

УДК 004.054

В. В. СКЛЯР¹, В. С. ХАРЧЕНКО¹, А. С. ПАНАРИН²¹Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Украина²ПАО «НПП «Радий», Украина

ТЕСТИРОВАНИЕ И РАЗРАБОТКА ДИВЕРСНЫХ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ НА БАЗЕ ПЛИС С ИСПОЛЬЗОВАНИЕМ СРЕДЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

В статье предложен подход к моделированию и тестированию программного обеспечения контроллеров с использованием инструментальной среды ForSyDe, поддерживающей функциональное программирование. Получена функциональная модель подсистемы обработки дискретных сигналов, которая затем преобразована в VHDL-код и в цифровое устройство на базе программируемой логической интегральной схемы (ПЛИС). Предложены варианты реализации жизненного цикла системы на базе ПЛИС с использованием альтернативных версий для тестирования и многоверсионного функционирования.

Ключевые слова: ПЛИС, ForSyDe, функциональное программирование, IP-ядро, soft-процессор.

1. Анализ проблемы: разрыв между спецификацией и реализацией гетерогенных SoPC

Современные системы на программируемых кристаллах (System on Programmable Chips – SoPC) представляют собой многоядерные гетерогенные объекты, включающие, в том числе, эмуляторы микропроцессоров. Примером таковых являются процессорные интеллектуальные ядра (Intellectual Property Core – IP-Core) Nios фирмы Altera и Cortex фирмы Actel, предназначенные для имплементации в программируемые логические интегральные схемы (ПЛИС).

SoPC разрабатываются и верифицируются средствами императивного программирования, как обычная последовательная программная система. При этом не всегда уделяется внимание физической реализации, которая заключается в создании параллельной цифровой системы.

В то же время, спецификация требований к SoPC, как правило, имеет «параллельный» вид, включая требования к обработке входных и выдаче выходных сигналов, требования к внутренним и внешним коммуникационным связям, к диагностике и т.п.

Таким образом, имеет место разрыв (так называемый “gap”) между исходной спецификацией требований и средой разработки, а также между средой разработки и конечной реализацией. Каждый такой разрыв требует существенного преобразования информации, специфицирующей SoPC, из одной формы в другую, что, как известно, является источни-

ком ошибок и дефектов.

В принципе такой подход мог бы быть оправдан требованиями к высокому уровню абстракции описаний при быстром проектировании сложных систем.

Однако несовершенство интегрированных сред разработки SoPC (Integrated Development Environment – IDE) приводит к аппаратным внутрисхемным проблемам (рассинхронизация, «иглы», «гонки», внутреннее и внешнее тактирование). Как правило, такие проблемы зависят от конкретной реализации SoPC, т.е. являются специфическими, а не типовыми. В силу этого, фирменные руководства по разработке не содержат описания мер противодействия описанным проблемам, и разработчики имеют дело с недокументированными свойствами IDE. Таким образом, ухудшаются эффективность и качество процессов и продуктов разработки.

Одним из возможных подходов к решению проблемы “gap” в SoPC со встроенными процессорными ядрами является использование для таких ядер функционального программирования [1], как средства аналитического описания и последующей имплементации.

2. Применение среды ForSyDe в качестве инструментального средства для функционального программирования на языке Haskell

При использовании парадигмы функционального программирования работа процессора трактуется как вычисление значений функций в их мате-

матическом понимании, т.е. когда каждому значению элемента x из некоторого множества X ставится в соответствие единственный элемент y из множества Y [1]. Таким образом, функциональное программирование предполагает неизменность данных при вызове функции с одними и теми же аргументами. Такой подход является наиболее адекватным средством описания вычислений.

Однако на практике более распространенной является парадигма императивного программирования, которая описывает процесс вычислений как последовательность изменения состояний, связанных с изменением переменных. При этом функция понимается как разновидность подпрограммы (процедура, возвращающая параметры).

В основу теории функционального программирования легли труды американских математиков Карри Хаскелла (Curry Haskell) в части комбинаторной логики и Алонзо Черча (Alonzo Church) в части формальных λ -исчислений. На основе этих трудов в 1958 г. Джон Маккарти (John McCarthy) разработал первый функциональный язык программирования LISP. Среди наиболее заметных достижений в области внедрения функциональных языков указывают две системы промышленного уровня с обширной пользовательской средой и поддержкой – Ericsson Erlang и Harlequin ML Works. Кроме того, для отдельного класса вычислений используется язык Haskell, последними стандартами которого являются Haskell-1998 и Haskell-2010.

В качестве достоинств функционального программирования называется изящность, эффективность, детерминированность и т.п. Недостатки вытекают из недостаточной популярности функциональных языков, из-за чего для них не обеспечиваются достаточные совместимость, переносимость, инструментальная поддержка и т.п. [2].

Примером применения функционального программирования для моделирования цифровых систем является инструментальное средство ForSyDe (Formal System Design), разработанное в Королевском техническом университете (КТН, Стокгольм, Швеция) [3,4].

Основная концепция ForSyDe заключается в решении проблемы “gap” путем создания методологии проектирования высокоуровневых абстракций, базирующейся на так называемом подходе трансформационного уточнения проекта (transformational design refinement). При этом система моделируется в виде сети взаимодействующих процессов, связанных между собой посредством сигналов. Процессы выполняют вычисления, преобразуя входные сигналы в выходные. Таким образом, коммуникационные и вычислительные функции отделены друг от друга. Применение абстрактного времени позволяет реали-

зовывать несколько моделей вычислений, основной из которых является синхронная модель [5].

Учитывая достаточную технологическую зрелость ForSyDe и фундаментальность теоретических концепций, представляется целесообразным его использование для альтернативной разработки и верификации IP-Cores для SoPC, реализующих функции параллельных цифровых систем. Поскольку функциональное программирование дает удобное представление цифровых параллельных систем, при помощи ForSyDe можно представить SoPC или его часть (встроенное IP-Core) в диверсном исполнении. Если результаты функционирования основного и диверсного представления SoPC сойдутся, то результаты такой верификации признаются успешными. Диверсное представление SoPC может быть использовано в информационно-управляющих системах (ИУС) для защиты от отказов по общей причине и для усиления так называемой «защиты в глубину» (defense-in-depth) [5]. Такой подход позволил бы исправлять ошибки в проекте SoPC уже на начальных этапах его создания.

Целью исследования является анализ возможности применения ForSyDe для реализации концепции формирования альтернативных IP-Cores, позволяющих выполнять независимую верификацию, а также диверсную реализацию для SoPC. В настоящем исследовании в качестве SoPC рассматриваются промышленные контроллеры на базе программируемых логических интегральных схем (ПЛИС).

3. Анализ функционирования модуля ввода дискретных сигналов на базе ПЛИС

В качестве моделируемого проекта был выбран модуль ввода дискретных сигналов. Задачи обработки дискретных сигналов являются типичными для промышленных систем управления. В частности, рассматриваемый модуль применяется в цифровой информационно-управляющей платформе НПП «Радий» для систем безопасности АЭС [6,7]. Основное назначение модуля – считывание цифровых данных с портов ввода и их передача в модуль логического управления (ЛМ), а также передача диагностической информации в диагностический модуль (ДМ).

В качестве программируемого компонента применяются ПЛИС фирмы Altera. Электронные проекты ПЛИС разрабатываются в IDE Altera Quartus II, с применением процессора базирующегося на ядре Altera Nios.

Микропроцессор Nios относится к классу микропроцессоров для встроенного применения, то есть его основная задача — работа в режиме реального

времени. Для этого микропроцессор должен оперативно реагировать на прерывания, а также переключаться с одной задачи на другую. При применении Nios в SoPC появляется дополнительная возможность – реализация команд и аппаратных сопроцессоров, определяемых и разрабатываемых пользователем.

Процессор Nios, является RISC-процессором с конвейерной обработкой команд, большинство команд которого выполняется в течение одного цикла, задаваемого тактовой частотой. Система команд Nios ориентирована на встроенные прикладные программы. Существует возможность выбора разрядности ядра Nios CPU между 16 и 32 битным вариантом.

Nios CPU поставляется с транслятором GNUPro, который включает в себя C/C++ транслятор, макроассемблер, компоновщик, отладчик, утилиты и библиотеки.

Система команд процессора Nios ориентирована на эффективность поддержки программы, написанной на языке C/C++. В процессор Nios версии 2.0 введены команды, определяемые и разрабатываемые пользователем.

На рис. 1 представлен алгоритм работы программы, написанной на языке программирования C для компилятора Nios ядра Altera Sopc Builder. Так как программа предназначена для непрерывной работы, она выполняется в бесконечном цикле. Завершение работы программы или ее сброс, предусмотрен только с помощью отключения питания микросхемы ПЛИС. Выполнение важнейших операций программы осуществляется с периодичностью в 10 мс. Передача информации во внешние узлы происходит по запросу о необходимости передачи данных.

Алгоритм программы разделяется на несколько шагов:

1. Начало – при запуске программы, по умолчанию все переменные инициализируются (например, переменная для считывания данных с порта Input устанавливается в значение 0). Также происходит инициализация всех устройств ввода-вывода.

2. Запрос от ЛМ – ожидание запроса на отправку данных в ЛМ. При его наличии программа переходит к формированию и передаче исходящего пакета данных (Передача данных в ЛМ). В исходящем пакете содержатся данные о состоянии входных дискретных портов. По окончании создания пакета иницируется начало его передачи по протоколу UART (Universal Asynchronous Receiver Transmitter) с применением механизма DMA (Direct Memory Access). Так как использование DMA предусматривает автономный процесс приема и передачи данных, программе необходимо следить только за фла-

гами готовности DMA.

3. Запрос данных от ДМ – аналогичным образом осуществляется ожидание и выполнение запроса на отправку диагностических данных в ДМ.

4. Таймер 10 мс – ожидание флага переполнения таймера, отсчитывающего 10 мс. При его наличии программа переходит к выполнению функции считывания обновленных данных с входных дискретных портов. Пока флаг переполнения таймера отсутствует, в переменных сохраняются старые значения.



Рис. 1. Алгоритм программы модуля ввода дискретных сигналов

4. Моделирование модуля ввода дискретных сигналов на базе ПЛИС в среде ForSyDe

Общая методика моделирования цифровых систем в среде ForSyDe включает следующие шаги [8].

1. Представление системы в виде входных/выходных сигналов и процессов (функций) их обработки.

2. Анализ комбинаций входных/выходных сигналов и формирование процессов их обработки.

3. Построение блок-схемы системы, включающей процессы и сигналы.

4. Разработка программы на языке Haskell, описывающей процессы системы.

5. Компиляция текста программы.

6. Формирование массивов входных данных и выполнение моделирования в среде ForSyDe.

7. Конвертация кода на языке Haskell в код на языке VHDL.

8. Анализ полученного кода на языке VHDL в

среде разработки электронных проектов ПЛИС и анализ корректности компиляции.

9. Исследование цифрового устройства в среде разработки электронных проектов ПЛИС и/или в составе микросхемы ПЛИС с имплементированным электронным проектом.

Рассмотрим данный подход применительно к исследуемому модулю ввода дискретных сигналов.

Представим алгоритм работы системы (см. рис. 1) в виде входных и выходных сигналов, а также процессов их обработки. Как показано на рис. 2, система имеет 4 входных сигнала (Input, 10 ms, Rx request from LCM, Rx request from LCM_d) и 2 выходных (Tx message to LCM, Tx message to LCM_d), обработка которых осуществляется в модуле System.

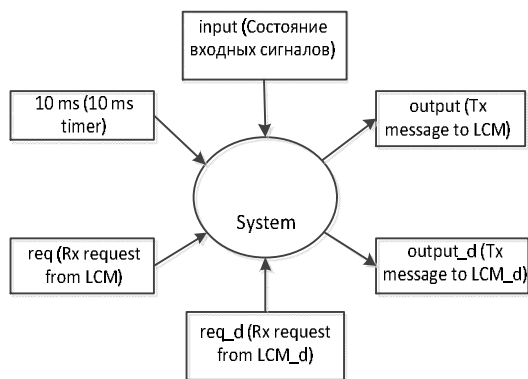


Рис. 2. Схема сигналов системы

Все входные сигналы имеют определенные значения, влияющие на их обработку и на выходные сигналы (см. таблицу 1). Так, например, при переполнении 10 мс таймера (10 ms = True) осуществляется считывание обновлённых данных порта (input = New Signal), и при наличии запроса на передачу данных во внешние модули системы (req = True) происходит передача обновлённых данных (output = New Signal). При отсутствии флага переполнения 10 мс таймера (10 ms = False) сохраняются старые считанные значения, и при наличии запроса на

передачу данных (req = True) происходит передача старых данных (output = Old Signal). При отсутствии запроса на передачу данных во внешние модули (req = False) передача данных не осуществляется (output = '-'), следовательно, переменная данных порта игнорируется.

Таблица 1.

Зависимость выходных сигналов от комбинаций входных сигналов

10 ms	input	req	output	req_d	output_d
True	New Signal	True	New Signal	True	New Signal
True	New Signal	True	New Signal	False	-
True	New Signal	False	-	True	New Signal
True	New Signal	False	-	False	-
False	Old Signal	True	Old Signal	True	Old Signal
False	Old Signal	True	Old Signal	False	-
False	Old Signal	False	-	True	Old Signal
False	Old Signal	False	-	False	-

На рис. 3 изображена схема модели программы в среде ForSyDe, в которой можно выделить следующие сущности [9].

1. Входные сигналы системы – это сигналы, влияющие на обработку данных, поступающие из внешних источников или модулей системы: input – порт считывания входных данных системы, req и req_d – порты запроса на передачу данных во внешние узлы системы.

2. Модуль эмуляции 10 мс таймера – позволяет создать таймер, опирающийся на системную частоту тактовых импульсов. При установке счетчика импульсов (Counter0_9) совместно с компаратором (Comp9) предоставляется возможность получить таймер с заданной задержкой.

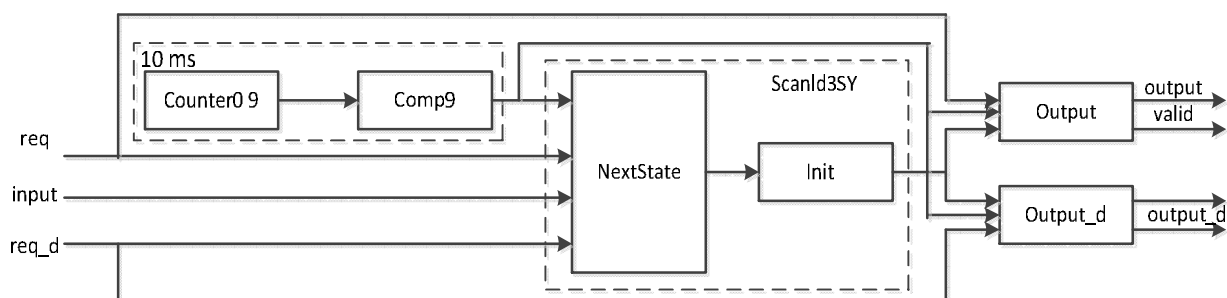


Рис. 3. Блок-схема программной модели системы в среде ForSyDe

3. Модуль обработки данных – при поступлении импульса переполнения 10 мс таймера, осуществляется считывание данных с внешних портов ввода (input) в локальные переменные. При отсутствии сигнала таймера – данные считаны не будут (остаются предыдущие значения). Модуль учитывает сброс переменных (инициализацию) при запуске программы. Данный модуль представляет собой библиотечную функцию (процесс) Scanld3SY, которая управляет запуском функции nextState (декодер входных сигналов в систему), но хранит в себе прошлый результат её выполнения, и подставляет его при последующих запусках этой функции. В то же время, при первом проходе программного цикла выполняется инициализация переменных посредством функции Init (initial = 0).

4. Модули обработки приемо-передатчика Output, Output_d – модули, ожидающие окончания приема байта запроса на передачу во внешние модули системы. При его наличии осуществляется отправка пакета данных во внешние узлы системы.

5. Выходные сигналы системы – это сигналы являющиеся результатом обработки входных данных системой, где установлено 2 приемо-передатчика, в каждом из которых на выходе 2 сигнала (порты данных output, output_d и порты готовности valid, valid_d).

Программа, написанная на языке Haskell, состоит из функций, для вызова которых необходимо передавать входные параметры (переменные или сигналы) и на выходе получить результат её выполнения. Рассмотрим функции, разработанные для реализации программной модели системы, представленной на рис. 3.

Описание функции nextState, реализующей декодер входных сигналов системы, имеет следующий вид:

```
nextState :: Integer -- Current State Value
-> Bool -- 10ms
-> Bool -- Req
-> Integer -- Signal Input Value
-> Integer -- Next State Value
nextState state True _ input = input
nextState state False _ input = state
```

Входные параметры функции nextState:

Integer – предыдущий результат выполнения функции. При отсутствии флага переполнения 10 мс таймера этот параметр передаётся на выход функции (сохраняется содержимое переменной без изменения).

Bool – флаг переполнения 10 мс таймера. При его наличии считываются новые данные с входного порта, иначе сохраняются предыдущие значения без

изменений.

Bool – флаг наличия принятого байта запроса на передачу данных во внешние модули системы.

Integer – вход порта данных.

Integer – результат выполнения функции (выход).

При вызове функции выполняется анализ входного параметра переполнения 10 мс таймера, и при его наличии на выход передается параметр обновленных данных с порта, иначе сохраняются старые значения данных (предыдущий результат выполнения функции). Так как на результат выполнения функции 3-й параметр (флаг наличия принятого запроса на передачу данных во внешние модули системы) не влияет, то в теле функции сигнал игнорируется (Absent), что осуществляется с помощью символа «_».

Функция cpuApp является ядром системы, и осуществляет управление функцией nextState и output. Она имеет следующий вид:

```
cpuApp :: Signal Bool -- 10 ms
-> Signal Bool -- Request
-> Signal Integer -- Input Signals
-> Signal (Integer, Bool) -- Output Package
cpuApp timer req input = output
  where out = scanld3SY nextState initial timer
        req input
        output = zipWithSY enable_output out req
        initial = 0
```

Входные параметры функции cpuApp следующие.

Signal Bool – флаг переполнения 10 мс таймера.

Signal Bool – флаг наличия принятого запроса на передачу данных во внешние модули системы.

Signal Integer – порт ввода считывания цифровых данных.

Signal (Integer, Bool) – результат выполнения функции (данные и признак активности передатчика).

scanld3SY – библиотечная функция, которая управляет запуском nextState, но хранит в себе прошлый результат её выполнения, и подставляет его при последующих запусках этой функции. В то же время, при первом проходе программного цикла выполняет инициализацию переменных (initial = 0).

zipWithSY – библиотечная функция, которая управляет запуском enable_output, и служит для эмуляции передатчика системы.

При вызове функции выполняется обновление данных (out), после чего осуществляется их передача в устройство вывода (output).

Функция enable_output преобразует 2 входящих параметра в один исходящий сигнал, содержащий

эти параметры. В компиляторе ForSyDe предусмотрена возможность выполнения функций, результатом выполнения которых является только один сигнал. Для обхода этих ограничений необходимо в один исходящий сигнал подставлять структуру из нескольких сигналов:

```
enable_output :: Integer
  -> Bool
  -> (Integer, Bool)
enable_output input req = if req == True then
  (input, True)
  else (input, False)
```

Входные параметры функции enable_output:

Integer – данные для передачи.

Bool – флаг активизации передатчика.

(Integer, Bool) – результатом выполнения функции является объединение входящих параметров в один сигнал.

Массивы входных параметров для запуска функций:

```
input = signal [1..20]
req = True :- False :- req
req_d = True :- True :- False :- False :- req_d
```

Массивы входных параметров используют следующие переменные:

input – счётчик входящих данных порта.

req – эмуляция запросов на передачу данных (функциональный канал).

req_d – эмуляция запросов на передачу данных (диагностический канал).

Функция comp9Proc контролирует количество импульсов таймера на входе для осуществления выдачи флага переполнения и имеет следующий вид:

```
comp9Proc :: Signal Integer -- convert integer to
  bool
  -> Signal Bool
comp9Proc input = mapSY comp9 input
```

Входные параметры функции comp9Proc:

Signal Integer – количество импульсов таймера.

Signal Bool – признак переполнения таймера.

mapSY – системная библиотечная функция, которая управляет запуском функции comp9. Функция получает в качестве параметра число тактов таймера, и результатом выполнения является флаг переполнения таймера Signal Bool.

Функция comp9 осуществляет функцию компаратора, анализирующего количество импульсов таймера на входе, и по достижению 9 выдает при-

знак переполнения на выходе:

```
comp9 :: Integer -- comparator set True, if input
  number = 9
  -> Bool
comp9 9 = True
comp9 _ = False
```

Входные параметры функции comp9:

Integer – количество тиков таймера.

Bool – признак переполнения таймера (выход).

Функция выполняет обработку автоматического счетчика путем эмуляции системной частоты тактирования:

```
counterProc0_9 :: Signal Integer -- sourceSY run
  counter0_9 first time with initialization
  counterProc0_9 = sourceSY counter0_9 0 -- then
  set on input value past run this function (a=a+1)
```

Signal Integer – результат выполнения функции.

sourceSY – системная библиотечная функция автоматического инкрементирования числа, с обработкой инициализации при первом запуске.

Функция counter0_9 выполняет автоматическое инкрементирование входящего числа, и при достижении 9 сбрасывает значение в 0.

```
:: Integer -- increment input number, and reset if
  number = 9
```

```
-> Integer
```

```
counter0_9 9 = 0
```

```
counter0_9 a = a + 1
```

Входные параметры функции counter0_9:

Integer – входящий параметр (количество тиков таймера).

Integer – результат выполнения функции.

Всеми модулями системы управляет функция system. Эта функция принимает три внешних сигнала, осуществляет полную обработку этих данных, и передает их для отправки во внешние модули системы. Функция имеет следующий вид:

```
system :: Signal Bool -- req
  -> Signal Bool -- req_d
  -> Signal Integer -- input
  -> (Signal (Integer, Bool), Signal (Integer,
  Bool)) -- Output Package
system req req_d input = output
  where output = (cpuApp comp9_output req in-
  put, cpuApp comp9_output req_d input)
  comp9_output = comp9Proc counterProc0_9
```

Входные параметры функции system:

Signal Bool – сигнал, указывающий о наличии запроса на передачу данных во внешние модули системы (функциональный канал)

Signal Bool – сигнал, указывающий о наличии запроса на передачу данных во внешние модули системы (диагностический канал)

Signal Integer – данные порта ввода.

(Signal (Integer, Bool), Signal (Integer, Bool)) – 2 передатчика данных во внешние модули системы (каждый передатчик имеет данные и флаг активации передатчика).

После завершения написания текста программы необходимо его скомпилировать. Для запуска компилятора необходимо запустить системную консоль операционной системы, перейти в каталог файла написанной программы, и запустить команду «ghci имя_файла». После чего произойдет запуск компилятора, и с помощью команды «:t» осуществляется компиляция файла. Если ошибок не обнаружено, с помощью вызова главной функции программы «system параметры» запускается выполнение написанной программы.

Для моделирования скомпилированной программы необходимо создать массивы входных данных. При запуске функции system, подставляются 3 параметра в виде массивов входных данных: req, req_d и input. После обработки модели программы, функция выводит на экран результат выполнения в виде 2х мерного массива, в котором первым аргументом является десятичное число данных в буфере передатчика, вторым аргументом выступает булевское значение активации передатчика. Так как построенная модель предусматривает работу с двумя приёмо-передатчиками, результатом выполнения программы является передача (вывод на экран) сообщений от двух передатчиков:

```
{(0,True), (0,False), (0,True), (0,False), (0,True),
(0,False), (0,True), (0,False), (0,True), (0,False),
(10,True), (10,False), (10,True), (10,False), (10,True),
(10,False), (10,True), (10,False), (10,True), (10,False),
(20,True)},
{(0,True), (0,True), (0,False), (0,False), (0,True),
(0,True), (0,False), (0,False), (0,True), (0,True),
(10,False), (10,False), (10,True), (10,True), (10,False),
(10,False), (10,True), (10,True), (10,False), (10,False),
(20,True)}
```

С помощью встроенного в систему ForSyDe конвертора можно конвертировать программу, написанную на языке Haskell в код на языке VHDL. Затем этот код может быть проанализирован в среде разработки электронных проектов ПЛИС. Программа на языке VHDL может быть проверена на работоспособность, а затем имплементирована в кристалл ПЛИС.

Для конвертации в VHDL выбрана упрощенная модель системы (рис. 4).

После окончания процедуры конвертации, с помощью утилиты RTL viewer, встроенной в среду разработки Quartus II, был открыт электронный проект программной модели системы, и проанализированы схемы цифрового устройства, представленные на рис. 5.

Общий вид проекта отображен на рис. 5 а. Система имеет следующие входные сигналы:

- clock – частота тактирования системы;
- tenms – импульсы переполнения 10 мс таймера;
- req – запрос на передачу данных;
- reseth – аппаратный порт сброса кристалла;
- input – входной порт данных.

Результатом обработки сигналов функцией mealySY является 2 выходных сигнала:

- output.tup_1 – сигнал активации передатчика;
- output.tup_2 – сигнал данных передатчика.

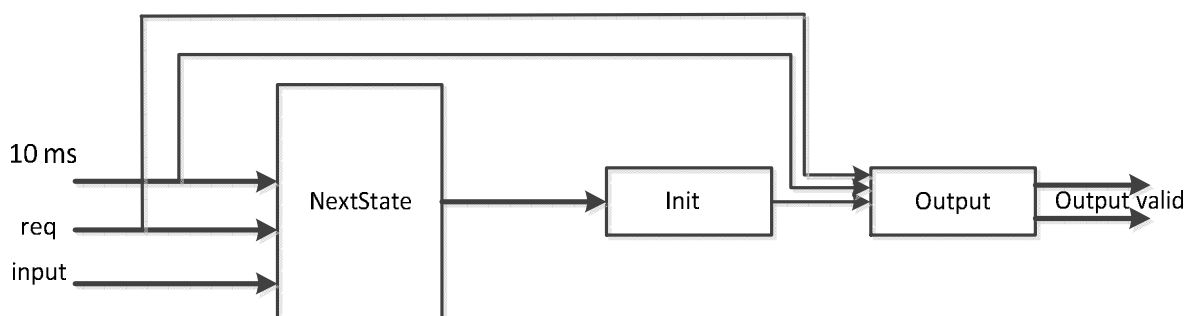


Рис. 4. Упрощенная блок-схема программной модели системы в среде ForSyDe, предназначенная для конвертации в код на языке VHDL

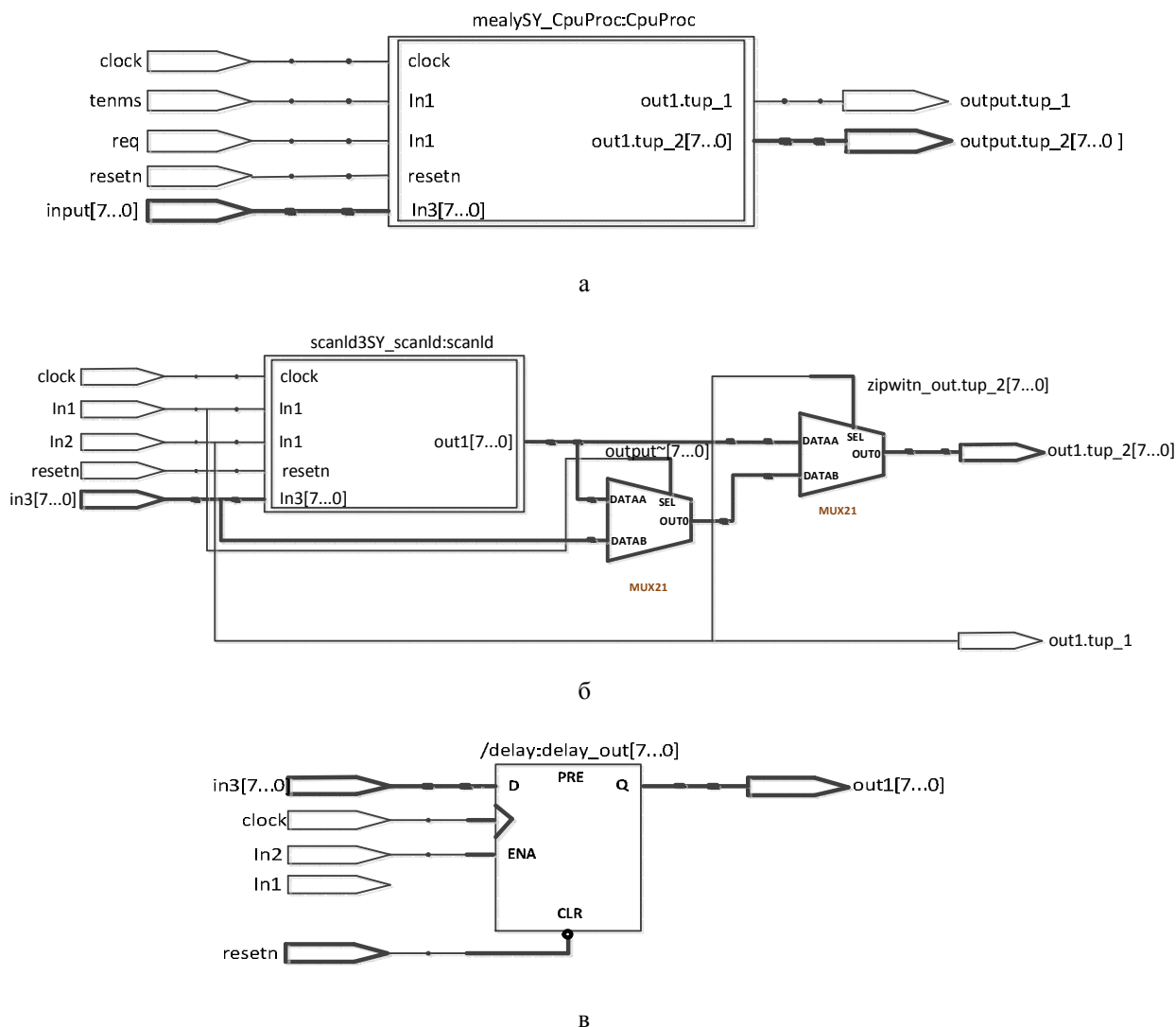


Рис. 5. Представление программной модели системы в среде Altera Quartus II:
 а – общий вид представления системы (функция mealySY); б – представление функции scanld3SY;
 в – представление функции scanld3SY на уровне триггеров

Для более детального рассмотрения схемы, предусмотрена возможность анализа функций на более детальном уровне представления. Так, на рис. 5 б представлена схема реализации функции mealySY. При анализе работы функции можно отметить зависимость обработки данных от импульсов 10 мс таймера, а также зависимость передатчика от импульсов запроса. На рисунке 5 в отображена схема функции scanld3SY.

Функция mealySY, представленная на рисунке 5 а, выполняет операции, отображенные на блок-схеме рис. 4, которая содержит общую структуру проекта со всеми входными и выходными сигналами. Функция Scanld3SY, отображенная на рис. 5 б, выполняет действия блока NextState в блок-схеме рис. 4. Процедура инициализации Init выполняется с помощью триггеров, отображенных на рис. 5 б. Блок output реализован с помощью функции scanld3SY, в

сочетании с триггерами, для активации считывания обновленных данных (рис. 5 б). Внутреннее представление функции scanld3SY дано на рис. 5 в.

Утилиты, входящие в комплект ForSyDe позволяют подключить модуль внешней программы ModelSim, и с её помощью производить моделирование проекта с различными входящими параметрами. Таким образом, предоставляется возможность сравнительного анализа программ запущенных непосредственно компилятором ForSyDe, и эмулятором VHDL-кода.

В результате выполнения моделирования, на экране консоли отображаются выходные данные, полученные с помощью обработки входных сигналов программой ModelSim. Для сравнения правильности обработки алгоритма, был выполнен запуск проекта на языке ForSyDe, с теми же входными данными. По результатам сравнения полученных дан-

ных можно отметить совпадение результатов, что говорит о правильности компиляции VHDL-кода и его обработки программой ModelSim.

Среда разработки Quartus II имеет встроенный симулятор VHDL-кода, с помощью которого возможно рассмотреть работу программы, за определенный промежуток времени, на уровне входных/выходных сигналов. Данная утилита позволяет произвести симуляцию проекта, скомпилированного и прошитого в ПЛИС.

На рис. 6 отображены результаты тестирования системы, обрабатывающей следующие входные и выходные сигналы:

input – входной информационный сигнал с разрядностью 8 бит, представляющий собой счетчик, автоинкрементирующийся каждые 5 мс;

req – входной сигнал наличия запроса на передачу данных во внешние модули системы с разрядностью 1 бит, длительность сигнала случайная, полученная с помощью функции randomize (равномерное распределение случайных чисел);

tenms – входной сигнал импульсов 10 мс таймера с разрядностью 1 бит, периодичность 10 мс, скважность 30%;

output.tup_1 – выходной сигнал активации передатчика с разрядностью 1 бит;

output.tup_2 – выходной сигнал данных для передачи во внешние модули системы с разрядностью 8 бит.

При детальном рассмотрении сигналов можно провести сравнительный анализ вариантов представления исследуемой цифровой системы:

- в виде блок-схемы алгоритма программы (исходные данные для моделирования);

- в виде функциональной программы на языке Haskell в среде ForSyDe;

- в виде электронного проекта ПЛИС в среде разработки Quartus II.

Результаты анализа подтвердили соответствие друг другу всех трех вариантов представления исследуемой цифровой системы.

5. Сравнение результатов функционирования разработанной системы и стандартного IP-Core

Одной из задач выполняемого исследования являлось сравнение полученной модели с программой, разработанной стандартным методом. Как было указано выше, стандартным методом разработки модуля ввода дискретных сигналов на базе ПЛИС является программирование на языке С. Код на языке С выполняется в составе эмулятора микропроцессора Altera Nios и является составляющей электронного проекта ПЛИС в среде разработки Altera Quartus II.

Таким образом, на языке С была разработана программа, соответствующая алгоритму, представленному на рис. 1.

В качестве источника приема сигналов запроса на передачу функциональных данных выступает ЛМ. Задачей ЛМ является формирование запроса и сбор информации от нескольких подключенных блоков с помощью последовательного интерфейса UART.

Источником диагностических запросов выступает ДМ. Задачей ДМ является формирование запроса и сбор информации от нескольких подключенных блоков с помощью последовательного интерфейса UART.

Обновление передаваемой информации осуществляется с помощью таймера, переполнение которого происходит один раз в 10 мс. При его наличии происходит считывание данных с порта OutputData, с последующей записью в переменную Signal. Такая архитектура создана во избежание ошибок при использовании массивов данных, считываемых с внешних портов. В этом случае не исключена ситуация, при которой приемо-передатчик передаст часть старых данных массива вместе с частью обновленных данных в одном пакете.

Все три процесса обрабатываются асинхронно, без какой-либо закономерности прохождения определенной ветки алгоритма.

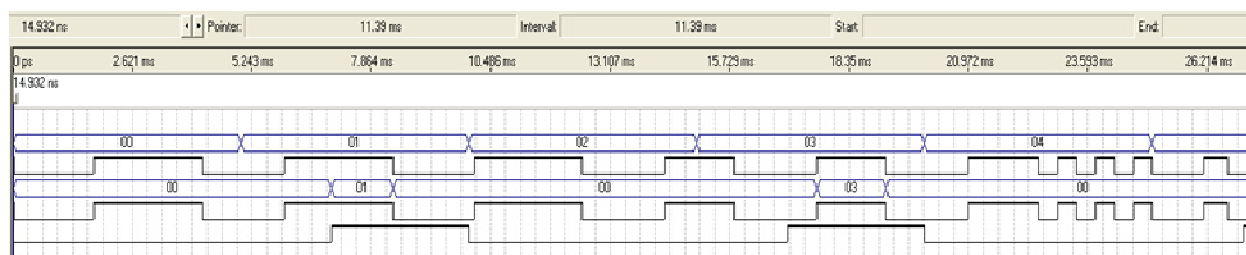


Рис. 6. Симуляция проекта ForSyDe в среде разработки Quartus II

Ниже представлен листинг кода на языке C, выполняющего алгоритм согласно рис. 1.

```
#include <nios.h>

unsigned int ThisPlace;
unsigned int ThisPlaceDiagn;
unsigned int Signal;

int main() {

    na_FuncUart->np_uartcontrol = 0x00;
    na_FuncUart->np_uartstatus = 0x00;

    na_DiagnUart->np_uartcontrol = 0x00;
    na_DiagnUart->np_uartstatus = 0x00;

    while(1)
    {
        if( na_FuncUart->np_uartstatus &
            np_uartstatus_rrdy_mask )
        {
            ThisPlace = na_FuncUart-
                >np_uarttxdata;
            nr_txstring("F:"); nr_txhex(Signal);
            nr_txchar(' ');
            na_FuncUart->np_uarttxdata = Signal;
        }

        if( na_DiagnUart->np_uartstatus &
            np_uartstatus_rrdy_mask)
        {
            ThisPlaceDiagn = na_DiagnUart-
                >np_uarttxdata;
            nr_txstring("D:"); nr_txhex(Signal);
            nr_txchar(' ');
            na_DiagnUart->np_uarttxdata = Sig-
                nal;
        }

        if(na_CycleTimer->np_timerstatus &
            np_timerstatus_to_mask)
        {
            Signal = na_InputData->np_piodata;
            nr_txstring("S:"); nr_txhex(Signal); nr_txchar('
            ');
            na_CycleTimer->np_timerstatus = 0;
        }
    }
}
```

Ядро процессора построено с использованием опорной частоты 48МГц, а также аппаратных UART-интерфейсов и таймера. При инициализации программы, необходимо подключение библиотеки «nios.h», с помощью которой предоставляется возможность управления периферийными устройства-

ми. При инициализации программы необходимо в регистр статуса и регистр управления UART записать управляющие слова, иначе прием не будет активирован. Так как таймер 10 мс является аппаратным, не возникает необходимость в его инициализации.

Тело программы работает в бесконечном цикле. Программа анализирует флаги готовности принятых данных и при их наличии программа осуществляет переход в соответствующую ветку обработки функциональных или диагностических данных. Ответным байтом послыки является содержимое переменной Signal.

Во время работы программы, анализируется флаг переполнения таймера 10 мс, и при его наличии осуществляется считывание обновленных данных из внешнего порта ввода InputData, и сохранение этих данных в переменную Signal. Для наглядности процесса отладки, на вход порта InputData подключен внешний автоинкрементирующийся счетчик размерностью 8 бит. После сохранения данных происходит сброс флага переполнения таймера.

Для более удобного рассмотрения результата выполнения программы, с помощью аппаратных команд, на терминал отладчика (рис. 7) осуществляется вывод информации в виде: «X:YYYY», где:

X = F – принят байт запроса функционального канала;

X = D – принят байт запроса диагностического канала;

X = S – принят флаг переполнения 10мс таймера считывания обновлённых данных внешнего порта;

YYYY – содержимое переменной Signal.

Как видно из рис. 7, передача ответного байта считанных из порта данных порта соответствует алгоритму реализуемой задачи. При наличии запроса на передачу осуществляется передача сохраненных данных порта, считанных только при переполнении соответствующего таймера.

При сравнении результата выполнения программ, написанных на языке C и языке Haskell, можно отметить идентичность результатов выполнения программ. Несмотря на это, стоит отметить определенную сложность построения одинакового алгоритма на принципиально разных языках, так как реализация похожих функций происходит разными методами.

6. Применение многоверсионных технологий для разработки и тестирования компонентов программируемых логических контроллеров

Многоверсионные технологии [10] применяются в наиболее критических системах, важных для

Синтез МВС может быть осуществлен путем комбинирования пар или троек вариантов реализации системы на основе выбора из множества доступных технологий. В качестве примера рассмотрим построение МВС на базе трех следующих технологий:

- традиционная разработка на основе САПР (Т);
- разработка на основе ForSyDe (F);
- разработка на основе VHDL (В).

В таблице 2 рассмотрены варианты комбинаций этих технологий, которые могут быть дополнены другими и сделать таблицу исходным полем для принятия решения.

Таблица 2

Варианты выбора альтернативных версий для верификации и проектирования МВС

Типы систем	Технологии разработки версий			Технологии для верификации
	1	2	3	
1-версионные	Т	-	-	F
				В
				Другие технологии
	F	-	-	Т
				В
				Другие технологии
	В	-	-	Т
				F
				Другие технологии
2-версионные	Т	F		В
				Другие технологии
	F	В		Т
				Другие технологии
	Т	В		F
				Другие технологии
3-версионные	Т	F	В	Другие технологии

7. Выводы и направления дальнейших исследований

В статье рассмотрена возможность реализации цифровых систем, традиционно разрабатываемых на императивных языках в среде процессоров, на функциональных языках программирования.

Для выполнения исследования предложена методика моделирования цифровых систем в среде ForSyDe.

Моделирование выполнялось для модуля ввода дискретных сигналов на базе ПЛИС. Для данного модуля получены и исследованы следующие варианты реализации цифровой системы:

- 1) описание на языке Haskell в среде ForSyDe –

предназначено только для моделирования, однако является исходными данными для получения второго варианта реализации;

- 2) электронный проект на языке VHDL в среде Altera Quartus;

- 3) код на языке С, предназначенный для выполнения в среде эмулятора процессора Nios, являющегося частью электронного проекта ПЛИС.

В результате исследования подтверждена идентичность всех трех вариантов реализации цифровой системы.

Полученные результаты целесообразно использовать для решения следующих практических задач.

Во-первых, реализация функций SoPC альтернативным путем и сравнение результатов тестирования может применяться для независимой верификации таких SoPC в составе систем, важных для безопасности критических объектов. Во-вторых, полученные альтернативные реализации SoPC могут быть использованы для разработки диверсных систем, важных для безопасности критических объектов. Наличие диверсных составляющих системы, выполняющих параллельно общие функции, позволяет усилить защиту системы в глубину и снизить вероятность отказов по общей причине.

