АППАРАТНАЯ РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ПЛИС ДЛЯ ВСТРАИВАЕМЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

В.М. Ефименко

Основными требованиями, предъявляемыми к встраиваемым системам реального времени (СРВ) являются, кроме функциональности, требования по быстродействию и потребляемой энергии. Типовая встраиваемая СРВ состоит из ПЛИС, процессора (CPU), памяти и узлов сопряжения с физическим объектом.

Начальный этап разработки СРВ характеризуется, как правило, быстрой разработкой программного обеспечения на языке Си и начальной аппаратной конфигурации ПЛИС на одном из языков HDL-группы. На этом этапе в ПЛИС программируются только те аппаратные блоки, которые в CPU, в принципе не могут быть реализованы. Например, сдвиговый регистр длинной 1024 бита, работающий на частоте 200Мгц, синтезаторы частоты, блоки сопряжения с АЦП, и др. Не все, что может сделать ПЛИС, может сделать процессор. Трудоемкость разработки программы существенно ниже, чем аппаратного обеспечения, и кроме того, следует учитывать реальное положение дел со специалистами по разработке программного и аппаратного обеспечения. Хотя язык Си не совсем подходит для оптимального исполнения в ПЛИС, но для разработчиков более удобен. Вопрос ускорения вычислений не вопрос аппаратуры, а вопрос программирования.

Далее следует поэтапное улучшение параметров системы по быстродействию и наращивание функциональности за счет аппаратной реализации ряда функций языка Си или всей программы процессора в структуре ПЛИС.

Аппаратно каждая процедура (функция) языка Си реализуется в виде отдельного аппаратного блока, описанного на языке VHDL. Внешне блок представляет собой (рис.1) блок с управляющими входами запуска «Start» и выходом готовности «Ready», а также входом синхронизации «CLK». Кроме того, блок имеет входную шину данных, по которой поступают исходные данные для функции (параметры функции) «D_in» и выходные данные. Разрядность шин «D_in» «D_out» определяется количеством и разрядностью входных параметров типом результата реализуемой функции. Стробирование записи данных во входной регистр производится по сигналу «Start». По этому же сигналу приступает к исполнению алгоритма автомат управления, расположенный в блоке. Для всех процедур разрабатывается их блочная реализация. Блоки находятся всегда в готовности для исполнения. Вызов процедуры – это активизация работы блока, и ожидание готовности при исполнении. Не нужно сохранять в стеке и восстанавливать данные при входе и выходе (возврате) из процедуры. Процедура реализована аппаратно, и территориально занимает определенную площадь на кристалле.

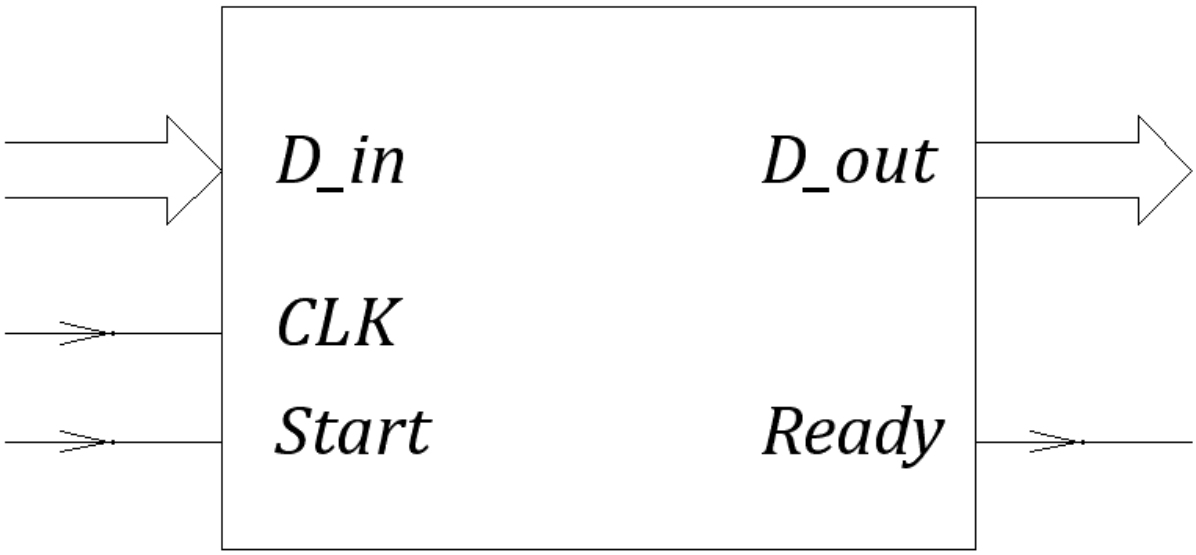

Рис. 1. Представление функции языка Си в виде аппаратного блока

Если данная функция function1( ) имеет вызов другой функции function2( ), то в соответствии с рисунком 2, функция 1 выставляет по выходу «D_out'» параметры для функции 2, опрашивает готовность функции 2, в случае готовности, запускает ее по выходу «Start’» одиночным импульсом, и далее ожидает готовности исполнения от функции 2. По входу «Start» во входные регистры блока function2( ) записываются входные параметры. В это время блок, реализующий вызывающую функцию, простаивает. По приходу

готовности от блока function2( ) выходные данные блока записываются во входные регистры блока function1( ), и блок 1 продолжает функционирование.

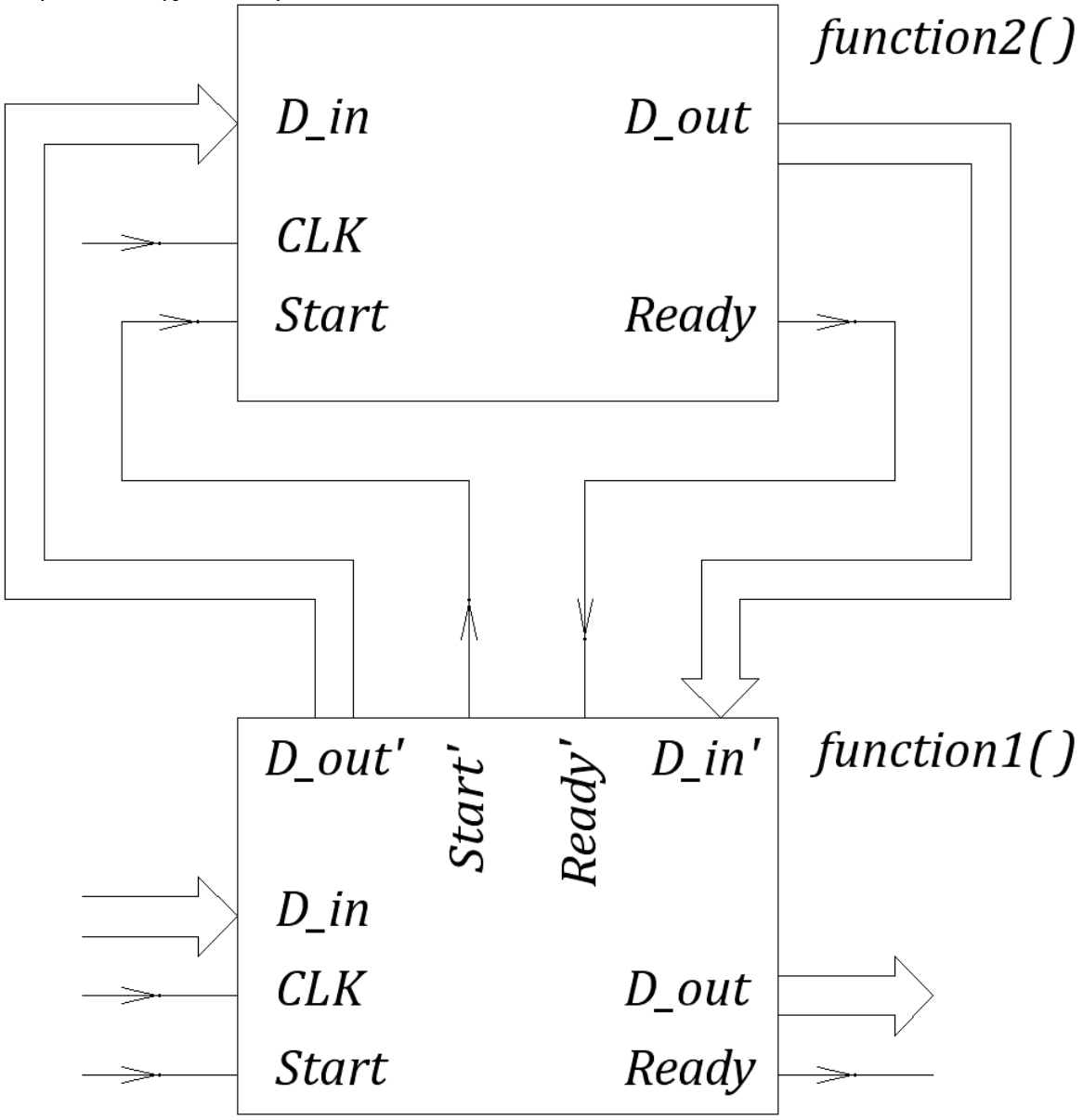

Рис. 2. Вызов функции 2 из функции 1

Управление исполнением программы в блоке осуществляется двоичным автоматом, т.к. любой алгоритм, выраженный на языке процедурного или объектного программирования, может быть реализован с помощью двоичного автомата. Инструкции исходного кода непосредственно переносятся на программный автомат, поддерживающий параллельность, заложенную в исходном тексте программы (коде). Реализация управления в виде жесткого автомата позволяет, кроме быстродействия, получить следующие преимущества. Любой переход, в отличие от программной реализации, не приводит к снижению быстродействия. В одном такте может быть проверено сколько угодно условий, и соответственно, из одного узла программы (состояния автомата) может быть сколько угодно вариантов переходов (имеется в виду оператор «switch( )»). Не нужно «разворачивать» циклы. Проверка условия выхода из цикла производится одновременно как с наращиванием параметра цикла, так и с исполнением подмножества операций внутри цикла. В одном такте могут производиться как вычисления, так и проверки условий. Автомат управления является «жестким» в том смысле, что он не может быть изменен в процессе функционирования, но легко модифицируется при «перепрошивке» ПЛИС.

Для программиста открывается путь повышения быстродействия путем принудительного распараллеливания процедур. Можно одновременно запускать сразу несколько экземпляров одной процедуры, которые реализованы физически в виде четырех одинаковых блоков в ПЛИС, занимают ресурсы ПЛИС и работают действительно параллельно. Возможен одновременный вызов нескольких разных процедур.

Все внутренние переменные данной функции объявляются регистрами. Это позволяет получить одновременный доступ ко всем переменным. Если есть внутренние массивы, то вводится внутренняя, локальная память для каждого массива за счет ресурсов ПЛИС. Массивы, по возможности располагаются не в разных областях одной памяти, а физически в разных ОