb1` включена в `testbench` архитектуру. Отметьте использвание VHDL **`entity`** `work.comb1` конструкции – это альтернатива для **`use`** `work.all;`
- Поддержка данных `test_vector` в файле `test_data` прочитывается процедурой `store_data`, возвращающей число линий данных читаемое в переменную. Файловая поддержка, включающая чтение данных с диска, гарантирована с предопределенными программами в пакете `textio`.
- Первая установка сообщения в процедуре `store_data` записывает строку `my_string` в файл `log.txt`. `My_string` – это пример VHDL ограниченного строкового типа, состоящего из непечатаемых управляющих символов[43], связанных с печатаемыми символами.
- Два цикла применяются в `process p1` для передачи сигнала `test_vectors` на вход `comb1`: первый цикл считает вверх от одного, а второй вниз от числа строк `test_vectors`, читаемых процедурой `store_data`, эффективно генерируя `test_vectors` путем, схожим со счетчиком генератора шаблона. Отметьте, что эти данные сигнала применяются к цепи при тестировании по

41 Тип bit был выбран для этого примера вместо одного из сигналов типа IEEE, поскольку пакет `textio` не поддерживает IEEE типы многозначной логики.

42 Я использовал версию Knoppix Linux/GNU операционной системы для всей работы по проекту Qucs. Абсолютное положение файла тестовых данных будет зависеть от того, где были установлены Qucs и FreeHDL, и расположения рабочих файлов.

43 Тип символов в стандартном списке пакета с двух-буквенным кодом используется VHDL для представления непечатаемых управляющих символов.

фронту тактового сигнала, и что примененная последовательность вектора сигнала действительно поднимается до фантазии VHDL программиста.

- Установка `write` в `process p1` для циклов демонстрирует форматированную версию установки `write` в `textio`. Это великолепно помогает в установках таблиц результатов. Таблица 4.9 дает типичное содержание `log.txt` для тестовой симуляции `comb1`.
- В `process p1` сигналы v1, v2, v3 и v4 назначаются индексированным значениям из (тип array_list) v1sd, v2sd, v3sd и v4sd сигналов. В процессе симуляции Qucs/FreeHDL сохраняет значения сигнала как продвижение симуляции. Следовательно, теоретически возможно отображать и стандартный, и композитный типы сигнала. Типичный графический вывод сигналов v1, v2, v3, v4 и y_out дан на рис. 4.30. Рис. 4.31 иллюстрирует графический вывод композитных сигналов v1sd, v2sd, v3sd и v4sd. На рис. 4.31 каждая группа нарисована по мягким изменениям тактовых фронтов идентичных групп значений; каждый вертикальный набор бит представлен значениями бит для единственной строки в файле `test1_data`. Сравните отображенные на рис. 4.31 значения с содержанием файла `test1_data`, показанным на рис. 4.32. Как отмечено раньше, некоторый присмотр нужен при черчении или табулировании композитных сигналов, особенно, когда размер массива большой; массив размером около 50 становится трудным для вычерчивания на экране с нормальным разрешением. В подобных случаях лучше отрезать часть массива и назначить требуемые значения сигналу, который может быть легче отображен.

```
Output :

Starting new simulation on Fri 25 . Aug 2006 at 14 : 35 : 48
running C++ conversion ... done .
compiling functions ... done .
compiling main ... done .
linking ... done .
simulating ...
0 fs +0d : NOTE:
Constrained string
0 fs +0d : NOTE: In file read loop .
.
0 fs +0d : NOTE: In file read loop .
count = 16
Time = 0 ns test_vectors  > 0000 k = 1
Time = 20 ns test_vectors  > 0000 k = 2
Time = 40 ns test_vectors  > 0001 k = 3
Time = 60 ns test_vectors  > 0010 k = 4
.
Time = 200 ns test_vectors  > 1001 k = 11
Time = 220 ns test_vectors  > 1010 k = 12
Time = 240 ns test_vectors  > 1011 k = 13
Time = 260 ns test_vectors  > 1100 k = 14
Time = 280 ns test_vectors  > 1101 k = 15
Time = 300 ns test_vectors  > 1110 k = 16
Time = 320 ns test_vectors  > 1111 k = 16
Time = 340 ns test_vectors  > 1111 k = 15
```

```
Time = 360 ns test_vectors  > 1110 k = 14
Time = 380 ns test_vectors  > 1101 k = 13
Time = 400 ns test_vectors  > 1100 k = 12
.
Time = 560 ns test_vectors  > 0100 k = 4
Time = 580 ns test_vectors  > 0011 k = 3
running VCD conversion ... done .
Simulation ended on Fri 25 . Aug 2006 at 14 : 35 : 50
Ready .
Errors :
```

Таблица 4.9. Редактированная версия табличного вывода результатов, записанных в файл `log.txt`

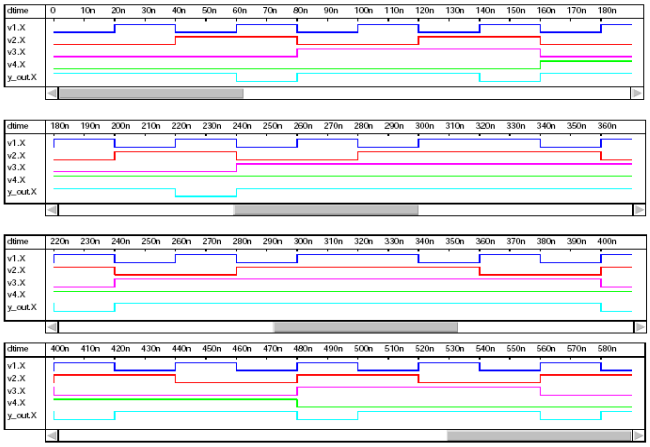


Рис. 4.30. Типичная временная диаграмма для `comb1` симуляции

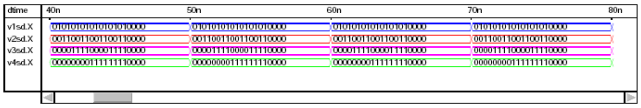


Рис. 4.31. Типичная временная диаграмма для композитных сигналов v1sd, v2sd, v3sd и v4sd

```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

Рис. 4.32. `Comb1` симуляции `test_vectors`

4.11 Окончание заметок

Qucs 0.0.8 добавил цифровую симуляцию к выразительному списку возможностей уже доступных в пакете Qucs. Выпуск 0.0.8 сделал значительный шаг вперед в развитии проекта Qucs. Тот факт, что в первой версии были ошибки цифровой симуляции, не вызывает удивления из-за сложности программного обеспечения. Выпуск 0.0.9 проделал большой путь по устранению большинства этих ошибок. Он также добавил некоторое количество новых возможностей, наиболее примечательными стали новый редактор VHDL и автоматическая генерация символов компонент из созданного вручную кода модели VHDL. Qucs 0.0.10 и FreeHDL 0.0.3 добавили целый ряд новых возможностей в программное обеспечение, особенно важным стало использование IEEE `std_logic_1164` пакета и программ файловой поддержки в пакете `textio`. Мои благодарности Michael Margraf и Stefan Jahn за их поддержку во весь период моего тестирования Qucs VHDL цифровой симуляции и последовавшего написания этих заметок.

5 Область переходных процессов моделей триггеров для симуляции смешанного режима

5.1 Введение

Одной из первых целей проекта Qucs стала разработка универсального симулятора схем, который позволял бы исследовать представление цепи от постоянного тока (DC) до микроволновых частот (microwave). Добавление анализа представления в цифровой области сделало Qucs действительно универсальным симулятором. Qucs 0.0.8 был первым выпуском, включающим цифровую симуляцию. Qucs цифровая симуляция сконцентрировалась на VHDL, используя FreeHDL VHDL компилятор для генерации машинного кода симуляции проверяемых цепей. Выпуск 0.0.8 включал встроенные модели для базовых цифровых вентилей и некоторое количество общих последовательных триггеров (flip-flop). Модели вентилей Qucs могли использоваться и в цифровой симуляции, и в симуляции переходных процессов. К сожалению, модели триггеров допускают только цифровую симуляцию. Текущая версия моделей триггеров Qucs использует VHDL и не предоставляет временную область моделей для симуляции переходных процессов. Это важное упущение, которое ограничивает возможности симулятора Qucs в смешанном режиме. Смешанный режим симуляции – это термин обычно применяемый к описанию симуляции цепей, которые содержат и аналоговые и цифровые компоненты. В реальном мире цепи, конечно, не подразделяются на аккуратные коробочки с метками: аналоговые, S-параметр, цифровые или любые другие физические области. Так что довольно важно, чтобы Qucs моделирование устройств разрабатывалось с поддержкой для цепей, содержащих ряд разных аналоговых и цифровых компонент, симулируемых одновременно. Обычно такие системы симулируются во временной области, используя симуляцию переходных процессов больших сигналов. Представляемые данные существуют и в аналоговых, и в цифровых выражениях в табличной или графической форме. Эти консультативные заметки представляют модели симуляции переходных процессов для триггеров, базирующиеся на структурированных цифровых цепях, описывают их использование и показывают некоторые примеры симуляции, происходящие от практических схем.

5.2 Защелки и триггеры

Последовательные цифровые устройства, в основном известные как триггеры (SR, D, JK и T типов), могут быть разбиты на три главные группы.

- Защелки: простые или стробируемые.
- Переключаемые импульсами триггеры: устройства ведущий-ведомый с или без блокирования данных.
- Переключаемые фронтом триггеры: с переключением передним или задним фронтом.

Поскольку скорость электронных систем увеличивается, популярность переключаемых фронтом триггеров выше, чем более медленных устройств ведущий-ведомый (master slave). Сегодня большинство IC разработок базируется на устройствах D типа, переключаемых фронтом, тогда как прежде это были устройства JK, типа ведущий-ведомый. Наша озабоченность в данный момент касается развития согласованного набора моделей, которые позволяют общим триггерам моделироваться аккуратно и надежно во временной области переходных процессов. В порядке поддержания этих тенденций были выбраны простые стробируемые D-типа или переключаемые фронтом устройства, как фундаментальные строительные блоки для Qucs моделей области переходных процессов. Использование базовых концепций булевой логики прямо указывает на то, что модели JK и T переключаемых фронтом триггеров могут быть производными от моделей D-триггеров.

5.3 Стробируемые D-защелки

Диаграмма цепи для стробируемой D-защелки, собранной из двух-входовых вентилей nand, показана на рис. 5.1[44]. Выходы Q и не Q (QB на рис. 5.1) образованы двумя пересекающимися связанными вентилями nand, соединенными как базовая SR nand защелка. Рис. 5.2 показывает представление характеристик для этой цепи. Они были получены с использованием простой тестовой конфигурации, показанной на рис. 5.3. Цифровые сигналы логической единицы представлены как 1V, а сигналы логического нуля как 0V в области анализа переходных процессов. Задержки распространения через различные вентили цепи могут быть установлены изменением времени задержки для каждого вентиля. Кроссовое соединение вентилей зачастую приводит к сбою симуляции, из-за того, что DC анализ перестает сходиться к стабильному решению в начальный момент симуляции переходного процесса. Есть приближение, помогающее форсировать стабильность DC решения, установить Q и QB в известные состояния, скажем логический 0 и логическую 1, при начале симуляции. В цепях подобных базовой стробируемой D-защелке, показанной на рис. 5.1, где входы асинхронной

44 Richard S. Sandige, Modern Digital Design, 1990, McGraw-Hill International Editions.

установки и сброса не включены, это невозможно. Однако триггер со входами асинхронной установки и сброса позволяет, чтобы состояние триггера было определено в заданный момент симуляции. В примерах, которые последуют, везде, где возможно, состояние защелки или триггера установлено при старте симуляции. В большинстве примеров цепей задержки распространения устройств также были установлены в нуль. Это, следовательно, приводит к тому, что большинство графических выводов показывают функциональные данные иначе, чем аккуратные временные характеристики. Во многих смешанных симуляциях цифровые элементы, присутствующие в разработке, зачастую моделируются как функциональные устройства, чья первая задача – генерировать сигналы нужные для функционирования остальной части схемы. Более детальное освещение эффектов симуляции переходных процессов, включающих временные задержки, появится в следующих разделах этих заметок.

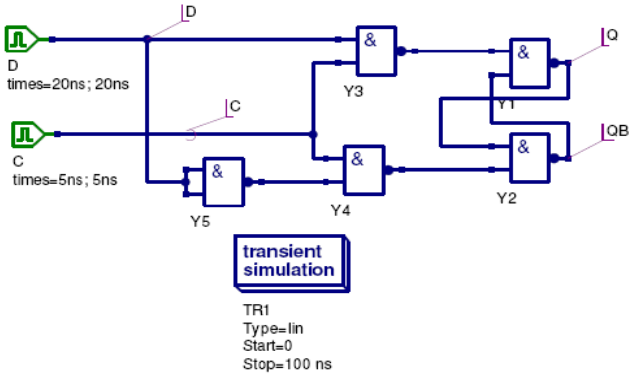


Рис. 5.1. Стробируемая D-защелка с цифровыми генераторами сигналов D и C

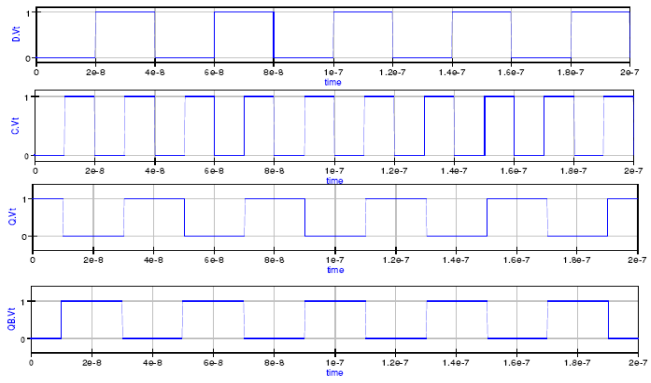


Рис. 5.2. Графический вывод (waveform) симуляции стробируемой D-защелки

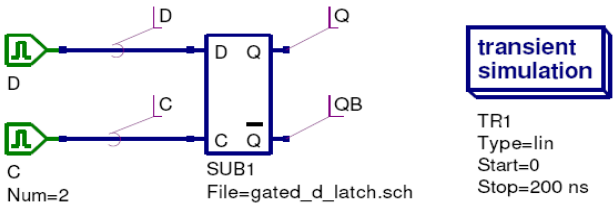


Рис. 5.3. Тестовая схема стробируемой D-защелки

5.4 Тактируемый фронтом D-триггер

Схема для тактируемого положительным фронтом D-триггера показана на рис. 5.4[45]. Асинхронные входы установки (SET) и сброса (RESET) позволяют установить выходы Q и не Q (QB на рис. 5.4) триггера в определенное состояние при старте симуляции. Вентили nand (НЕ-И), формирующие каждую из перекрестно включенных SR-защелок, имеют времена задержки установленными в 0 ns. Управляемое фронтом D устройство – это строительный блок и для JK, и для T типов триггеров. Типичный набор тестовых результатов симуляции переходных процессов для модели D-триггера проиллюстрирован на рис. 5.5. Они были получены с использованием базовой тестовой конфигурации, показанной на рис. 5.6.

45 David A. Hodges and Horace G. Jackson, Analysis and Design of Digital Integrated Circuits, 1998, Second edition, McGraw-Hill Book Company.

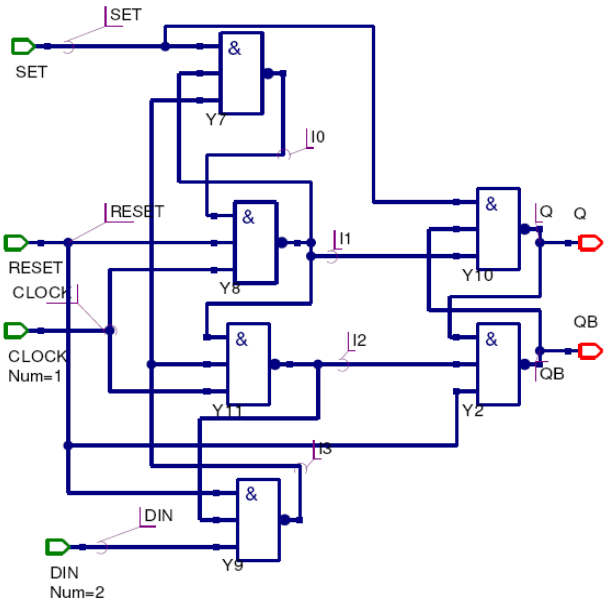


Рис. 5.4. Схема переключаемого положительным фронтом D-триггера

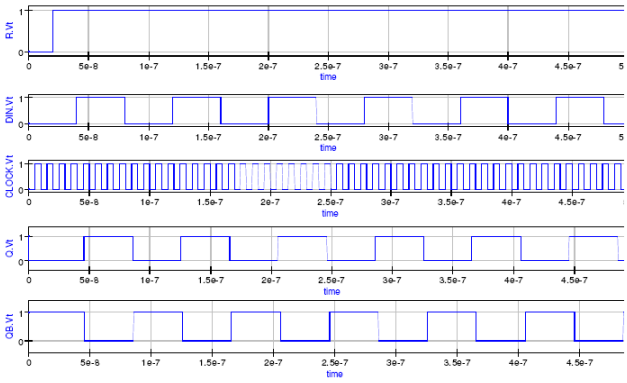


Рис. 5.5. Графический вывод анализа переходного процесса (transient) для схемы рис. 5.6

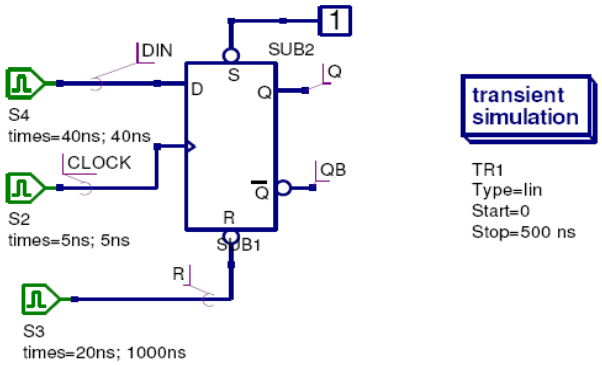


Рис. 5.6. Проверочная схема D-триггера

5.5 Переключаемый фронтом JK-триггер

Переключаемый передним фронтом JK-триггер может быть построен с использованием переключаемого положительным фронтом D-триггера и внешней логики[46]. Внешняя логика генерирует требуемое характеристиками JK-триггера уравнение, задаваемое как

$$Q^{+} = J.\overline{Q} + \overline{K}.Q$$

Где Q, не-Q, J и не-K – это значения текущего состояния сигналов устройства, а Q^{+} значение следующего состояния Q после прихода переднего фронта устройства тактовых импульсов. Схема для переключаемого фронтом триггера показана на рис. 5.7, а типичный набор тестовых диаграмм на рис. 5.8. Они были получены с использованием схемы тестирования, показанной на рис. 5.9.

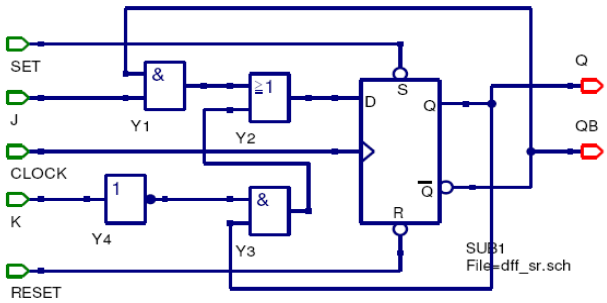
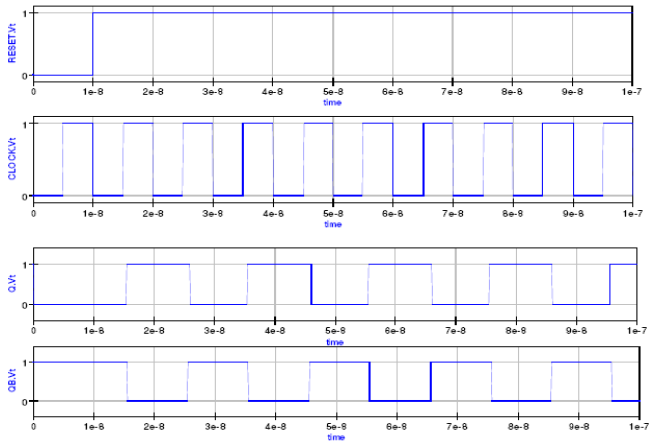


Рис. 5.7. Схема переключаемого передним фронтом JK-триггера



46 M. Morris Mano and Charles R Kime, Logic and Computer Design Fundamentals, 2004, Third edition, Pearson Education International, Prentice Hall

Рис. 5.8. Графический вывод симуляции переходного процесса для схемы рис. 5.9

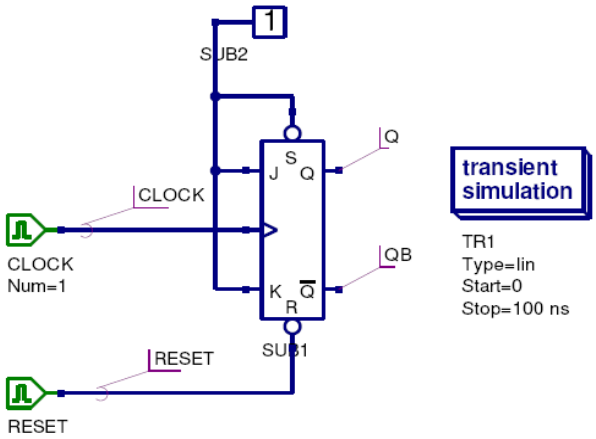


Рис. 5.9. Проверочная схема JK-триггера, показывающая режим переключения

5.6 Переключаемый фронтом T-триггер

Характеристическое уравнение для переключаемого фронтом триггера[47]

$$Q^{+} = T \quad \overline{Q}$$

где символы имеют то же значение, что для JK-триггера. Схема, графический вывод и схема проверки для этого триггера даны на рис. 5.10 – 5.12.

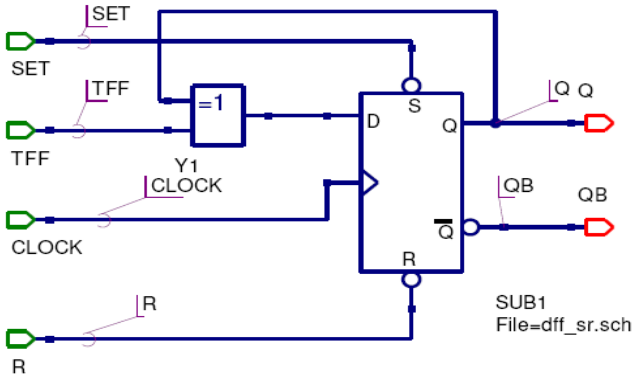


Рис 5.10. Схема переключаемого положительным фронтом Т-триггера

47 См. сноску 45.

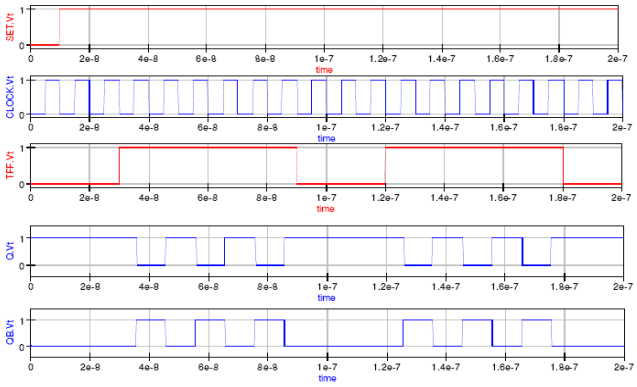


Рис. 5.11. Графический вывод для схемы рис. 5.12

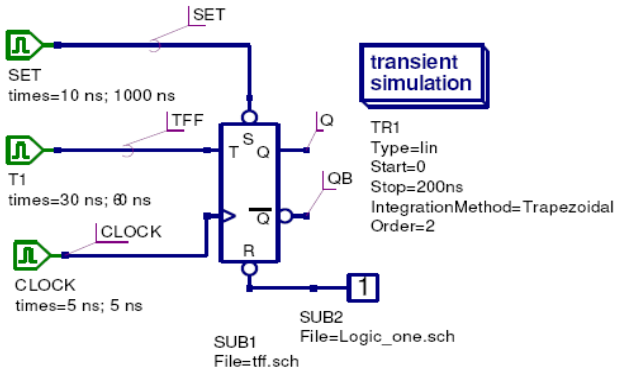


Рис. 5.12. Схема тестирования Т-триггера

5.7 Два примера цифровых цепей

- Синхронный BCD счетчик с наращиванием счета. Рис. 5.13 показывает синхронный счетчик с наращиванием BCD собранный из четырех тактируемых фронтом JK-триггеров, соединенных как переключаемый триггер. Графики входного сигнала и соответствующих выходов счетчика Q0, Q1, Q2 и Q3 показаны на рис. 5.14. Эти результаты симуляции были получены с использованием предопределенного интегрального метода трапеций второго порядка.

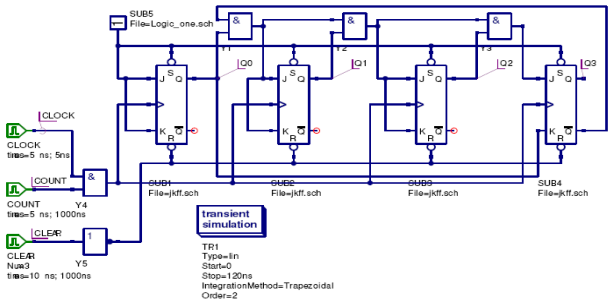


Рис. 5.13. Синхронный прямого счета BCD счетчик

При старте симуляции сигнал CLEAR устанавливается в логическую 1, что заставляет счетчик сброситься к 0000. Аналогично сигнал COUNT должен быть установлен в 1, чтобы счетчик мог начать счет. Заметьте, что счетчик считает от 0 до 9, а затем сбрасывается в 0.

- Простая машина состояний: рис. 5.15 показывает простую последовательную машину состояний со входом X и выходами Y1 и Y2. Выходы синхронизированы с тактовым входом. Уравнения состояния для этого примера

$$J = \overline{X},\ K = 1,\ Y1 = \overline{Q0}.\overline{X},\ Y2 = Q0$$

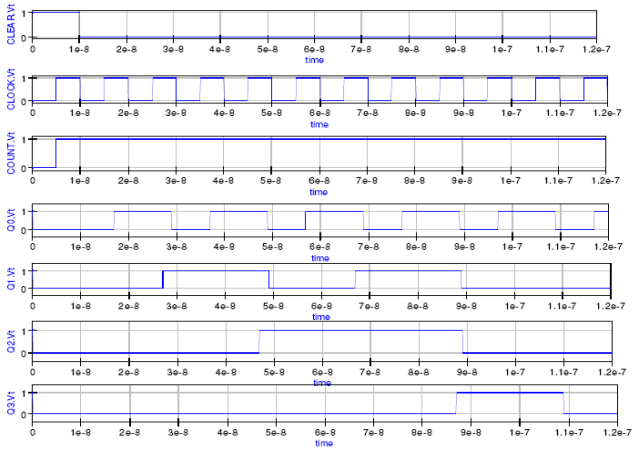


Рис. 5.14. Transient waveforms (графический вывод) для схемы рис. 5.13

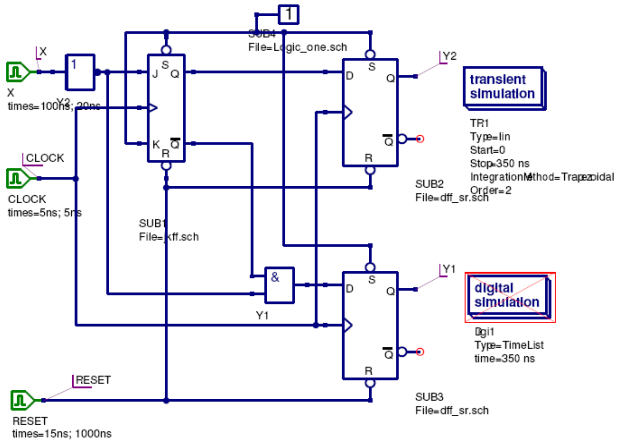


Рис. 5.15. Простая машина состояний

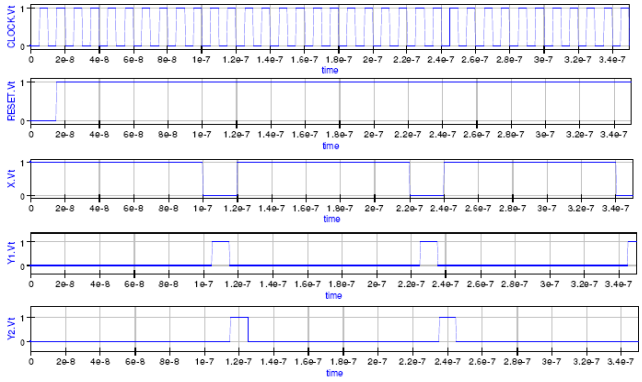


Рис. 5.16. Графический вывод для схемы, показанной на рис. 5.15

5.8 VHDL код для моделей триггеров в области переходных процессов

Хотя первоочередная задача развития моделей триггеров в области переходных процессов ставилась в рамках симуляции смешанных цепей, это ничего не стоило, поскольку модели были сконструированы из примитивов вентилей Qucs, используя приближение снизу-вверх. Qucs также может использовать модели для цифровой симуляции. Кроме того, предоставляя возможность симулировать схемы без содержания каких-либо чисто аналоговых компонент, Qucs будет генерировать VHDL испытательный стенд модели, который описывает функцию и тестовую последовательность для симулируемых схем. Показанное на рис. 5.17 – это временной (timelist) цифровой графический вывод для синхронного прямого счетчика BCD, представленного в предыдущем разделе этих заметок. Листинг 5.1 показывает код VHDL, сгенерированный Qucs для примера синхронного прямого счета BCD счетчика.

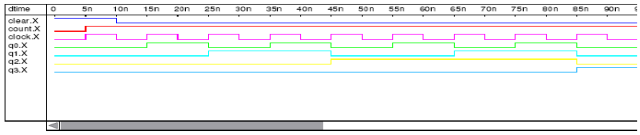


Рис. 5.17. Цифровой график TimeList для схемы рис. 5.13

```
   Qucs 0.0.9
   /mnt/hda2/Digital_Subcircuits_prj/Sync_BCD_counter.sch

entity Sub_Logic_one is
port (nnout_L1 : out bit );
end entity ;
use work . all ;
architecture Arch_Sub_Logic_one of Sub_Logic_one is
signal gnd ,
L1 : bit;
        begin
                gnd <= '0 ' ;
                L1 <= not gnd ;
                nnout_L1 <= L1 or '0 ' ;
end architecture ;

entity Sub_dff_sr is
port (CLOCK: in bit ;
        DIN : in bit ;
        nnout_Q : out bit ;
        nnout_QB : out bit ;
        RESET: in bit ;
        SET: in bit );
end entity ;
use work . all ;
architecture Arch_Sub_dff_sr of Sub_dff_sr is
signal I0 ,
```

```
        I2 ,
        I1 ,
        I3 ,
        QB,
        Q : bit;
        begin
                nnout_QB <= QB or '0 ' ;
                nnout_Q <=Q or '0 ' ;
                I1 <= not (CLOCK and RESET and I0 );
                I3 <= not (DIN and I2 and RESET) ;
                QB <= not (RESET and I2 and Q);
                Q <= not ( I1 and QB and SET ) ;
                I0 <= not ( I3 and I1 and SET ) ;
                I2 <= not (CLOCK and I3 and I1 );
end architecture ;

entity Sub_jkff is
port ( nnnet6 : in bit ;
        nnnet1 : in bit ;
        nnnet8 : in bit ;
        nnout_nnnet3 : out bit ;
        nnout_nnnet7 : out bit ;
        nnnet9 : in bit ;
        nnnet10 : in bit );
end entity ;

use work . all ;
architecture Arch_Sub_jkff of Sub_jkff is
signal nnnet0 ,
        nnnet2 ,
        nnnet4 ,
        nnnet5 ,
        nnnet7 ,
        nnnet3 : bit ;
        begin
                nnnet0 <= not nnnet1 ;
                nnnet2 <= nnnet3 and nnnet0 ;
                nnnet4 <= nnnet2 or nnnet5 ;
                nnnet5 <= nnnet6 and nnnet7 ;
                nnout_nnnet7 <= nnnet7 or '0 ' ;
                nnout_nnnet3 <= nnnet3 or '0 ' ;
SUB1 : entity Sub_dff_sr port map (nnnet8 , nnnet4 , nnnet3 ,
        nnnet7 , nnnet10 , nnnet9 );
end architecture ;

entity TestBench is
end entity ;
use work . all ;

architecture Arch_TestBench of TestBench is
signal CLEAR,
        COUNT,
        CLOCK,
        Q3,
        Q0,
        Q1,
        Q2,
        nnnet0 ,
        nnnet1 ,
        nnnet2 ,
        nnnet3 ,
        nnnet4 ,
        nnnet5 ,
        nnnet6 ,
```

```
        nnnet7 ,
        nnnet8 ,
        nnnet9 : bit ;
        begin
SUB5 : entity Sub_Logic_one port map (nnnet0 );
        nnnet1 <= Q0 and nnnet2 ;
        nnnet3 <= Q1 and nnnet1 ;
        nnnet4 <= Q2 and nnnet3 ;
SUB2 : entity Sub_jkff port map (nnnet1 , nnnet1 , nnnet5 ,
        Q1, nnnet6 , nnnet0 , nnnet7 );
SUB3 : entity Sub_jkff port map (nnnet3 , nnnet3 , nnnet5 ,
        Q2, nnnet8 , nnnet0 , nnnet7 );
SUB1 : entity Sub_jkff port map (nnnet0 , nnnet0 , nnnet5 ,
        Q0, nnnet9 , nnnet0 , nnnet7 );
        nnnet5 <= COUNT and CLOCK;
        nnnet7 <= not CLEAR;

CLEAR: process
        begin
                CLEAR <= '1 ' ; wait for 10 ns;
                CLEAR <= '0 ' ; wait for 1000 ns ;
        end process ;

COUNT: process
        begin
                COUNT <= '0 ' ; wait for 5 ns;
                COUNT <= '1 ' ; wait for 1000 ns ;
        end process ;

CLOCK: process
        begin
                CLOCK <= '0 ' ; wait for 5 ns;
                CLOCK <= '1 ' ; wait for 5 ns;
        end process ;

SUB4 : entity Sub_jkff port map (nnnet4 , Q0, nnnet5 ,
        Q3, nnnet2 , nnnet0 , nnnet7 );
end architecture ;
```

Листинг 5.1. Код VHDL испытательного стенда (testbench) для схемы рис. 5.13

5.9 Генерация библиотеки смешанных цифровых компонент

Возможности проекта Qucs предлагают пользователям простой и удобный подход для разработки библиотек компонентов, которые связываются общей темой. В этих заметках – это модели цифровых компонент для симуляции переходных процессов (transient simulation). Для формирования библиотеки создайте новую папку, в точке дисковой файловой системы, к которой пользователи имеют доступ на чтение/запись, дав ей подходящее имя, например

```
fli_ flop_models_tran_sim_prj
```

Затем поместите в новую папку библиотеки копию каждого файла схемы для моделей триггеров, представленных в этих заметках. Это:

```
dff_sr . sch , jkff . sch, tff . sch , и gated_d_latch . sch
```

Копия схемы для установок узлов к логической единице также потребуется

```
logic_one.sch
```

Эти модели затем становятся доступны для использования в любых проектах, с которыми работают пользователи. Они могут копироваться в такие проекты с использованием клавиши меню «Добавить файлы в проект...», находящейся в выпадающем меню Qucs «Проект». Аналогично, любая новая модель, разработанная как часть проекта, может быть добавлена в библиотеку и использована вновь в будущем.

5.10 Время задержки распространения цифровых компонент и числовая стабильность симуляции переходных процессов

Симуляция переходных процессов в основном много медленнее, чем цифровая симуляция, использующая VHDL сгенерированный машинный код. Модели симуляции переходных процессов для больших сигналов триггеров и других последовательных цифровых устройств предназначены для использования в симуляции схем в смешанном режиме, а не для использования в симуляции чисто цифровых схем. Интересный, и в действительности очень важный вопрос, относится к эффективности и точности алгоритмов числового анализа, применяемого в интегральных подпрограммах, которые являются центром симуляции переходных процессов цепей. Qucs позволяет пользователям выбрать алгоритм, который они хотят применить для симуляции переходных процессов. Доступные алгоритмы – это трапецеидальный, алгоритмы Euler, Gear и Adams Moulton. В каждом случае порядок алгоритма может быть задан от 1 до 6. Второй порядок трапецеидального интегрального алгоритма используется Qucs по умолчанию для симуляции переходных процессов. Для проверки, какой из этих алгоритмов дает наибольшую эффективность по времени при симуляции переходных процессов цифровых цепей, включающих триггеры, был использован счетчик BCD, показанный на рис. 5.13 в качестве тестового с повторной симуляцией при использовании разных интегральных подпрограмм и порядков алгоритмов. Результаты тестов сведены в таблицу 5.1. Очень небольшая разница была обнаружена между схемами, где перекрестно соединенные вентили оба имели нулевую задержку распространения и случаем, когда один вентиль имел задержку 0.5 ns, а второй нулевую.

Один очевидный факт обнаруживается в данных таблицы 5.1, именно, что интегральная подпрограмма высшего порядка Adams Moulton, похоже, быстрее, чем трапецеидальный алгоритм по умолчанию.

Порядок	Trapezoidal	Euler	Gear	Adams Moulton
1	1	1.62	1.65	1.62
2	1	1.62	0.44	1
4	1	1.62	1.28	0.39
6	1	1.62	0.28	0.18

Таблица 5.1. Относительные времена симуляции для цепи, показанной на рис. 5.13

Порядок	Число отбраковок	Среднее время шага
1	1470	5.17737e-12
2	1750	9.4585e-12
4	1454	2.866e-11
6	61	5.76646e-11

Таблица 5.2. Число отбраковок и среднее время шага данных для алгоритма Adams Moulton

Это подтверждается средним временем шага и количеством отбракованных точек данных, выведенных Qucs в конце симуляции. Таблица 5.2 содержит перечень этих данных для алгоритма Adams Moulton из таблицы 5.1.

Таблица 5.2 указывает на увеличение среднего времени шага и эффектное уменьшение числа отбраковок решений симуляции, как на возможное основание для уменьшения времени в симуляции переходного процесса при использовании интегральных подпрограмм Adams Moulton высшего порядка. Однако другие факторы могут оказывать влияние на выбор интегральной подпрограммы. Порою скорость не единственный критерий, имеющий значение при симуляции больших сложных цепей. Рассмотрим следующий случай (цепь, показанная на рис. 5.13, с интегральным анализом переходных процессов Adams Moulton шестого порядка); установка задержки одного из вентилей в 1ns, а другого в 0 ns, в каждой из RS защелок переключаемого фронтом D-триггера производит графический вывод, показанный на рис. 5.18. Совершенно ясно, решение неправильно, что указывает на возможную числовую нестабильность, вызванную выбором интегральной подпрограммы.

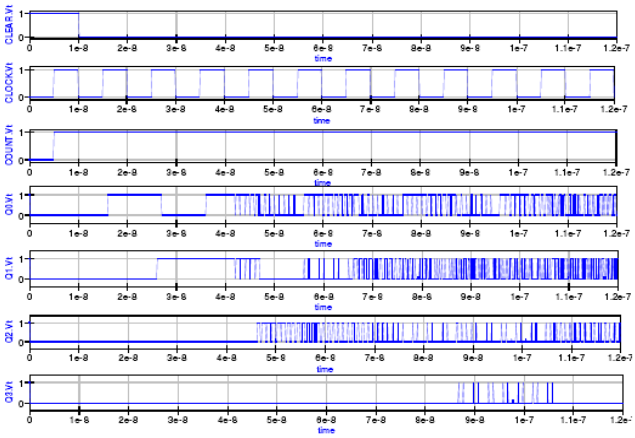


Рис. 5.18. Графика цифрового TimeList для цепи, показанной на рис. 5.13

5.11 Пример симуляции смешанного режима

Симуляция смешанного режима включает симуляцию цепей, которые содержат электронные устройства и цепи из разных физических областей. Наиболее очевидна смесь из аналоговых и цифровых компонент. Qucs был разработан до точки, где он может поддерживать подобный тип цепей, заданный моделями устройств, охватывающими разные физические области. В будущем такие цепи уподобятся встроенным компонентам из разных областей, включая, например, цифровые компоненты обработки сигналов (DSP) и, возможно, нано-механические устройства. Много-доменная симуляция добавляет дополнительную сложность в процесс симуляции, обычно отсутствующую при одно-доменной симуляции. Каждая область (домен) обычно представляет сигнальные данные со специфическими для данной области атрибутами – напряжение и ток для аналоговых величин, булевы «1» и «0» для цифровых сигналов и числа с плавающей точкой для DSP (цифровая обработка сигналов). Следовательно, сигналы проходящие от одного домена к другому, должны быть преобразованы из одного формата в другой. Эти преобразованные элементы, часто называемые узловыми мостами (node-bridge), и формируют сущностную часть процесса симуляции смешанного режима. Три примера, которые приведены в этом разделе заметок, были выбраны для иллюстрации некоторых базовых идей, сосредоточенных вокруг смешанной симуляции цепей с аналоговыми и цифровыми компонентами, и чтобы показать, как Qucs обходится с симуляцией данного типа. В последнем разделе был сделан акцент на важность правильного выбора интегральной подпрограммы при симуляции схем во временной области. Смешанные цепи часто включаются в набор компонент, демонстрирующих весьма различные временные значения. Этим создаются проблемы числовой стабильности даже более критичные, чем работа симулятора. С определенными числовыми интегральными подпрограммами, подобно трапецеидальной, возникает числовая нестабильность, если временной шаг симуляции становится много больше, чем наименьшая временная константа в цепи. Следовательно, чтобы выполнить успешное завершение симуляции, временной шаг интеграции должен быть уменьшен, что в свою очередь делает предельное время симуляции многократно увеличенным. Неявный алгоритм Gear[48] не страдает от этих проблем, и это естественный выбор для цепей с компонентами, которые имеют большую разницу во временных константах.

- Пример 1: Аналоговый график, производимый цифровыми устройствами с выходным узловым мостом. Цепь на рис. 5.19 показывает источник аналогового напряжения производящий цифровой инвертор с элементом узловой мост, обрабатывающим выходной сигнал инвертора. Входной сигнал

48 Алгоритм интеграции Gear – это мощный метод решения жестких систем дифференциальных уравнений, см. Donald A. Calahan, Computer Aided Network Design, Revised edition, 1972, McGraw-Hill.

– это синусоидальное напряжение с амплитудой 1V. Выходной сигнал инвертора, V1 на рис. 5.19, имеет асимметричную форму, поскольку пороговая точка для инвертора установлена в 0.5V, на полпути между двумя логическими уровнями. Элемент узлового моста базируется на управляемом напряжением источнике, где приращение устройства и задержка времени могут программироваться. В этом первом примере приращение было установлено в 5, а задержка времени в 0.5 ns. Рис. 5.20 иллюстрирует графический вывод TimeList симуляции для этого примера смешанной цепи. Узловой мост, показанный на рис. 5.19 – это базовое устройство. Более того, могут устанавливаться значение таких дополнительных параметров, как время нарастания и спада. Следующий пример демонстрирует использование активного узлового моста.

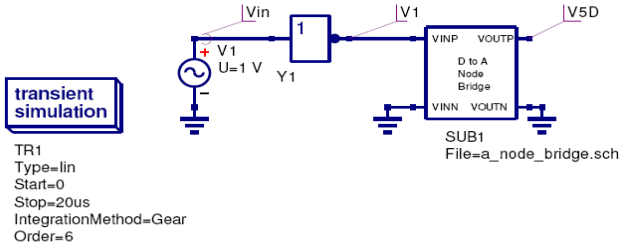


Рис. 5.19. Аналоговый вывод, производимый цифровым устройством с выходным узловым мостом

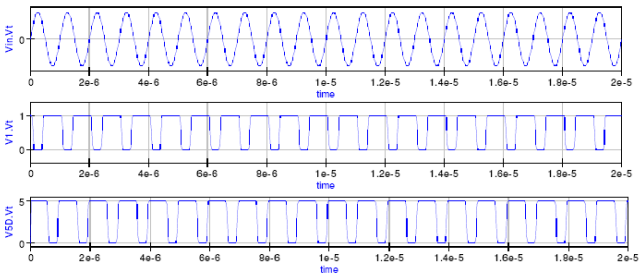


Рис. 5.20. Цифровой графический TimeList для цепи рис. 5.19

- Пример 2: Импульсы, производимые цифровым инвертором с активным узловым мостом. Иллюстрация рис. 5.21 – это цепь похожая на предыдущий пример. На рис. 5.21 генератор импульсов ведет цифровой инвертор. Выходной сигнал инвертора проходит через активный узловой мост, произведенный из базового BJT переключаемого усилителя. Выходной график для этой цепи показан на рис. 5.22. Заметьте, что времена переднего и заднего фронтов определяются усилителем узлового моста, и что амплитуда результирующего аналогового сигнала установлена в 5V.

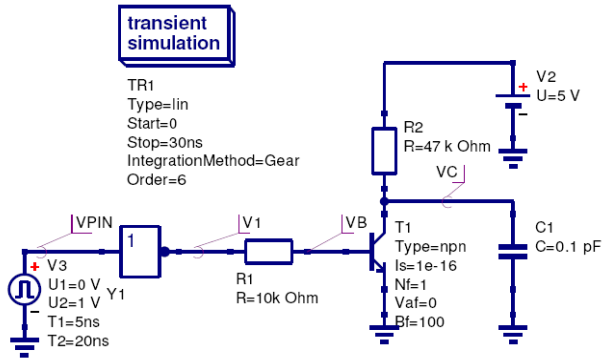


Рис. 5.21. Управляемый импульсами цифровой инвертор с активным узловым мостом

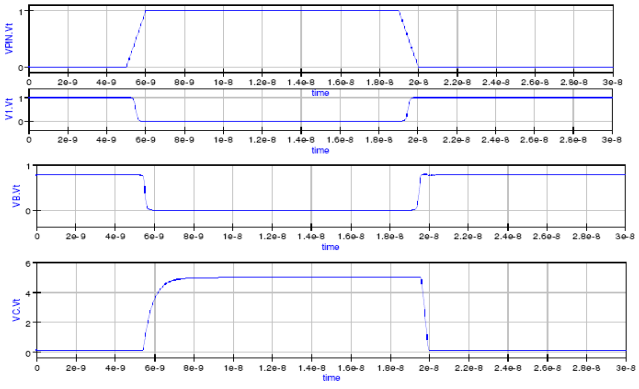


Рис. 5.22. Графические выводы цифрового TimeList для цепи на рис. 5.21

- Пример 3: Более сложный пример симуляции смешанного режима. Цепь, показанная на рис. 5.23, собрала вместе ряд идей, очерченных в этих заметках. 4X-битовый сигнал генерируется простым асинхронным двоичным счетчиком, работающим с цифровым тактовым сигналом. Выход счетчика преобразуется в аналоговую область с помощью простого узлового моста (node-bridge) типа, приведенного в примере 1. 4-х битовый взвешенный DAC конвертирует сигналы узлового моста в окончательный аналоговый выходной сигнал. Операционный усилитель DAC моделирован как блок усиления с одно-полюсной частотной характеристикой и ограничением выходного DC напряжения. Выходные графики для этого примера показаны на рис. 5.24, а детали модели операционного усилителя на рис. 5.25.

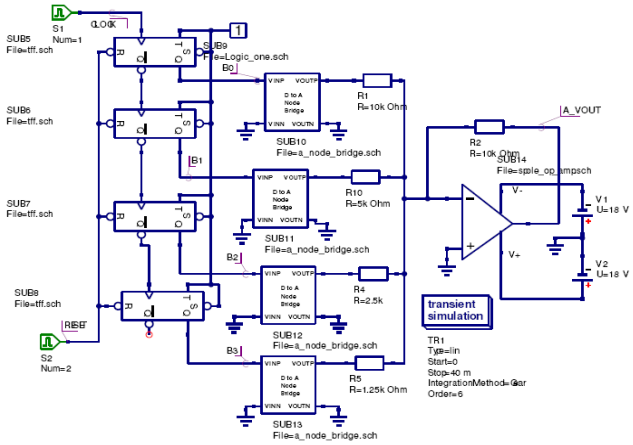


Рис. 5.23. Более сложный пример аналого-цифровой смешанной симуляции

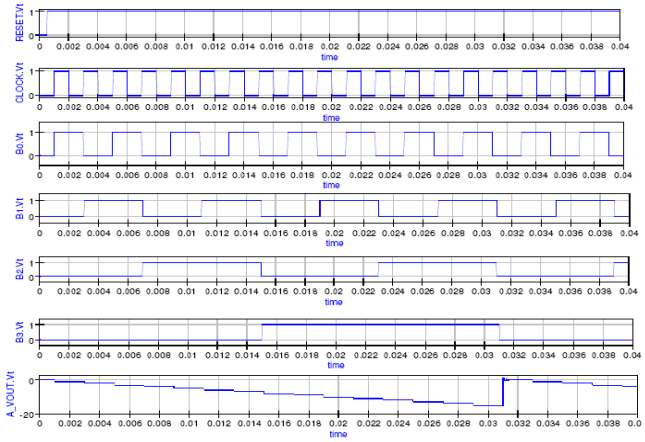


Рис. 5.24. Графический вывод цифрового TimeList для цепи, показанной на рис. 5.23

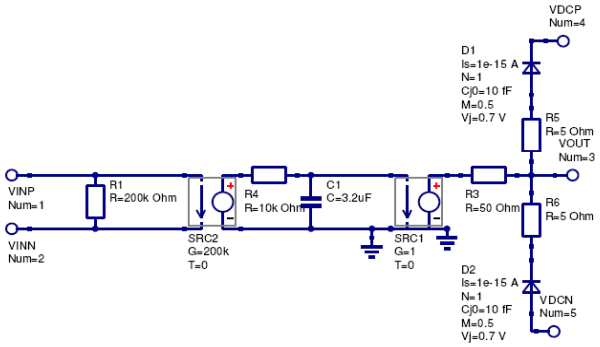


Рис. 5.25. Модель операционного усилителя с Rin = 200k, частотным полюсом = 5Hz, DC дифференциальным усилением = 200k и Rout = 50 Ω

5.12 Заключительные замечания

Примеры, описанные в этих заметках были все симулированы с использованием последней версии CVS кода Qucs. Начиная с выпуска версии 0.0.8, Qucs был готов к тому, чтобы использоваться для симуляции в смешанном режиме, а многие из известных ошибок в Qucs 0.0.8 будут скорректированы в выпуске Qucs 0.0.9 через некоторое время. Выпуск 0.0.9 будет представлять другой важный шаг в развитии действительно универсального симулятора. Однако значительно больше работы потребуется сделать по развитию моделей для использования в разных физических областях. Мои благодарности Michael Margraf и Stefan Jahn за всю их тяжелую работу по коррекции ошибок, которые всплыли при тестировании примеров, представленных в этом руководстве.

6 Моделирование операционных усилителей

6.1 Введение

Операционные усилители (OP AMP) – это фундаментальные строительные блоки линейной электроники. Они широко применялись в разработках линейных цепей с тех пор, как были впервые представлены около тридцати лет назад. Использование моделей операционных усилителей для симуляции схем с помощью SPICE и других популярных симуляторов схем широко распространено, и многие производители предоставляют модели для их устройств. В большинстве случаев эти модели не пробуют симулировать внутренние цепи на уровне устройства, а используют макро-моделирование для представления поведения усилителя, как это происходило бы на выводах устройства. Цель этих заметок по руководству пояснить, как эти макро-модели могут быть использованы для симуляции ряда свойств операционных усилителей, и показать, как параметры макро-модели могут быть получены из справочных данных производителей. Руководство концентрируется на моделях, которые могут быть симулированы с использованием Qucs выпуска 0.0.9.

6.2 Qucs встроенная модель операционного усилителя

Qucs включает модель идеального операционного усилителя. Его символ можно найти в списке нелинейных компонент. Эта модель представляет операционный усилитель, как идеальное устройство с дифференциальным усилением и ограничением выходного напряжения. Модель предназначена для использования с простыми блоками усиления и не должна использоваться в симуляции схем, где свойства операционного усилителя критичны для поведения всей остальной схемы. Рис. 6.1 показывает базовый инвертирующий усилитель с усилением в 10 раз, базируемый на Qucs OP AMP модели. Симуляция AC представления этой цепи показано на рис. 6.2. Из рис. 6.2 видно, что усиление схемы и фазовый сдвиг – это константы и они не меняются с изменением частоты входного сигнала. Это, конечно, идеальная ситуация, которую практический операционный усилитель не воспроизведет. Давайте сравним поведение этой же схемы с операционным усилителем, представленным схемой уровня устройства. Показанное на рис. 6.3 – это диаграмма транзисторной схемы для хорошо известного операционного усилителя UA741[49]. Результаты усиления и фазы для этой схемы

49 UA741 операционный усилитель – это одно из самых изученных устройств. Все почти уникальное в этой транзисторного уровня модели было сконструировано для этого устройства. Детали работы схемы и моделирования этого устройства можно найти в (1) Paul R. Grey et. al., Analysis and Design of Analog

показаны на рис. 6.1, где OP AMP смоделировано по UA741 транзисторного уровня модели, данной на рис. 6.4. Кривые на этом рисунке ясно иллюстрируют разницу между двумя моделями симуляции. Когда симулируется схема, которая включает операционные

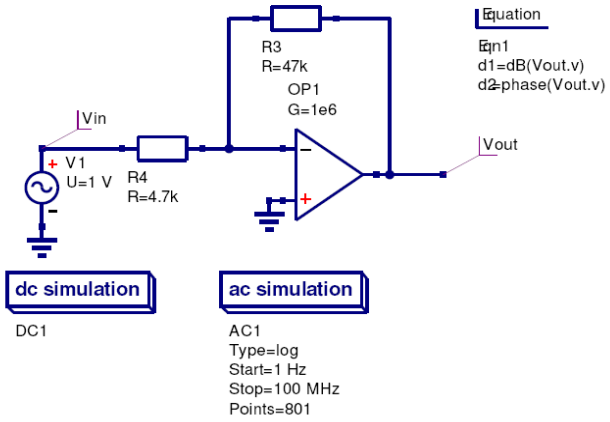


Рис. 6.1. Qucs схема для базового OP AMP инвертирующего усилителя: Qucs OP AMP имеет G=1e6 и Umax=15V

усилители, качество модели OP AMP может быть зачастую ограничивающим фактором в точности всех остальных результатов симуляции. Точные OP AMP модели обычно включают ряд следующих характеристик устройства: (1) DC и AC дифференциальное усиление, (2) смещение входного тока, (3) уход входного тока и напряжения, (4) входной импеданс, (5) синфазный сигнал, (6) скорость нарастания выходного напряжения, (7) выходной импеданс, (8) эффекты пониженного питания, (9) шумы, (10) ограничения выходного напряжения, (11) ограничения выходного тока и (12) эффекты восстановления сигнала после перегрузки. Точное смешивание выбранных свойств в большой степени зависит от цели используемой модели. Например, если модель требуется только для симуляции AC передаточной функции при малом сигнале, тогда включение раздела ограничений выходного напряжения OP AMP модели не является необходимым, или вполне может считаться неуместным. В следующих разделах этой статьи руководства разбираются макро-модели для нескольких параметров OP AMP, перечисленных выше, и в каждом случае необходимая техника ограничена показом того, как воссоздать параметры макро-модели по справочным данным производителей.

Integrated Circuits, Fourth Edition, 2001, John Wiley and Sons INC., ISBN 0-471-32168-0, и (2) Andrei Vladimirescu, The SPICE book, 1994, John Wiley and Sons, ISBN 0-471-60926-9.

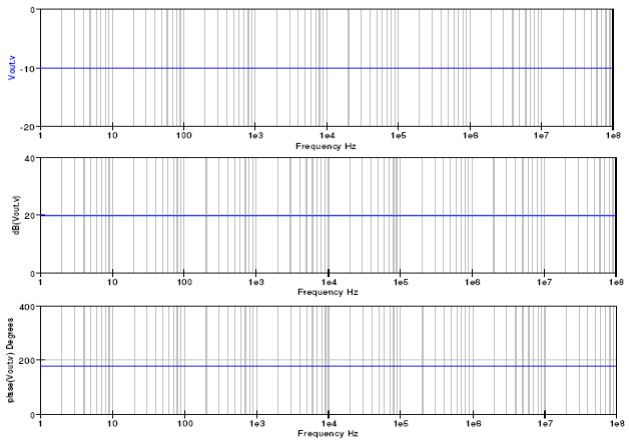


Рис. 6.2. Кривые усиления и фазы для базового OP AMP инвертирующего усилителя

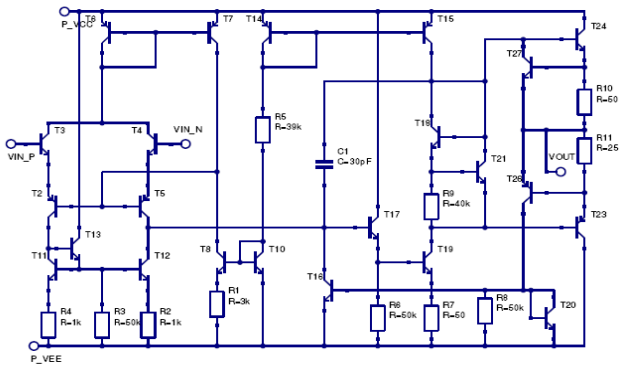


Рис. 6.3. Транзисторный уровень схемы для операционного усилителя UA741

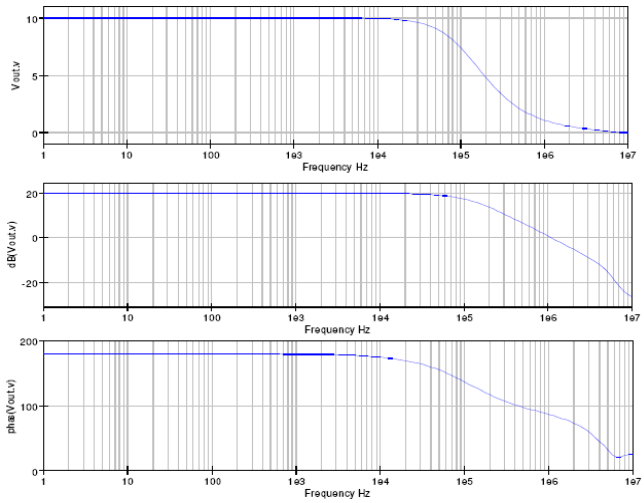


Рис. 6.4. Кривые усиления и фазы для 10 кратного усиления инвертирующим усилителем с OP AMP, представленным транзисторным уровнем UA741 модели

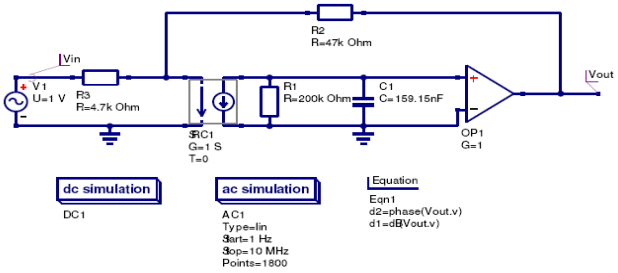


Рис. 6.5. Модифицированная Qucs OP AMP модель для получения одно-полюсной частотной характеристикой

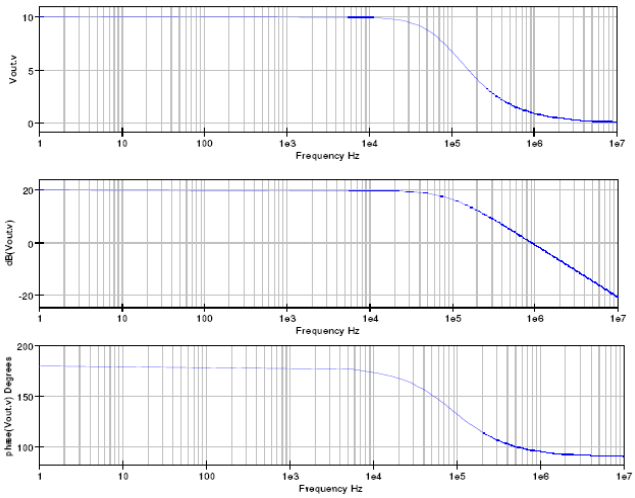


Рис. 6.6. Кривые усиления и фазы для схемы, показанной на рис. 6.5

6.3 Дополнительные возможности Qucs OP AMP модели

В предыдущем разделе было показано, что Qucs OP AMP модель имеет частотную характеристику, которая не зависит от частоты. Добавлением внешних компонентов к Qucs OP AMP модели функциональность модели может быть улучшена. Усиление UA741 с разомкнутой петлей обратной связи имеет полюс около 5Hz и спад частотной характеристики 20 dB на декаду от первого полюса до второго, около 3 MHz. Схема, показанная на рис. 6.5, моделирует дифференциальную частотную характеристику UA741 от DC до 1 MHz. Рис. 6.6 иллюстрирует частотную характеристику с замкнутым контуром для модифицированной Qucs OP AMP модели.

6.4 Макро-модели модульного операционного усилителя

Макро-моделирование – это термин, данный процессу моделирования электронного устройства, как «черного ящика», где индивидуальные характеристики устройства специфицированы в терминах сигналов и других свойств, получаемых на входных и выходных выводах «черного ящика». Такие модели работают на функциональном уровне, а не на более детальном транзисторном уровне схемы, предлагая значительное увеличение эффективности вычислений[50]. Макро-модели обычно произведены непосредственно из справочных данных производителя. Для большинства операционных усилителей модели транзисторного уровня не предоставляются производителями. Одно примечательное исключение сделано для операционного услителя UA741, показанного на рис. 6.3. Блок диаграмма модулей[51] общего назначения OP AMP макро-модели проиллюстрирована на рис. 6.7. На этой диаграмме блоки представляют специфические характеристики усилителя, моделируемые электрическими сетями, собранными из компонент, находящихся во всех популярных симуляторах схем[52]. Каждый блок состоит из одного или большего числа компонент, которые моделируют единственный параметр усилителя или группу родственных параметров, таких как выходное смещение тока и напряжения. Этим обеспечивается, что изменения одного частного параметра, не распространятся косвенно на другие

50 Вычислительная эффективность многократно возрастает благодаря тому факту, что макро-модели операционных усилителей имеют, в среднем, около одной шестой от числа узлов переходов по сравнению с моделью транзисторного уровня. Кроме того, число нелинейных p-n переходов, включенных в макро-модель, часто меньше десяти, что сравнительно лучше сорока или пятидесяти нужных для модели усилителя на транзисторном уровне.

51 Brinson M. E. and Faulkner D. J., Modular SPICE macromodel for operational amplifiers, IEE Proc. Circuits Devices Syst., Vol. 141, No. 5, October 1994, pp. 417-420.

52 Модели, использующие нелинейные управляемые источники, например для SPICE B источники напряжения и тока, не допускаются в Qucs выпуска 0.0.9. Нелинейные управляемые источники – одна из возможностей в Qucs to-do списке.

параметры. Локальные узлы и масштабирование также применяются в макро-модельных блоках. Кроме того, поскольку каждый блок работает отдельно, масштабируемые напряжения не распространяются за пределы отдельных блоков. Каждый блок может перемещаться с подсхемой Qucs, что имеет требуемую спецификацию и буферизацию от других блоков. Более того, все подсхемы – это самодостаточные сущности, где детали внутренней схемы скрыты от других блоков. Такой подход похож на структурированное высоко-уровневое компьютерное программирование, где внутренние детали функций скрыты от пользователей. Когда характеристики устройства, специфицированные для каждого блока, отделены от всех других характеристик устройства, только те характеристики усилителя, что нужны, включаются в данную макро-модель. Такой подход направляет к подлинно структурированным макро-моделям. Следующие разделы представляют детали и источники электрических сетей, формирующих блоки, показанные на рис. 6.7. Чтобы проиллюстрировать работу модульной OP AMP макро-модели, значения параметров блока рассчитываются для UA741 OP AMP и используются в серии показательных симуляций. До конца этих заметок будут представлены данные для некоторых других популярных общего назначения операционных усилителей.

6.5 Базовая AC OP AMP макро-модель

Минимальный набор блоков, требуемый для того, чтобы модульная макро-модель функционировала, как усилитель: входной каскад, усилительный каскад и выходной каскад. Этим формируется ядро модулей для всех макро-моделей.

6.5.1 Входной каскад

Входной каскад включает напряжение смещение усилителя, отклонение и смещение токов, и компоненты дифференциального входного импеданса. Цепь для входного каскада показана на рис. 6.8, где

1. *R1 = R2* = Половина входного дифференциального сопротивления усилителя (*RD*).
2. *Cin* = Дифференциальная входная емкость усилителя (*CD*).
3. *Ib1 = Ib2* = Отклонение входного тока усилителя (*IB*).
4. *Ioff* = Половина входного тока смещения усилителя (*IOFF*).
5. *Voff1 = Voff2* = Половина входного напряжения смещения (*VOFF*).

Типичные значения для UA741 OP AMP:

1. *RD = 2 MΩ и R1 = R2 = 1MΩ*

2. *CD = Cin1 = 1.4 pF.*
3. *IB = Ib1 = Ib2 = 80 nA.*
4. *IOFF = 20 nA u Ioff1 = 10 nA.*
5. *VOFF = 0.7 mV u Voff1 = Voff2 = 0.35 mV.*

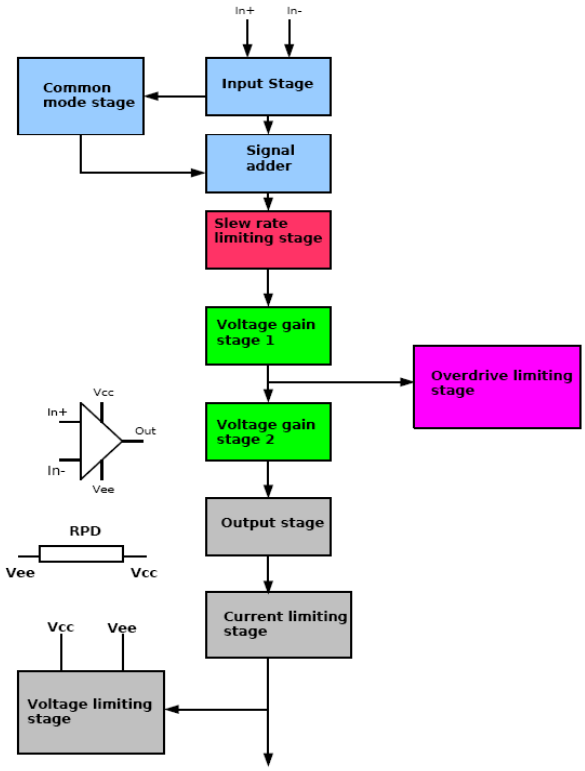


Рис. 6.7. Блок-диаграмма макро-модели операционного усилителя

Дифференциальный выходной сигнал (VD) задан через VD–P1 – VD–N1, а синфазный выходной сигнал (VCM) через (VD–P1 + VD–N1)/2.

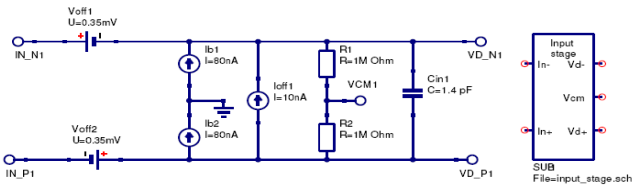


Рис. 6.8. Модульный блок OP AMP входного каскада

6.5.2 Усилитель напряжения с одним каскадом

Цепь для усилителя напряжения с одним каскадом показана на рис. 6.9, где

1. RD1 = 100 MΩ = «Фальшивый» входной резистор, добавленный для соединения узлов IN−P 1 и IN−N1 по постоянному току.
2. GMP1 = 1 S = Напряжение единичного усиления, управляющее генератором тока.
3. RADO = Разомкнутое по постоянному току дифференциальное усиление (AOL(DC)) OP AMP.
4. CP1 = 1/(2*π*GBP), где GBP = OP AMP произведение к-та усиления на ширину полосы пропускания

Типовые значения UA741 OP AMP:

1. RADO = 200k (AOL(DC) = 106 dB).
2. CP1 = 159.15 nF (Типовое значение для UA741 GBP – 1 MHz).

6.5.3 Производная передаточная функция одно-каскадного усилителя напряжения

Большинство операционных усилителей общего применения характеризуются дифференциальным усилением по напряжению разомкнутого контура, которое имеет (1) очень большое значение на DC, (2) главный полюс (fp1) на низкой частоте, обычно ниже 100 Hz, и (3) частотную характеристику, которая спадает на 20 dB на декаду до частоты единичного усиления, которая часто в районе Mhz. Эта форма частотной характеристики имеет постоянное произведение коэффициента усиления на полосу пропускания (GBP) во всем диапазоне частот от fp1 до GBP. Типичная частотная характеристика OP AMP с разомкнутой обратной связью показана на рис. 6.10. Передаточная функция усиления по напряжению для этого типа характеристики может быть моделирована электрической сетью данной на рис. 6.9, где AC передаточная характеристика напряжения

$$vout(POLE\_1\_OUT1) = \frac{GMP1 * (V(IN\_P1) - V(IN\_N1)) * RADO}{1 + j(\omega * RADO * CP1)} \quad (6.1)$$

Where

$$f_{P1} = \frac{1}{2\pi * RADO * CP1} \quad (6.2)$$

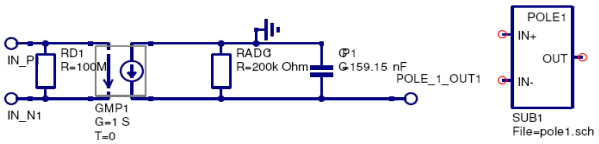


Рис. 6.9. Модульный OP AMP одно-каскадного усилителя напряжения

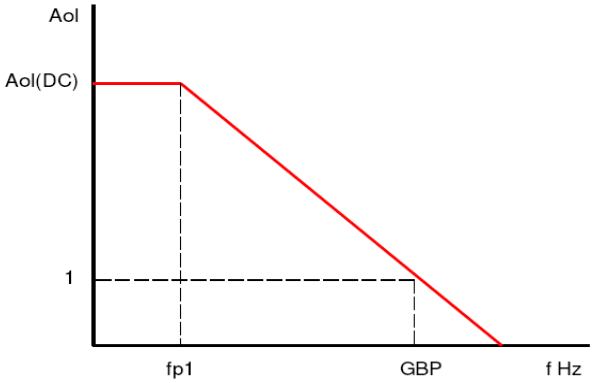
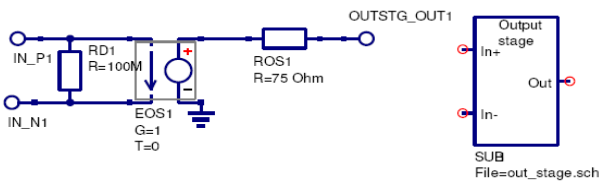


Рис. 6.10. OP AMP дифференциальное усиление по напряжению с разомкнутой обратной связью как функция частоты

Let 1



(6.3)

Рис. 6.11. Модульная макро-модель выходного каскада

6.5.4 Выходной каскад

Электрическая сеть, представляющая базовый выходной каскад дана на рис. 6.11, где

1. RD1 = 100 MΩ = «Фальшивое» входное сопротивление, добавленное для соединения узлов IN−P1 и IN−N1 по DC (постоянному току).
2. EOS1 G = 1 = Единичное усиление по напряжению, управляющее генератором напряжения.
3. ROS1 = OP AMP выходное сопротивление.

Типовое значение для UA741 OP AMP выходное сопротивление ROS1 = 75Ω.

6.5.5 Модель подсхемы для базовой AC OP AMP макро-модели

Модель для базовой AC OP AMP макро-модели показана на рис. 6.12. Напряжение синфазного сигнала входного каскада (V cm) не используется в этой макро-модели и было оставлено свободным. Для проверки поведения AC макро-модели ее действие сравнивалось с моделью UA741 транзисторного уровня. Рис. 6.13 показывает схему цепи для двух инвертирующих усилителей, каждый с коэффициентом усиления десять, подключенных к общему AC источнику. Один из усилителей использует простую AC макро-модель, а второй UA741 модель транзисторного уровня. Рис. 6.14 иллюстрирует выходные усиления и фазы кривые для обоих усилителей. В основном нарисованные кривые очень схожи. Однако на частоте выше GBP базовая AC макро-модель не корректно моделирует реальное OP AMP поведение. Этого и следовало ожидать, поскольку простая AC макро-модель не включает каких-либо высокочастотных моделирующих компонент. Заметьте также, что DC выходное напряжение для vout и vout3 очень похожи, см. DC табличные результаты, данные на рис. 6.13.

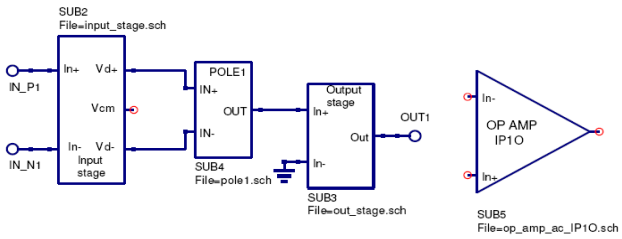


Рис. 6.12. Простая AC OP AMP макро-модель

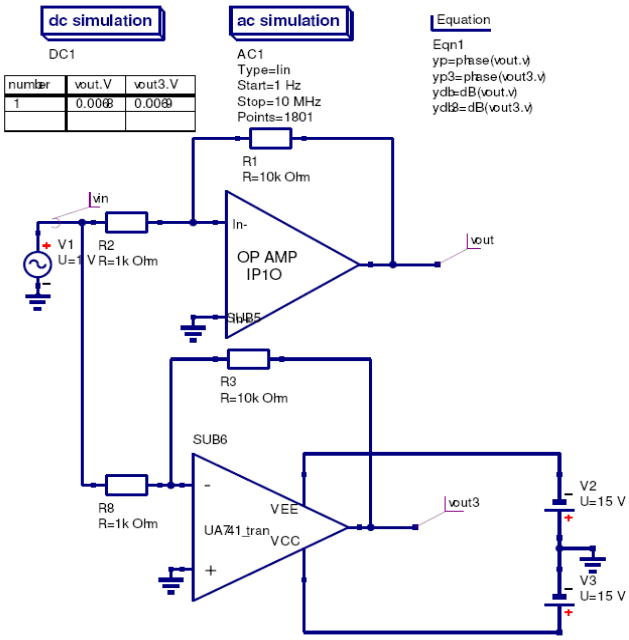


Рис. 6.13. Тестовая цепь для инвертирующего усилителя. Выходные сигналы: (1) vout для AC макромодели, (2) vout3 для UA741 транзисторной модели

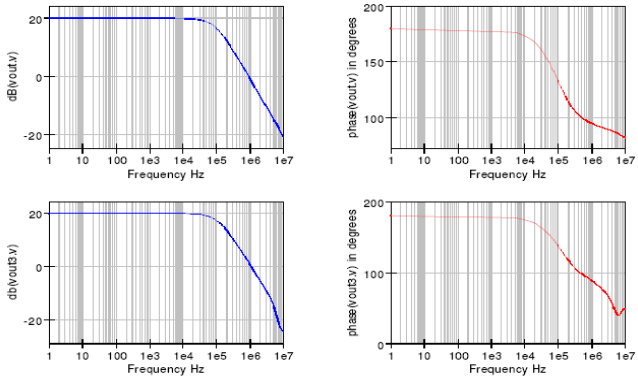


Рис. 6.14. Тестовые результаты симуляции схемы, показанной на рис. 6.13

6.6 Более точная OP AMP AC макро-модель

Большинство OP AMP общего назначения имеют высокочастотный полюс на дифференциальной частотной характеристике с разомкнутой петлей обратной связи. Добавлением второго каскада усиления в простую AC макро-модель поведение частотной характеристики на высоких частотах может быть скорректировано. Модель для второго усилительного каскада показана на рис. 6.15. Этот дополнительный каскад усиления похож на первый каскад усиления, где

1. RD2 = 100 MΩ = «Фальшивый» входной резистор, добавленный для образования пути прохождения DC между узлами IN_P2 и IN_N2.
2. GMP2 = 1 S = Единичного усиления, управляемый напряжением генератор тока.
3. RP2 = 1 Ω.
4. CP2 = 1/(2π*fp2), где fp2 = второй частотный полюс в Hz.

Типичное значение для UA741 OP AMP высокочастотного полюса fp2 = 3M Hz.

6.6.1 Производная переходная функция двухкаскадного усилителя напряжения

Дифференциальная передаточная функция усиления напряжения для двухкаскадного усилителя задается

$$vout(POLE\_2\_OUT1) = \frac{GMP2 * (V(IN\_P2) - V(IN\_N2)) * RP2}{1 + j(\omega * RP2 * CP2)} \tag{6.4}$$

Let $RP2 = 1\Omega$ and $GMP2 = 1\ S$. Then

$$vout(POLE\_2\_OUT1) = \frac{V(IN\_P2) - V(IN\_N2)}{1 + j(\omega * CP2)} \tag{6.5}$$

and

$$CP2 = \frac{1}{2\pi * fp2} \tag{6.6}$$

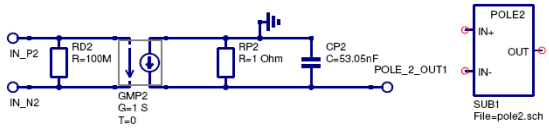


Рис. 6.15. Модульный OP AMP, второй каскад усилителя напряжения

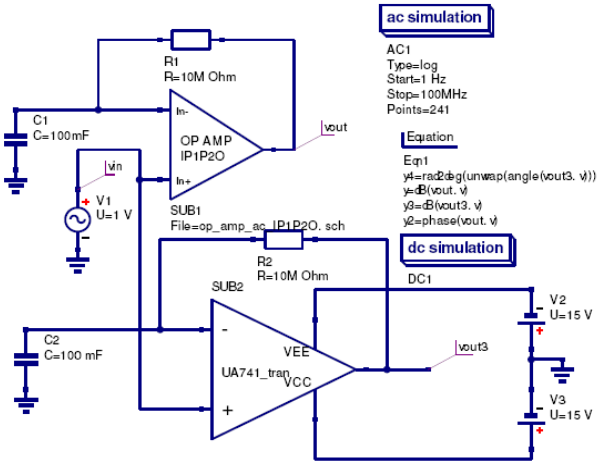


Рис. 6.16. Тестовая цепь для симуляции OP AMP дифференциального усилителя с разомкнутой петлей обратной связи

6.6.2 Симуляция OP AMP дифференциального усилителя с разомкнутой обратной связью

Схема, показанная на рис. 6.16 позволяет симулировать дифференциальный усилитель с разомкнутым контуром обратной связи (Aol(f)). Эта цепь обслуживается резистором обратной связи для обеспечения DC стабильности. Рис. 6.16 иллюстрирует две

тестовые цепи, подключенные к общему источнику AC. Это позволяет сравнить поведение AC макро-модели и модели UA741 транзисторного уровня. Передаточная функция AC напряжения для тестовой цепи

$$vout(f) = \frac{Aol(f)}{1 + \frac{Aol(f)}{1 + j\omega * R * C}} vin(f) \qquad (6.7)$$

$$\text{where } vout(f) = (V^+ - V^-) * Aol(f),\ V^+ = vin(f),\ \text{and } V^- = \frac{vout(f)}{1 + j\omega * R * C}$$

При $\frac{(Aol(f))}{(\omega * R * C)} << 1$, уравнение (7) становится $vout(f) \Rightarrow Aol(f) \quad vin(f)$.

Следовательно, и для частот, где выполняется условие *vout(f) = Aol(f),* когда *vin(f) = 1 V*. Рис. 6.17 показывает графический вывод симуляции данных с разомкнутым контуром. С тестовой константой времени схемы, установленной в 1e6 секунд, данные точны до частоты 1 Hz.

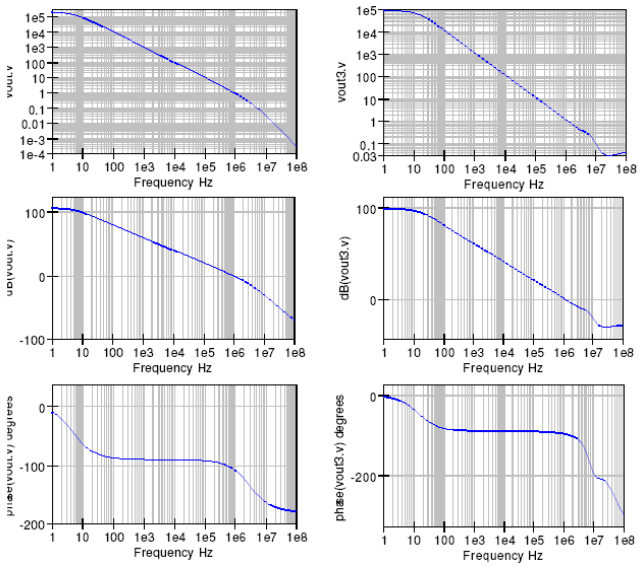


Рис. 6.17. Результаты тестовой симуляции для цепи рис. 6.16

6.7 Добавление эффектов синфазного сигнала к OP AMP AC макро-модели

Дифференциальное усиление с разомкнутым контуром AD(f) для большинства операционных усилителей общего назначения может быть приближено

$$A_D(f) = AD(0)\frac{1}{1+j\frac{f}{f_{PD}}} \tag{6.8}$$

Похожим образом усиление синфазного сигнала ACM (f) может быть представлено такой же одно-полюсной частотной характеристикой и характеристикой с единственным нулем задаваемой

$$A_{CM}(f) = A_{CM}(0)\frac{1+j\frac{f}{f_{CMZ}}}{1+j\frac{f}{f_{PD}}} \tag{6.9}$$

Определение коэффициента подавления синфазного сигнала CMRR(f) для OP AMP

$$CMRR(f) = \frac{A_D(f)}{A_{CM}(f)} \tag{6.10}$$

дает

$$CMRR(f) = CMRR(0)\frac{1}{1+j\frac{f}{f_{CMZ}}} \tag{6.11}$$

где

$$CMRR(0) = \frac{A_D(0)}{A_{CM}(0)} \tag{6.12}$$

Эффекты синфазного сигнала могут быть добавлены в OP AMP макро-модели включением каскада в модульную макро-модель, который вводит нуль в частотную характеристику усилителя. Выход VCM из входного каскада макро-модели чувствует синфазный сигнал. Этот сигнал, проходя через CR сеть, генерирует требуемый синфазным сигналом ноль. Рис. 18 дает модель сети, генерирующей ноль, где

1. RDCMZ = 650 MΩ = синфазного сигнала входное сопротивление/2.
2. RCM1 = 1 MΩ
3. ECM1 G = 31.623 = $\frac{\frac{RCM1}{RCM2}}{CMRR(0)}$(ЗАМЕТЬТЕ: RCM1/RCM2 масштабный к-т)

$$\frac{1}{2\pi * RCM1 * f_{CMZ}}.$$

4. CCM1 = 795.8 pF =

5. RCM2 =1Ω

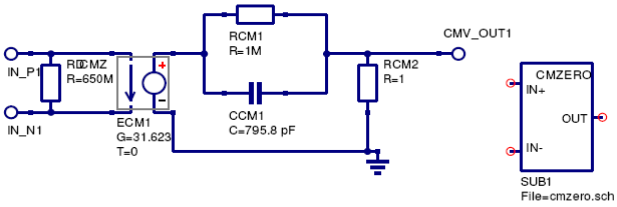


Рис. 6.18. Макро-модель нулевого синфазного сигнала

Типичные значения для UA741 OP AMP:

1. Входное сопротивление синфазного сигнала = 1300 MΩ.
2. CMRR(0) = 90 dB.
3. f_{CMZ} = 200 Hz.

Передаточная функция AC напряжения для нуля синфазного сигнала передаточной функции

$$Vout(\texttt{CMV\_OUT1}) = \frac{RCM2}{RCM1}\left[\frac{1+j\omega * RCM1 * CCM1}{1+j\omega * RCM2 * CCM1}\right][V(\texttt{IN\_P1}) - V(\texttt{IN\_N1})] \quad (6.13)$$

Когда $\frac{RCM2}{RCM1}$ << 1, полюс, задаваемый RC сетью синфазного сигнала, находится на очень высокой частоте и может быть опущен. Комбинируя ноль синфазного сигнала с ранее определенными моделями каскадов усиления, получаем макро-модель, показанную на рис. 6.19. На этой модели дифференциальный и синфазный сигналы скомбинированы с использованием простого аналогового сумматора, базируемого на управляемых напряжением генераторах тока.

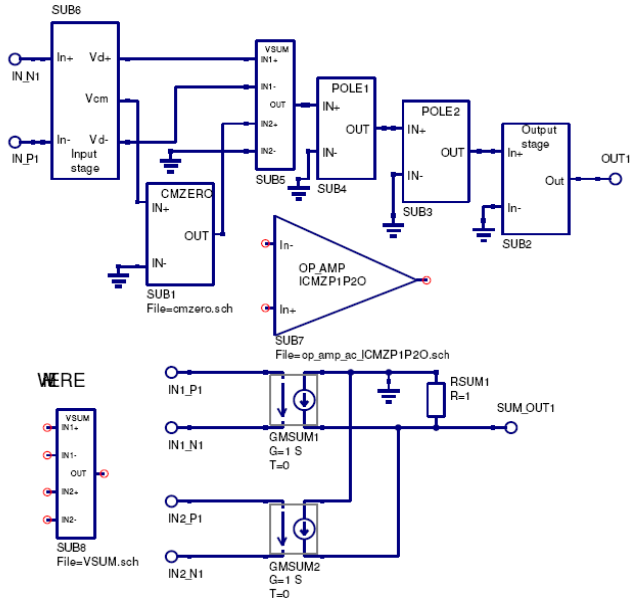


Рис. 6.19. AC макро-модель, включающая ноль синфазного сигнала

6.7.1 Симуляция эффектов OP AMP синфазного сигнала

Эффекты OP AMP синфазного сигнала могут симулироваться цепью, показанной на рис. 6.20[53]. Результирующие напряжения (vout.v и vout3.v) для тестирования цепи с соответствующими резисторами показано на рис. 6.21, где $\frac{vout(0)}{vin} = \frac{1}{CMRR(0)}$.

Чисто тестовые результаты для макро-модели и транзисторной модели UA741 очень похожи. В случае макро-модели для расчета значений компонент были использованы параметры устройства. Однако в модели транзисторного уровня точные значения параметров компонент не известны[54].

53 Brinson M.E. и Faulkner D.J., New approaches to measurement of operational amplifier common-mode rejection ratio in the frequency domain, IEE Proc-Circuits Devices Sys., Vol 142, NO. 4, August 1995, стр. 247-253.

54 UA741 модель транзисторного уровня базируется на оценке параметрического процесса, который определяет характеристики транзисторов UA741. Следовательно, модель уровня устройства не похоже, чтобы была абсолютно идентична модели, произведенной из значений типичных параметров, найденных в справочных данных OP AMP. Из результатов симуляции CMRR(0) значения приблизительно (1) макро-модель 90 dB, (2) UA741 транзисторная модель 101 dB. Аналогично, нулевые частоты общего режима приблизительно (1) макро-модели 200 Hz, (2) UA741 транзисторной модели 500 Hz.

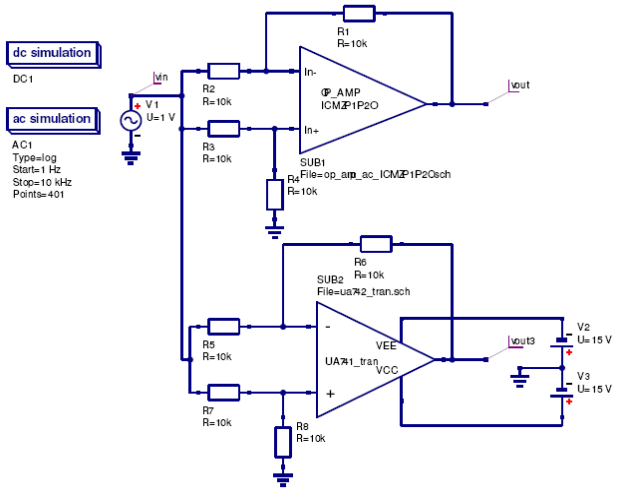


Рис. 6.20. Симуляция представления OP AMP синфазного сигнала

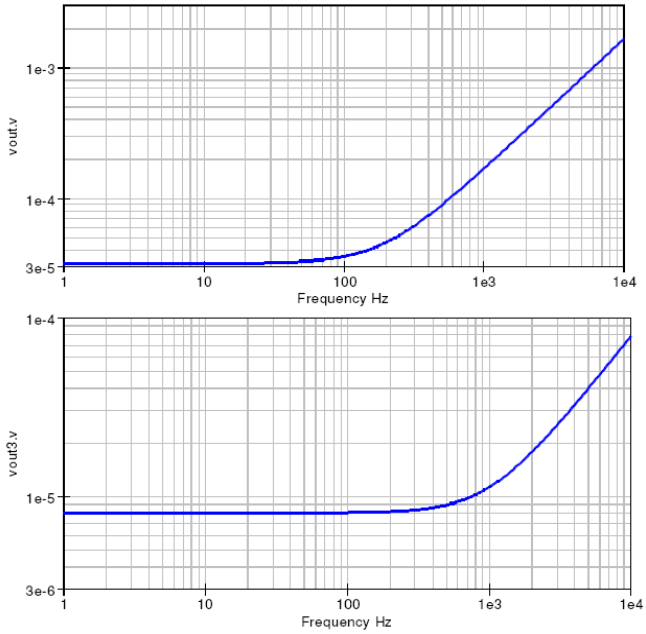


Рис. 6.21. Результаты тестовой симуляции цепи, показанной на рис. 6.20

6.8 Область переходных процессов большого сигнала OP AMP макро-моделей

Модульная макро-модель, представленная в предыдущих разделах, концентрировалась на моделировании OP AMP представления в области малых сигналов AC. Модели большого сигнала нуждаются в том, чтобы принималось в расчет прохождение сигнала через OP AMP в области времен и ограничений отклонения от номинального значения напряжения и тока с размахом практических значений, обнаруживаемых в реальных усилителях. Начиная с AC области, макро-модель, представленная в предыдущих разделах, дополняется скоростью нарастания, ограничивающей усиление и перегрузку каскада, и более корректно моделирующую ограничения высокой скорости OP AMP при большом сигнале. Кроме того, с добавлением каскадов, ограничивающих выходное напряжение и ток, макро-модель OP AMP будет корректно моделировать эффекты большого сигнала, когда уровни сигнала достигают питающих напряжений цепи или текущих ограничений OP AMP выхода.

6.8.1 Источник скорости нарастания макро-модели

Скорость нарастания OP AMP может моделироваться ограничением тока заряда *CP1* в первом каскаде усиления напряжения POLE1. Из рис. 6.9

$$GMP1\,(V(IN\_P1) - V(IN\_N1)) = \frac{V(POLE\_1\_OUT1)}{RADO} + CP1 * \frac{dV(POLE\_1\_OUT1)}{dt} \tag{6.14}$$

Следовательно, при условии, что *RADO* большое[55]

$$GMP1\,(V(IN\_P1) - V(IN\_N1)) \simeq CP1 * \frac{dV(POLE\_1\_OUT1)}{dt} \tag{6.15}$$

Но $CP1 = \frac{1}{2\pi * GBP}$

Вывод

$$GMP1\,(V(IN\_P1) - V(IN\_N1)) \simeq \frac{1}{2\pi * GBP} * \frac{dV(POLE\_1\_OUT1)}{dt} \tag{6.16}$$

$\frac{dV(POLE\_1\_OUT1)}{dt}$

55 Это условие обычно справедливо, поскольку *RADO* устанавливается DC дифференциальным усилением в макро-модуле POLE1.

Кроме того, если установлено эквивалентным скорости нарастания

OP-AMP, тогда ток, заряжающий *CP1,* будет ограничен максимально разрешенным. На рис. 6.9 GMP1 – это 1 S.

Следовательно, разница напряжений V (IN–P1) – V (IN–N1)

должна быть установлена в $\frac{1}{2\pi * GBP} * \frac{dV(POLE\_1\_OUT1)}{dt}$.

Это выполняется сетью SLEWRT, показанной на рис. 6.22, где

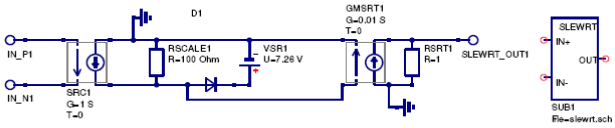


Рис. 6.22. Макро-модель OP AMP скорости нарастания

1. *RSCALE1* = 100 Ω = Масштабирующее сопротивление (к-т x 100).
2. *SRC1* G = 1 S.
3. *VSR1* = V1.
4. *GMSRT1* G = 0.01 S. (к-т = 1/100).
5. *RSRT1* =1Ω.

И

1. V 1 = $\frac{100 * Positive\_slew\_rate}{2\pi * GBP} - 0.7V$

2. V 2 = $\frac{100 * Negative\_slew\_rate}{2\pi * GBP} - 0.7V$

3. Параметры диода: IS=1e-12 IBV=20mA BV=V1+V2, другие по умолчанию.

Типовые значения для UA741 OP AMP:

1. Положительная-скорость-нарастания = Отрицательная-скорость-нарастания = 0.5V/ μ S.
2. V 1= V 2 = 7.25V.

Масштабирование используется для модели скорости нарастания, чтобы позволить использовать большее напряжение в фиксирующей цепи. Увеличение напряжения уменьшает ошибки, благодаря прямому смещению напряжения на (p-n) переходе. Текущее ограничение – результат фиксации напряжения через резистор *RSCALE1* с диодом. Диод работает, как стабилитрон, и сохраняет одно нелинейное соединение по

сравнению с общепринятыми фиксирующими цепями. Выходное звено цепи SLEWRT удаляет внутреннее масштабирование, передавая полное усиление модулю.

Цепь на рис. 6.23 демонстрирует ограничения скорости нарастания на OP AMP представление переходных процессов. Три идентичных OP AMP схемы инверторов питаются от общего входного источника сигнала 10 kHz AC. Управляемый напряжением источник напряжения используется для усиления входного сигнала для второй и третье цепи. Три входных сигнала имеют (1) 5 V пиковые, (2) 10 V пиковые и (3) 15 V пиковые, соответственно. Входные и выходные графики (waveforms) для этих цепей иллюстрирует рис. 6.24. Эффект ограничения скорости нарастания на большом сигнале представления переходного процесса ясно демонстрируется этими кривыми. В случае входного сигнала 15 V (пиковое значение) выходной сигнал (vout3.Vt) имеет наклон, приблизительно, 0.5 V на μ S.

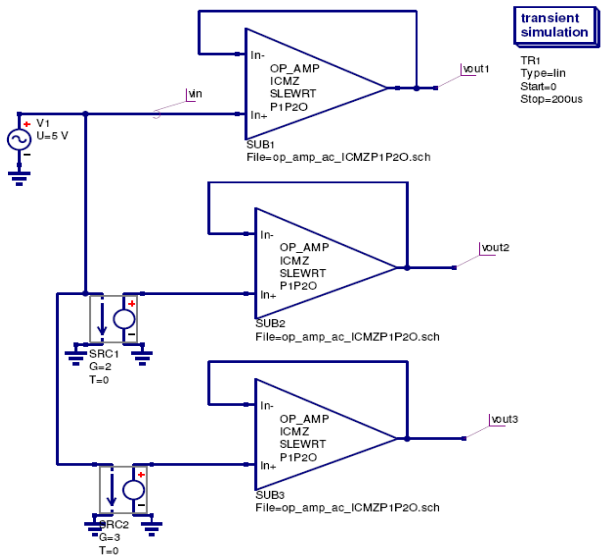


Рис. 6.23. Тестовая схем для скорости нарастания OP AMP

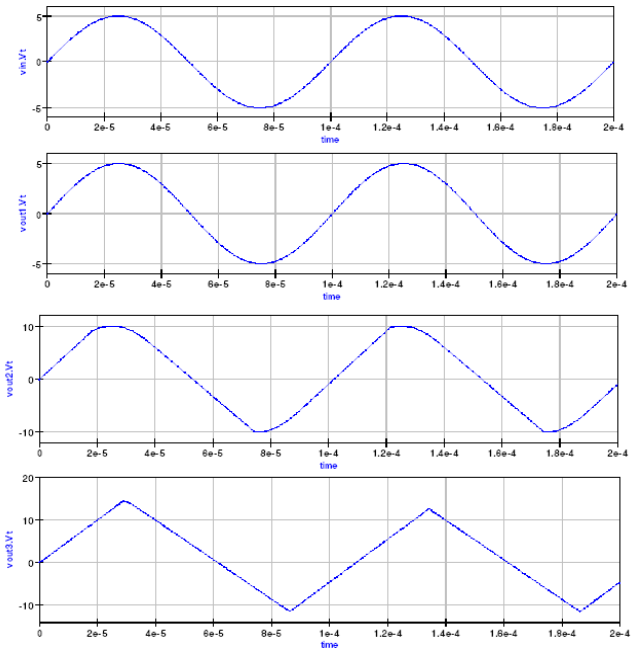


Рис. 6.24. Диаграммы симуляции скорости нарастания OP AMP для схемы рис. 6.23

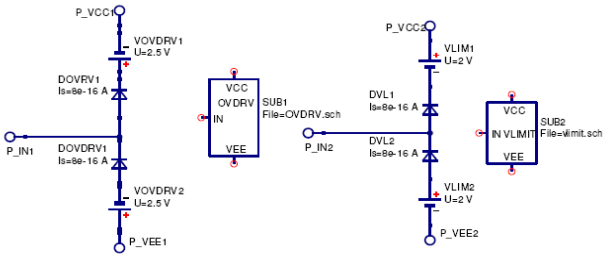


Рис. 6.25. OP AMP перегрузки и ограничения выходного напряжения макро-модели

6.8.2 Моделирование OP AMP перегрузки и ограничения выходного напряжения

Большие переходные сигналы могут перегружать OP AMP, приводя к тому, что выходное напряжение насыщается. После прекращения перегружающего сигнала OP AMP нужно конечное время для восстановления[56] и возвращения к нормальному

56 Время восстановления после перегрузки для OP AMP – это время, требуемое для того, чтобы выходное напряжение перешло к номинальному значению из условий насыщения. Типичное значение находится в пределах μS.

линейному поведению цепи. При насыщении выходное напряжение сжимается до напряжения (по плюсу или минусу) близкого к значениям питающих шин. Свойства перегрузки и сжимания напряжения для OP AMP взаимосвязаны и макро-модели для обоих эффектов следует добавить к модели OP AMP, когда симулируются перегрузочные характеристики OP AMP. Однако в при большинстве симуляций схем макро-модель перегрузки может быть опущена без потери функциональности или точности.

Эффект перегрузки сигналов может моделироваться цепью сжимания напряжения, которая берет в расчет время восстановления OP AMP после перегрузки по напряжению. Этот дополнительный элемент сжимает выход модуля *POLE1* на уровне выше OP AMP DC питающего напряжения. Общий эффект перегрузочной цепи – задержка восстановления линейного поведения цепи, когда перегружающий сигнал пропадает. В противоположность модулю перегрузки модуль ограничения выходного напряжения сжимает выходное напряжение до напряжения близкого к уровню питающего напряжения, обрезая выбег любого выходного напряжения выше уровня питающего. Рис. 6.25 иллюстрирует макро-модели для перегрузочной и ограничивающей выходное напряжение моделей, где

1. *VOVDR1* = 2.5 V = (Положительная скорость нарастания)*(Время восстановления усилителя).
2. *VOVDR2* = 2.5 V = (Отрицательная скорость нарастания)*(Время восстановления усилителя).
3. *VLIM1* = 2.0 V=(+напряжение питания)-(максимум положительного выходного напряжения)+1V.
4. *VLIM2* = 2.0 V = (-напряжения питания)-(максимум отрицательного выходного напряжения)+1V.
5. Параметры диода: Is = 8e-16 A, другие по умолчанию.

Типичные значения для UA741 OP AMP:

1. Время восстановления усилителя 5 μ S.
2. + питающего напряжения = 15 V.
3. - питающего напряжения = -15 V.
4. Максимум положительного выходного напряжения = 14 V.
5. Максимум отрицательного выходного напряжения = -14 V.

Тестовая цепь дана на рис. 6.26, иллюстрирующем эффекты перегрузки и сжимания выходного напряжения на буферной схеме единичного усиления. Тестовый входной сигнал – сигнал 1 kHz со следующими напряжениями (1) vin1 = 10 V пиковых, (2) vin2 = 18 V пиковых и (3) vin3 = 22 V пиковых. Соответствующие выходные диаграммы показаны на рис. 6.27. Показано, что увеличение сигналов перегрузки приводит к удлинению времени восстановления OP AMP, прежде, чем усилитель возвращается к линейному поведению.

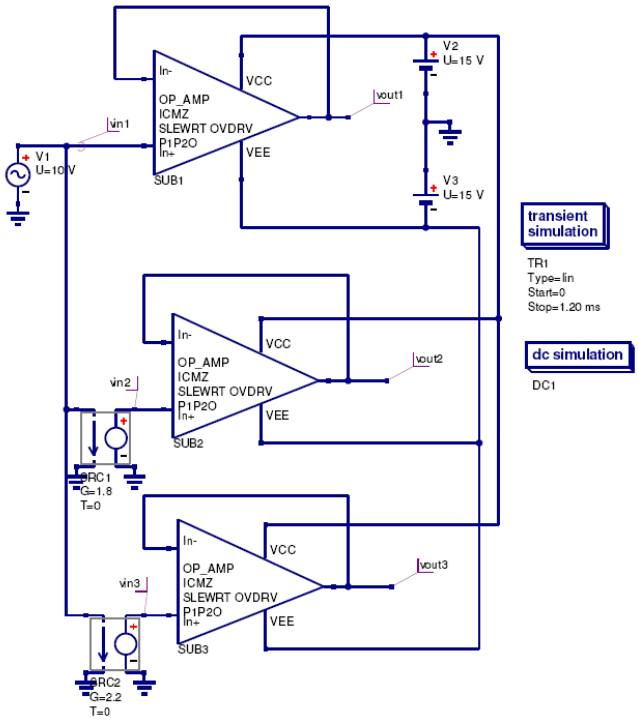


Рис. 6.26. Тестовая схема OP AMP перегрузки и ограничения выходного напряжения

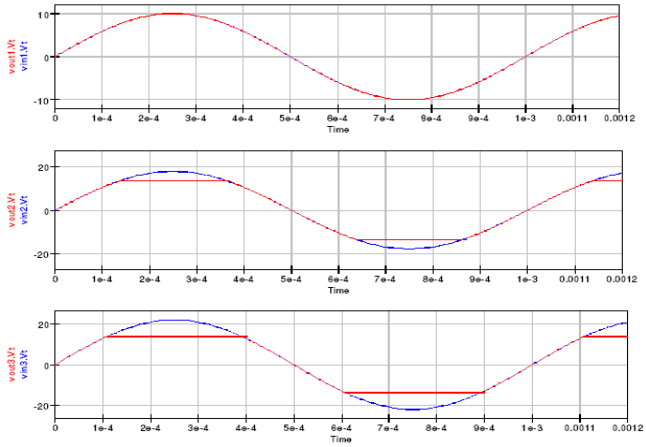


Рис. 6.27. Графики вывода OP AMP перегрузки и ограничения выходного напряжения для цепи, показанной на рис. 6.26

6.8.3 Моделирование ограничений OP AMP выходного тока

Большинство OP AMP общего назначения имеет цепи в схеме выхода для защиты устройства от токов перегрузки, создаваемой закорачиванием выходного вывода на землю или некоторых других ситуаций, где большие токи проходят через выходной каскад OP AMP. Электрическая сеть, показанная на рис. 6.28, работает как ограничитель тока: ток, проходящий между выводами P_IN1 и P_OUT1, отслеживается управляемым током генератором напряжения HCL1. Выход напряжения генератора HCL1 последовательно включен с управляемым напряжением генератором ECL1. Соединение этих генераторов сделано в противоположной полярности. Следовательно, когда ток нагрузки достигает максимума, допускаемого OP AMP, либо диод DCL1, либо DCL2 начинает сжимать выходное напряжение OP AMP, предупреждая увеличение выходного тока. Параметры для макро-модели ограничителя тока даются следующие

1. *RDCL1* = 100 MΩ = «Фальшивый» резистор.
2. *ECL1* G = 1.
3. *HCL1* G = 36Ω = 0.9 V/(максимум выходного тока A).
4. Параметры диода Is = 1e-15 A, другие по умолчанию.

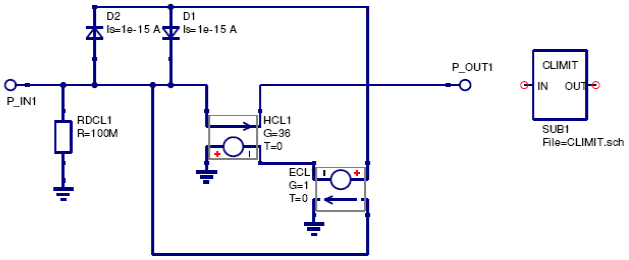


Рис. 6.28. Макро-модель OP AMP ограничителя выходного тока

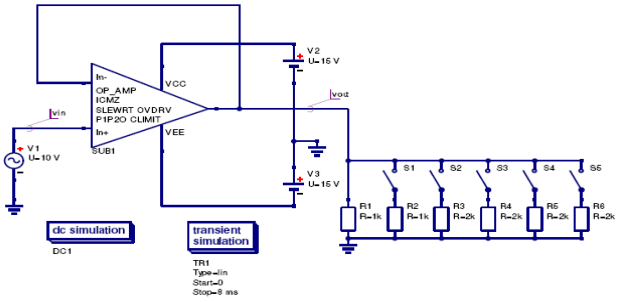


Рис. 6.29. Тестовая цепь OP AMP ограничителя выходного тока

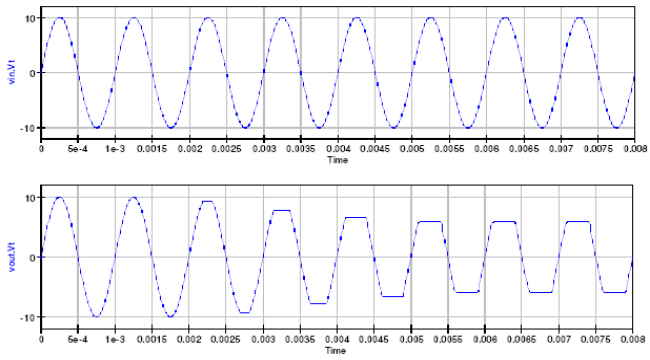


Рис. 6.30. Графический вывод симуляции для цепи рис. 6.29

Parameter	UA741	OP27	OP42	OPA134	AD746	AD826
Offset voltage (V)	7e-4	30e-6	4e-4	5e-4	3e-4	5e-4
Bias current (A)	80e-9	15e-9	130e-12	5e-12	110e-12	3-3e-6
Offset current (A)	20e-9	12e-9	6e-12	2e-12	45e-12	25e-9
Differential input res. (ohm)	2e6	4e6	1e12	1e13	2e11	300e3
Differential input cap. (F)	1.4e-12		6e-12	2e-12	5.5e-12	1.5e-12
Avd(0) dB	106	125	120	120	109	75
fp1 (Hz)	5	6	20	5	0.25	10e3
fp2 (Hz)	3e6	17e6	20e6	10e6	35e6	100e6
CMRR(0) dB	90	125	96	100	85	100
fcm (Hz)	200	2e3	100e3	500	3e3	2e3
GBP (Hz)	1e6	8e6	10e6	8e6	13e6	35e6
Rout (ohm)	75	70	50	10	10	8
Slew rate (V per micro sec.)	0.5	2.8	50	20	75	300
Overdrive recovery time (S)	5e-6		700e-9	0.5e-6		
DC supply current (A)	1.4e-3	2.5e-3	5.1e-3	4e-3	7e-3	6.6e-3
Short circuit output current(A)	34e-3	32e-3	30e-3	40e-3	25e-3	90e-3
Common-mode input res. (ohm)	1.3e8	2e9		1e13	2.5e11	
Common-mode input cap. (F)				5e-12	5.5e-12	

Таблица 6.1. Типичные параметры OP AMP, взятые из справочных данных устройства

Типичные значения для UA741 OP AMP тока короткого замыкания – 34 mA при 25^0 C.

Рис. 6.29 и 6.30 показывают простую тестовую цепь ограничителя тока и результирующие графические выводы. В этих тестах время симуляции схемы управляет выключателями, уменьшающими резисторы нагрузки с интервалом в 1 mS. Когда ток нагрузки достигает, примерно, 34 mA выходное напряжение сжимается, предупреждая дальнейшее увеличение тока в нагрузке.

6.9 Получение параметров макро-модели OP AMP из публикуемых данных устройства

Модульная макро-модель OP AMP имеет одно очень явное преимущество по сравнению с другими моделями усилителей, именно, что она может получать параметры макро-модели непосредственно из общего набора характеристик, который можно найти у большинства производителей в справочных данных. Данные, показанные в таблице 6.1, являются типичными значениями, находящимися в справочных данных OP AMP. В случаях, когда некоторые параметры не даны, для начала можно использовать значения, полученные из справочных данных аналогичного устройства. Значения элементов макро-модели затем вычисляются по уравнениям, приведенным в предыдущих разделах этого руководства. Как правило успеха, хорошей практикой служит проверка каждого блока в модульной макро-модели, прежде чем будет сконструирована полная макро-модель OP AMP.

6.10 Более сложные примеры разработки

В этих разделах представлены два больших примера разработок. Что должно продемонстрировать характеристики разных OP AMP макро-моделей, представленных в предыдущем тексте, и попытаться дать читателям руководство по коррекции модели при выборе конкретной симуляции.

6.10.1 Пример 1: Фильтр переменного состояния, разработка и симуляция

Цепь, данная на рис. 6.31 – это фильтр переменного состояния, который одновременно генерирует полосовой, высоких частот и низких частот отклики. Цепь состоит из OP AMP сумматора и двух интегрирующих цепей и требует трех OP AMP, двух конденсаторов и некоторого количества резисторов. Выбор типа OP AMP для успешной работы с этим фильтром критичен, поскольку устройства с большим смещением напряжения будут приводить к насыщению интеграторов и схема не будет функционировать корректно. Для работы ниже 20 kHz OP27 – это хороший выбор OP AMP, поскольку у них низкое напряжение смещения в пределах μ V. В этой симуляции обе характеристики, и DC, и переходная для малого сигнала AC, нужны, чтобы проверить разработку фильтра, следовательно макро-модель AC с параметрами

DC, заложенными во входной каскад, позволят аккуратно моделировать представление фильтра[57]. Вставленный на рис. 6.31 список DC выходных напряжений для каждого каскада OP AMP, показывает, что интеграторы не насыщаются. Разработка фильтра переменного состояния использует следующие уравнения:

1. Из принципа суперпозиции следует

$$vhp = -\frac{R1}{R6}vin - \frac{R1}{R7}vlp + \left(1 + \frac{R1}{R7 \parallel R6}\right)\frac{R4}{R4 + R5}vbp \qquad (6.17)$$

Когда R1= R6= R7

$$vhp = -vin - vlp + \frac{3R4}{R4 + R5}vbp \qquad (6.18)$$

2. Также

$$vbp = -\frac{1}{j\frac{f}{f_0}}vhp \qquad (6.19)$$

где

$$f_0 = \frac{1}{2\pi R_2 C_1} = \frac{1}{2\pi R_3 C_2} \qquad (6.20)$$

3. Похожим образом

$$vlp = -\frac{1}{j\frac{f}{f_0}}vbp = -\frac{1}{(\frac{f}{f_0})^2}vhp \qquad (6.21)$$

4. Следовательно

$$\frac{vhp}{vin} = \frac{(\frac{f}{f_0})^2}{1 - (\frac{f}{f_0})^2 + (\frac{j}{Q})(\frac{f}{f_0})} \qquad (6.22)$$

Где

$$Q = \frac{1}{3}(1 + \frac{R5}{R4}) \qquad (6.23)$$

5. Также

57 Величина выходных сигналов от фильтра должна быть также проверена, чтобы удостовериться, что эти сигналы не превышают напряжений питания.

$$\frac{vbp}{vin} = \frac{j\frac{f}{fo}}{1 - (\frac{f}{fo})^2 + (\frac{j}{Q})(\frac{f}{fo})} \qquad (6.24)$$

6. Также

$$\frac{vlp}{vin} = \frac{-1}{1 - (\frac{f}{fo})^2 + (\frac{j}{Q})(\frac{f}{fo})} \qquad (6.25)$$

Принимая f0 = 1 kHz и требуемую ширину полосы пропускания полосового фильтра в 10 Hz, при установке R1= R6= R7 = 47kΩ и C1= C2=2.2nF , вычисление дает R2= R3 = 72.33k[58] В этой разработке Q =1k/10 = 100. Следовательно установка R4=1kΩ даст R5 = 294kΩ (1 % отклонение).

Графики вывода симуляции для полосового фильтра даны на рис. 6.32[59]. Когда фактор схемы Q уменьшается к нижним значениям, другие выходы фильтра работают, как традиционные фильтры верхних и нижних частот. Результаты симуляции для Q фактора единица показаны на рис. 6.33.

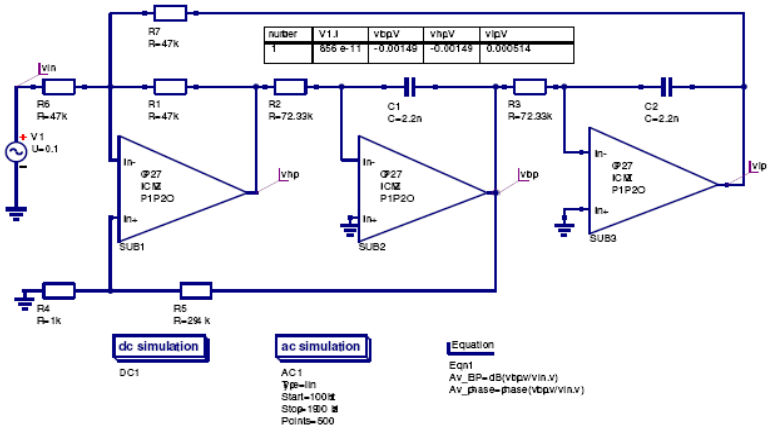


Рис. 6.31. Трех-каскадный OP AMP перестраиваемый фильтр

58 Значения R2 и R3 нуждаются в уменьшении, если центральная частота фильтра и полоса требуют большей точности.

59 Заметьте, что входной сигнал vin был установлен в 0.1V (пиковое значение). Цепь имеет Q фактор 100, что означает, что выходное напряжение пропускания 10V (пиковое). Входные сигналы амплитудой много больше, чем 0.1V, похоже, приведут выходной сигнал в насыщение, когда питающее напряжение ±15V.

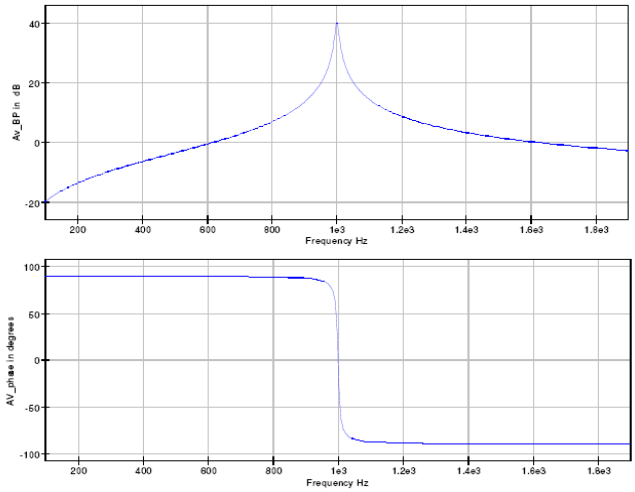


Рис. 6.32. Графический вывод симуляции для схемы рис. 6.31

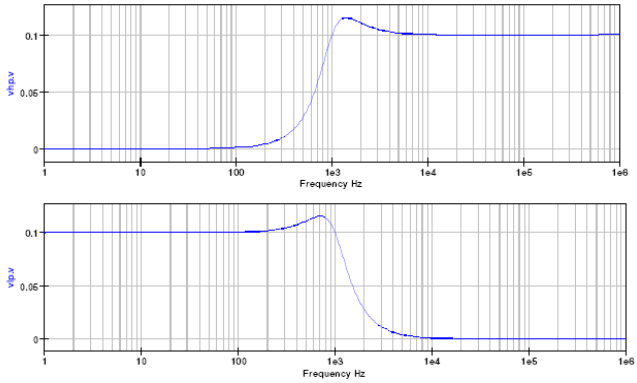


Рис. 6.33. Низкочастотная и высокочастотная характеристики для Q = 1, R5=2k.

6.10.2 Пример 2: Генератор синусоидального сигнала с осциллятором, использующим мост Вина

Синусоидальный осциллятор с мостом Вина стал классическим, благодаря его простоте и малым искажениям. Это идеальный объект для демонстрации свойств OP AMP макро-моделей и действительного представления о симуляторах схем. Показанное на рис. 6.34 – это базовый осциллятор с мостом Вина, который состоит из единственного OP AMP с цепями положительной и отрицательной обратной связи. Уравнения для этой цепи

1. Не инвертирующий усилитель.

$$\frac{vout}{v+} = 1 + \frac{R3}{R4} \qquad (6.26)$$

2. К-т обратной связи.

$$b = \frac{vout}{v+} = \frac{1}{3 + j(\frac{f}{f_0} - \frac{f_0}{f})} \qquad (6.27)$$

Где $f_0 = \frac{1}{2\pi R1C1} = \frac{1}{2\pi R2C2}$

3. Петлевое усиление
   Петлевое усиление осциллятора bA_v должно быть равно единице для стабильной генерации. Следовательно,

$$bA_v = \frac{1 + \frac{R3}{R4}}{3 + j(\frac{f}{f_0} - \frac{f_0}{f})} \qquad (6.28)$$

   Кроме того, при $f = f0$,

$$bA_v = \frac{1 + \frac{R3}{R4}}{3} \qquad (6.29)$$

   Задание R3/R4 немного больше двух приводит к тому, что осцилляция начинается и увеличивается по амплитуде в каждом цикле осцилляции. Если же R3/R4 меньше двух, осцилляция никогда не начнется или уменьшится до нуля.

Рис. 6.35 показывает набор графических выводов для осциллятора с мостом Вина. В этом примере OP AMP моделируется с использованием OP27 AC макро-модели. Это было сделано намеренно, чтобы продемонстрировать, что происходит при плохом выборе OP AMP модели. Частота осцилляции 10 kHz с обоими конденсаторами обратной связи и резисторами, имеющими равные значения. Заметьте, что выходное напряжение осцилляции продолжает увеличиваться с увеличением времени, пока его значение не приблизится к пределу, установленному практическим напряжением питания OP AMP. Нижняя из двух кривых на рис. 6.35 иллюстрирует спектр частот выходного сигнала осциллятора. Данные для этой кривой были сгенерированы с помощью функции Time2Freq. Добавление скорости нарастания и ограничения напряжения к макро-модели OP27 ограничит выходное напряжение осциллятора (двойной амплитуды) значениями питающего напряжения OP AMP. Диаграммы для этой симуляции показаны на рис. 6.36. При анализе данных реакции переходных процессов с использованием функции Time2Freq благоразумно ограничить анализ диапазоном отклика, где выходной график достигает устойчивого состояния, иначе частотный спектр будет включать эффекты обязанные возрастанию, или спаду переходного процесса. Сеть ограничения напряжения сжимает выходное напряжение

осциллятора, ограничивая его размах ниже напряжения питания OP AMP. Сжимание очень хорошо видно на рис. 6.36. Заметьте также, что выходной график искажен, и что нет больше чисто синусоидального графика с частотой 10 kHz. Нечетные гармоники отчетливо видны, а основная частота уменьшена из-за искажений насыщения сигнала. В практических осцилляторах с мостом Вина выходной график должен быть чисто синусоидальным с нулевыми или маленькими гармоническими искажениями. Один из путей добиться этого, изменить усиление OP AMP с помощью изменения уровня сигнала: когда выходной сигнал увеличивается, тогда *Av* уменьшается, или когда уровень выходного сигнала уменьшается, *Av* увеличивается. В любое время параметры цепи изменяются, чтобы поддержать условие $bAv = 1$. Цепь, показанная на рис. 6.37, использует два диода и резистор для автоматического изменения OP AMP усиления в петле обратной связи с изменением уровня сигнала. Рис. 6.38 показывает соответствующие выходные графики для цепи с мостом Вина и автоматическим управлением усилением. Изменение значения резистора R5 приводит к тому, что амплитуда выходного напряжения осциллятора стабилизируется при разных значениях, уменьшение R5 также уменьшает vout. Версия осциллятора с мостом Вина и автоматическим управлением усилением также уменьшает величину гармонических искажений, производимых осциллятором. Это ясно видно на рис. 6.38. Изменение частоты осциллятора может быть достигнуто либо изменением величины конденсатора, или значения резистора в цепи обратной связи b. Для демонстрации того, как это может быть сделано с помощью Qucs, посмотрите на схему рис. 6.39. На этой схеме переключатели, управляемые по времени, изменяют значения обоих конденсаторов в процессе симуляции. Записанные выходные графики для этой цепи показаны на рис. 6.40.

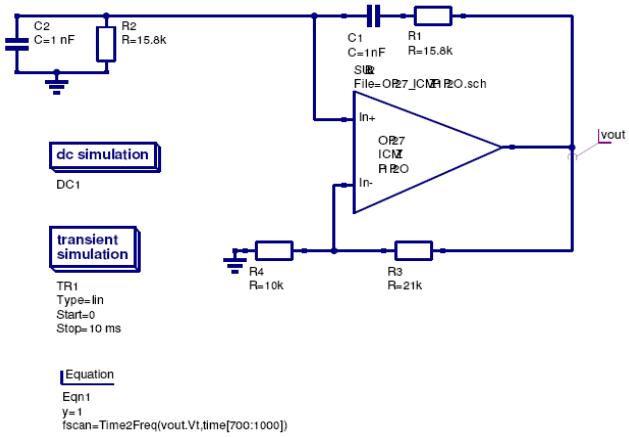


Рис. 6.34. Классический синусоидальный осциллятор с мостом Вина

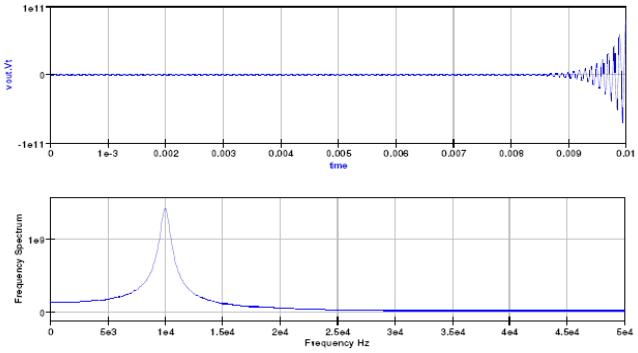


Рис. 6.35. Графики симуляции для цепи, показанной на рис. 6.34 – OP27 AC макро-модель

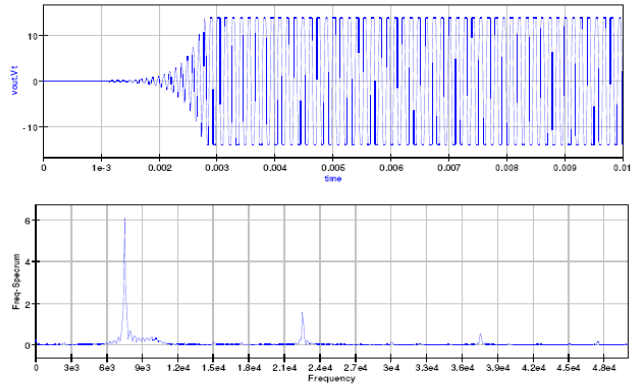


Рис. 6.36. Диаграммы симуляции для цепи, показанной на рис. 6.34 – OP27 AC + скорость нарастания + vlimit макро-модель

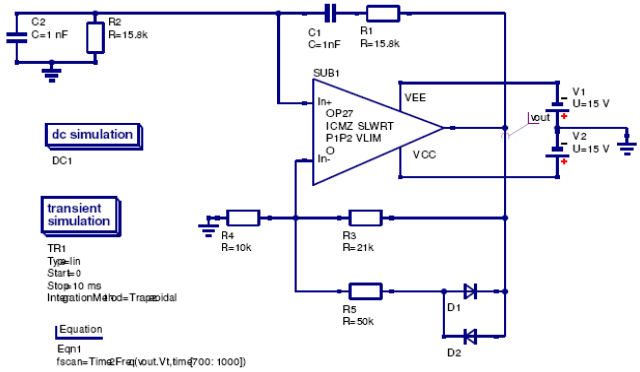


Рис. 6.37. Осциллятор с мостом Вина и АРУ

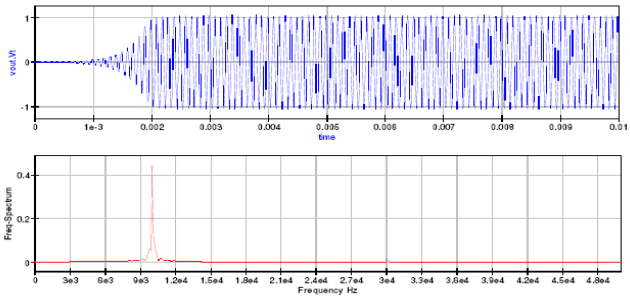


Рис. 6.38. Графики симуляции для цепи рис. 6.37 – OP27 AC + скорость нарастания + vlimit макро-модель

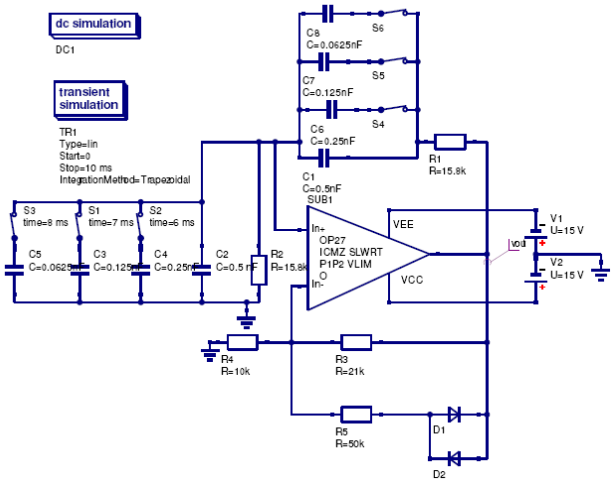


Рис. 6.39. Осциллятор с мостом Вина и управлением частотой переключением конденсаторов

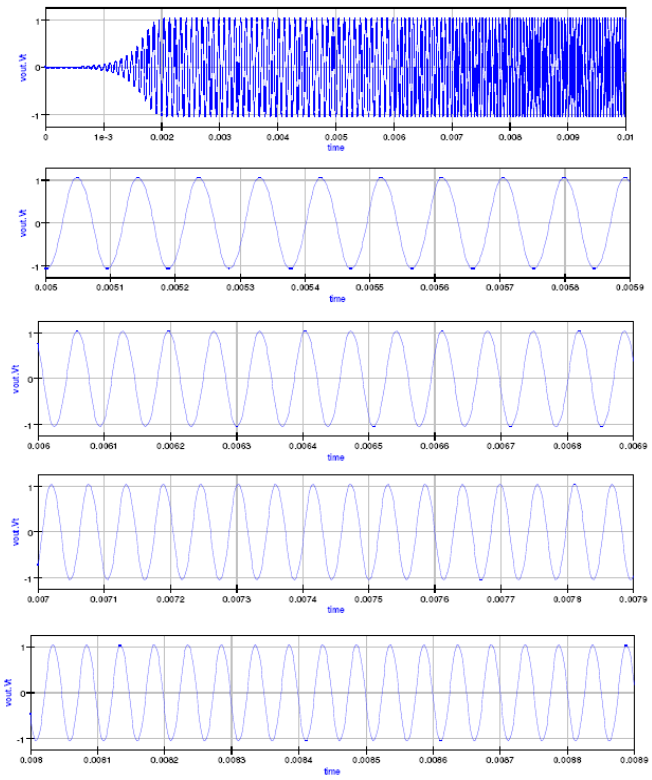


Рис. 6.40. Диаграммы симуляции цепи рис. 6.39 – OP27 AC + скорость нарастания + vlimit макро-модель

6.11 Заключительные замечания

При написании этого руководства я попытался продемонстрировать, как можно конструировать практические модели операционных усилителей, используя базовые концепции электроники и ряд встроенных моделей Qucs, включенных в выпуск 0.0.9. Модульная макро-модель OP AMP была намерено выбрана, как фундаментальная для руководства по двум причинам. Во-первых, Qucs выпуска 0.0.9 достаточно подготовлен для симуляции таких моделей, и, во-вторых, параметры, которые определяют операции макро-модели, могут легко вычисляться из информации, предоставляемой в справочных данных. Это руководство должно рассматриваться прежде всего, как рабочее пособие, поскольку некоторые важные свойства OP AMP не включались в макро-модели, описанные в предыдущих разделах. Это, например, отклонения питающего напряжения и шумовые свойства. В будущем, если пользователи Qucs найдут эти заметки полезными, и, фактически, если отклики на них будут положительными, я предполагаю обновить руководство по моделированию OP AMP. Другие важные разделы будут также добавлены, когда модели нелинейных управляемых источников из списка to-do будут добавлены Stefan в Qucs. Тогда появится возможность расширить ряд моделей, которые сможет симулировать Qucs, включением модели Boyle и более развитых вариантов, которые часто включаются в справочные данные производителей устройств. Мои благодарности Dr. David Faulkner[60] за его помощь и поддержку во все время нашей работы над многими концепциями, сформировавшими базис этого руководства. И опять, большое спасибо Michael Margraf и Stefan Jahn за всю их помощь и ободрение, пока я писал это руководство и проверял множество примеров, включенных в него.

60 Department of Computing, Communications Technology and Mathematical Science, London Metropolitan University, UK.

7 Моделирование таймера 555

7.1 Введение

Таймер 555 был разработан Hans R. Camenzind в 1970[61] и впервые произведен Signetics в период 1971-1972[62] . Устройство первоначально называлось «IC машина времен» и имело спецификацию SE555/NE555. За последние 30 с небольшим лет более десятка разных компаний производителей полупроводниковых микросхем выпускали 555, сделав ее самой популярной IC всех времен[63]. По сей день он еще активно используется в широком диапазоне схемных приложений.

Таймер 555 один из первых примеров смешанных IC цепей, которые включают и аналоговые, и цифровые компоненты. Первоначальное назначение таймера 555 – генерация одиночных импульсов точной длительности или работа в режиме осциллятора. Добавлением одного или двух дополнительных резисторов и одного конденсатора устройство превращается в одно-вибратор или мультивибратор.

Таймер 555 – устройство, которое трудно симулировать. Во время схемных операций он быстро переключается между двумя разными DC состояниями[64]. Столь быстрые изменения могут стать причиной ошибок сходимости симулятора DC и анализатора переходных процессов. Большинство популярных симуляторов включают некоторую форму модели таймера 555, либо встроенную, либо как подсхему, которая функционирует до некоторой степени. Эти модели обычно включают сколько-то p-n переходов и нелинейные управляемые источники, делающие время симуляции больше, чем с более простыми моделями. Сердце таймера 555 – это два компаратора и триггер с установкой/сбросом. Блок-диаграмма основных функциональных элементов, составляющих таймер 555, показана на рис. 7.1.

Текущий выпуск Qucs не включает модель для таймера 555. Цель работы, представленной в этих заметках руководства, была разработать рабочую модель таймера 555, которая эффективно симулируется и базируется только на схемных компонентах, реализованных в Qucs 0.0.10. Кроме того, при разработке Qucs 555

61 См. ”The 555 Timer IC. An interview with Hans Camenzind -The designer of the most successful integrated circuit ever developed”, http://semiconductormuseum.com/Transistors/LectureHall/Camenzind/

62 Сейчас часть организации Philips.

63 Сегодняшний каталог производителей показывает, что таймер 555 популярен как никогда, например, Samsung (Korea) произвела более одного миллиарда устройств в 2003; см. Wikipedia статью на http://en.wikipedia.org/

64 Обычно между землей и напряжением, подключенным к шине питания VCC.

модели в каждом случае делались попытки свести количество p-n переходов к минимуму, придавая модели и простоту, и уменьшая время симуляции схем. Стержнем такого подхода становится упрочившаяся техника макро-моделирования, где сигналы от выводов таймера правильно моделируют реальные сигналы устройства, но внутренние сигналы макро-модели подчас никак не соотносятся с аналогичными в реальном устройстве. Изнутри макро-модель просто пропускает информацию входных и выходных сигналов, в нужном формате, на выходные выводы устройства. И нет другого пути попытаться симулировать реальную схему таймера 555.

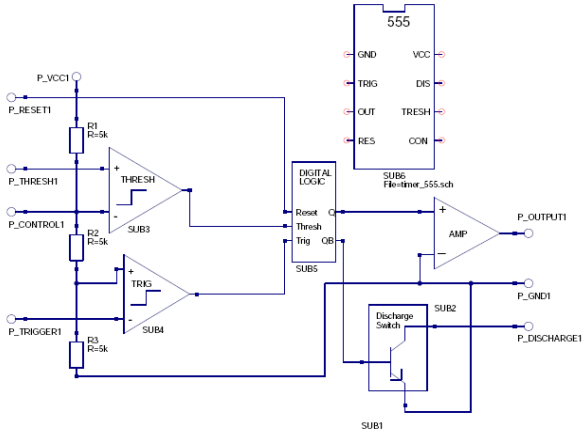


Рис. 7.1. Функциональная блок-диаграмма таймера 555

7.2 Модель Qucs таймера 555

Рис. 7.1 показывает новую модель Qucs таймера 555. В этой модели каждый из основных функциональных блоков был разделен на макро-модельную подсхему, группирующую похожие типы компонент вместе. Существенно, что модель включает только стандартные компоненты Qucs, которые работают вместе для создания корректных выходных сигналов путем тщательного выбора пороговых параметров, ограничений напряжения, логических уровней и времен фронтов. Эти заметки больше концентрируются на объяснении структуры и параметров подсхем макро-модели, которые формируют модель таймера 555, чем на описании функционирования устройства[65]. Таймер 555 – устройство с 8 выводами:

- Вывод 1 Земля [GND] – наибольшее отрицательное напряжение питания, подключенное к устройству, обычно это общий провод, ground (0V).
- Вывод 2 Триггер [TRIG] – входной вывод нижнего компаратора.

65 Хорошее руководство по работе таймера 555 можно найти на http://www.uoguelph.ca/~antoon/gadgets/555/555.html

Используется для установки RS-защелки.

- Вывод 3 Выход [OUT] – вывод выходного сигнала таймера 555.
- Вывод 4 Сброс [RES] – используется для сброса RS-защелки.
- Вывод 5 Управление [CON] – точка прямого доступа к узлу делителя (2/3)VCC. Используется для установки опорного напряжения для верхнего компаратора.
- Вывод 6 Порог [THRESH] – входной вывод верхнего компаратора. Используется для сброса RS-защелки.
- Вывод 7 Разряд [DIS] – вывод коллектора npn BJT переключателя. Используется для разряда внешнего время-задающего конденсатора.
- Вывод 8 VCC [VCC] – наибольшее положительное напряжение питания, подключенное к устройству, обычно это 5V, 10V или 15V.

7.2.1 Макро-модель переключающего компаратора

Входные выводы триггерного компаратора соединены между узлом делителя (1/3)VCC и выводом 2 корпуса устройства (TRIG). Входной переключающий сигнал, опускаясь ниже напряжения узла делителя (1/3)VCC, приводит к тому, что переключается выходное напряжение триггера, устанавливая RS-защелку в подсхеме цифровой логики. Это действие также вызывает установку выходного сигнала таймера 555 в высокое состояние. Вход триггера чувствителен к уровню сигнала. Переключение произойдет, если переключающий импульс удерживается чуть дольше длительности выходного импульса таймера 555. Цепь переключающего компаратора также имеет время сохранения в несколько микросекунд, ограничивая минимальный выходной импульс длительность около 10 μ S. DC ток, обычно связываемый с током переключения, проходит от вывода устройства 2 (TRIG) во внешнюю цепь. Он имеет типичное значение 500 nA, устанавливая верхний предел для резистора, который может быть подключен между выводом 2 и землей[66]. Диаграмма макро-модели цепи триггерного компаратора показана на рис. 7.2. Дифференциальный входной сигнал воспринимается операционным усилителем OP1. Его усиление задано в 1e6, давая разрешение входного дифференциального сигнала 1 μ V. OP1 выходное напряжение ограничено до ±1V. Заметьте, что верхний +1V уровень сигнала относится к сигналу логической «1». И, наконец, выходное напряжение триггерного компаратора имеет времена фронтов, задаваемые временной константой R1 C1. Эта цепь также добавляет задержку времени в макро-модель компаратора.

66 При VCC = 5V этот резистор, примерно, 3.3M.

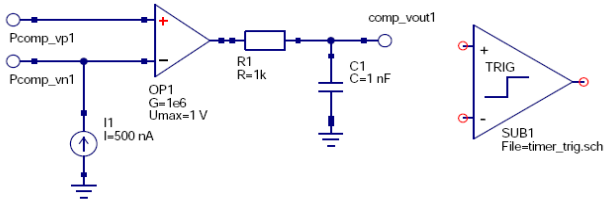


Рис. 7.2. Макро-модель триггерного компаратора

7.2.2 Макро-модель порогового компаратора

Макро-модель порогового компаратора показана на рис. 7.3. Она очень похожа на макро-модель триггерного компаратора; одна заметная разница есть в размере и направлении вывода 6 (THRES), порога DC тока, который обычно составляет 100nA и проходит к выводу 6 от внешней цепи[67]. Пороговый компаратор используется для сброса RS-защелки в блоке цифровой логики таймера 555, вызывая переход выхода таймера 555 в состояние низкого напряжения. Сброс происходит, когда сигнал, приложенный к внешнему выводу 6 (THRES), меняется снизу-вверх к напряжению узла делителя (2/3)VCC. Опять же, пороговый вход чувствителен к уровню.

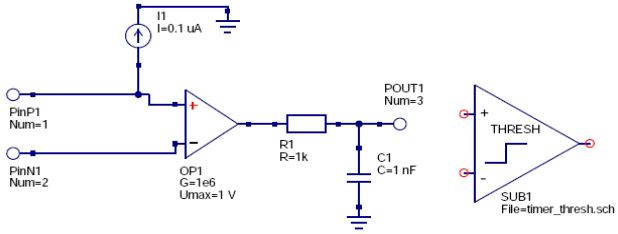


Рис. 7.3. Макро-модель порогового компаратора

Set (S)	Reset (R)	Q (P-Q1)	QB (P-QB1)	Notes
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

Таблица 7.1. Таблица истинности для SR-защелки, собранной с использованием вентилей NOR

67 Порог DC тока устанавливает верхний предел значения внешнего резистора, который может быть подключен между выводом 6 и питающим VCC – для VCC = 5V это приблизительно 16M, при VCC = 15 V значение увеличивается, примерно, до 20M.

7.2.3 Макро-модель цифровой логики

Макро-модель цифровой логики состоит из SR-защелки с дополнительной комбинацией вентилей на входе модели, см. рис. 7.4. Таблица истинности для SR-защелки приведена в таблице 7.1. Все вентили в макро-модели имеют логическую «1», установленную в 1V и логический «0», задаваемый 0V. RC временная цепь была добавлена к выходу каждого вентиля, обеспечивая конечное время фронтов, вместо значения по умолчанию Qucs равного нулю секунд[68]. Входные сигналы вентилей со значениями меньше, чем пороговое напряжение вентиля, (0.5V) считаются логическими сигналами «0». Логический сигнал «0» на выводе 4 (RES) таймера 555 также сбрасывает SR-защелку, вызывая переход выходного сигнала, вывод 3 (OUT), в низкое состояние. Сигнал сброса – сигнал перегрузки в том смысле, что он заставляет перейти выходной сигнал таймера в низкий уровень, независимо от сигналов на других выводах входов. Сброс имеет время задержки, приблизительно, 0.5 μ S, делая минимальную длительность импульса сброса около 0.5 μ S. Сигнал сброса инвертирован, а затем соединен по OR с выходным сигналом порогового компаратора.

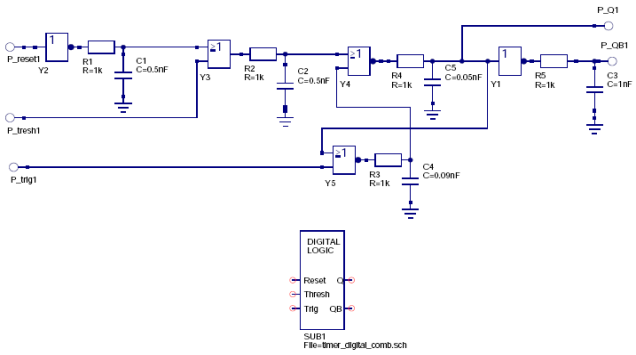


Рис. 7.4. Макро-модель цифровой логики

7.2.4 Макро-модель Выходного усилителя таймера 555

Иллюстрация на рис. 7.5 – это макро-модель выходного усилителя таймера. Это простая модель, созданная из блока усиления плюс резистор для представления выходного сопротивления таймера 555. Блок усиления по напряжению имеет значение заданное в 3.5 на рис. 7.5. Это значение, необходимое для масштабирования сигнала логической «1» к требуемому выходному напряжению выхода таймера, вывод 3 (OUT). Это значение корректируется только для питающего напряжения VCC,

68 В смешанных цепях симуляция переходных процессов может обнаруживать проблемы, когда устройства меняют состояние в ноль секунд, см. последние заметки с примечаниями по этой теме.

установленного в 5V, и должно меняться для других значений напряжения[69].

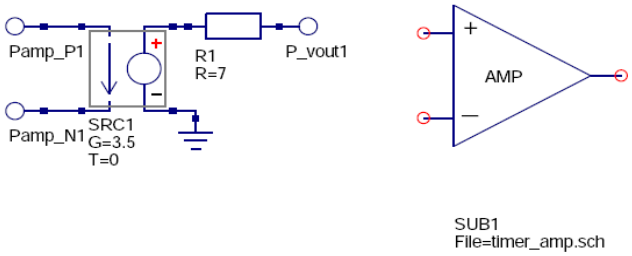


Рис. 7.5. Макро-модель выходного усилителя

7.2.5 Макро-модель коммутатора разряда

Макро-модель переключателя разряда показана на рис. 7.6. Подобно реальному таймеру 555 коммутатор разряда макро-модели базируется на npn транзисторе. Сигнал логической «1», приложенный к выводу `pin_control_in1` включает npn транзистор, приводя к тому, что путь от коллектора (таймер 555, вывод DIS) к земле становится низкоомным. И через эту ветвь внешний конденсатор таймера разряжается. Обратная картина наблюдается, когда входное управляющее напряжение становится логическим «0». В этом случае ветвь от коллектора к земле имеет очень большое сопротивление. Резистор R1 включен в макро-модель для ограничения npn тока базы, когда BJT включается. Аналогично, резистор R2 был добавлен в модель для ограничения тока разряда внешнего конденсатора[70].

69 В настоящее время Qucs не позволяет передавать параметры в подсхемы, делая затруднительным написание обобщенных макро-моделей. Добавление передачи параметров в подсхемы и вычисления значений компонент с использованием уравнений есть в списке to-do. Предлагаемые значения для усилительного каскада: (1) VCC = 5V, G = 3.5, (2) VCC = 10V, G = 8.5V и (3) VCC = 15V, G = 13.5. Эти значения усиления корректны для падения напряжения в каскадном выходе усилителя таймера 555.

70 Обычно внешний время-задающий конденсатор разряжается через резистор, последовательно включенный между коллектором и землей. Однако, если этот последовательный резистор очень мал, или фактически отсутствует, теоретически возможно для тока разряда стать очень большим, что, в свою очередь, приведет к ошибке сходимости DC или очень большим временам симуляции переходного процесса.

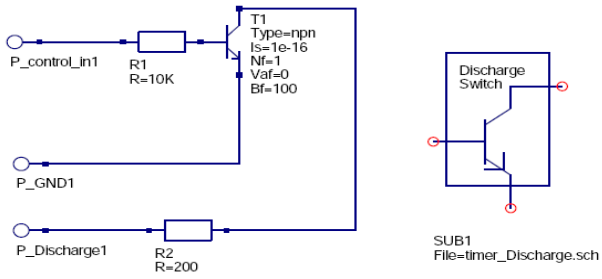


Рис. 7.6. Макро-модель коммутатора разряда