Дальнейшие исследования могут быть выполнены в следующих направлениях:

- формальная оценка различия между версиями, полученными с применением разных технологий разработки; такая оценка может быть выполнена, например, методом засева дефектов (fault injection);
- исследование диверсных архитектур электронных проектов, включающих диверсные IP-Cores (диверсные soft-процессоры, диверсные шины передачи данных, диверсные структуры, выполняющие функции логического управления и т.п.).

Литература

1. Field, A. *Functional Programming [Text]* / A. Field, P. Harrison. – Addison-Wesley, 1988. – 616 p.
2. Wadler, P. *Why no one uses functional languages [Text]* / P. Wadler // *ACM SIGPLAN Notices*. – 1998. – Vol. 33(8) August. – P. 23-27.
3. Sander, I. *System Modeling and Design Refinement in ForSyDe. PhD thesis [Text]* / I. Sander. – Royal Institute of Technology, Stockholm, Sweden, 2003. – 228 p.
4. Raudvere, T. *Application and verification of local non-semantic-preserving transformations in system design [Text]* / T. Raudvere, I. Sander, A. Jantsch // *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*. – 2008. – Vol. 27(6) June. – P. 1091-1103.
5. Navas, B. *The RecoBlock SoC platform: A flexi-*

ble array of reusable run-time-reconfigurable IP-blocks [Text] / B. Navas, I. Sander, J. Öberg // *Proceedings of Design Automation and Test in Europe (DATE '13) Grenoble, France. – 2013. – March. – P. 833-838.*

6. Системы управления и защиты ядерных реакторов [Текст] / М. А. Ястребенецкий, Ю. В. Розен, С. В. Виноградская, Г. Джонсон, В. В. Елисеев, А. А. Сиора, В. В. Скляр, Л. И. Спектор, В. С. Харченко ; под. ред. М. А. Ястребенецкого. – К. : Основва-Принт. – 2011. – 768 с.

7. Kharchenko, V. *FPGA-based NPP Instrumentation and Control Systems: Development and Safety Assessment* [Text] / V. Kharchenko, V. Sklyar (Edits). – Kharkiv : National Aerospace University, Ukraine, 2008. – 188 p.

8. Применение концепции Model-Based Testing для верификации систем на базе IP-ядер [Текст] / В. В. Скляр, В. С. Харченко, А. С. Панарин, И. Сандер // *Радиоелектронні і комп'ютерні системи. – 2010. – № 5(46). – С. 237-241.*

9. *Development and Verification of Dependable Multiversion Systems of the Basis of IP-Cores* [Text] / V. Sklyar, A. Siora, A. Herasimenko, A. Panarin // *Technical Approach to Dependability. - Wroclaw, Oficyna Wydawnicza Politechniki Wroclawskiej, 2010. – P. 133-145.*

10. *Defence-in-Depth and Diversity Analysis of FPGA-Based NPP I&C Systems: Conception, Technique and Tool* [Electronic resource] / V. Kharchenko, O. Siora, V. Sklyar, A. Volkoviy // *Proceedings of 20th International Conference on Nuclear Engineering "ICONE 20". - Anaheim, CA, USA. – August 1-3, 2012. - on CD-ROM.*

11. *Towards Designing FPGA-Based Systems by Refinement in B.* [Text] / S. Ostroumov, E. Troubitsyna, L. Laibinis, V. Kharchenko ; edits Luigia Petre, Kaisa Sere, Elena Troubitsyna // *Dependability and Computer Engineering: Concepts for Software-Intensive Systems. – USA : IGI-Global, 2011. – P. 92-112.*

Поступила в редакцію 10.12.2013, рассмотрена на редколлегии 12.02.2014

Рецензент: д-р техн. наук, проф., проф. кафедры автоматизации проектирования ВТ В. И. Хаханов, Харьковский национальный технический университет радиоэлектроники, Харьков.

ТЕСТУВАННЯ І РОЗРОБКА ДИВЕРСНИХ ПРОГРАМОВАНИХ ЛОГІЧНИХ КОНТРОЛЕРІВ НА БАЗІ ПЛІС З ВИКОРИСТАННЯМ СЕРЕДОВИЩА ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

В. В. Скляр, В. С. Харченко, А. С. Панарін

У статті запропоновано підхід до моделювання та тестування програмного забезпечення контролерів з використанням інструментального середовища ForSyDe, що підтримує функціональне програмування. Отримано функціональну модель підсистеми обробки дискретних сигналів, яка потім перетворена в VHDL-код та в цифровий пристрій на базі програмованої логічної інтегральної схеми (ПЛІС). Запропоновано варіанти реалізації життєвого циклу системи на базі ПЛІС з використанням альтернативних версій для тестування і багатoversійного функціонування.

Ключові слова: ПЛІС, ForSyDe, функціональне програмування, IP-ядро, soft-процесор.

TESTING AND DEVELOPMENT OF DIVERSE PROGRAMMABLE LOGIC CONTROLLER ON FPGA USING FUNCTIONAL PROGRAMMING ENVIRONMENT

V. V. Sklyar, V. S. Kharchenko, A. S. Panarin

This paper proposes an approach to modeling and testing software controller using the tool ForSyDe that supports functional programming. A functional model of discrete signal processing subsystem is obtained. This model is then converted into a VHDL-code and a digital device based on Field Programmable Gates Array (FPGA). Variants of the system life cycle implementation on FPGA using alternative versions for testing and functioning of multi-version are proposed.

Key words: FPGA, ForSyDe, functional programming, IP-core, soft-processor.

Скляр Владимир Владимирович – д-р техн. наук, доцент, проф. каф. компьютерных систем и сетей, Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», Харьков, Украина, e-mail: vvslyar@mail.ru.

Харченко Вячеслав Сергеевич – д-р техн. наук, проф., зав. каф. компьютерных систем и сетей, Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», г. Харьков, Украина, e-mail: V.Kharchenko@khai.edu.

Панарин Артём Сергеевич – инженер-программист КБ АСУ ТП, Научно-производственное предприятие «Радий», г. Кировоград, Украина, e-mail: temchick@yandex.ru.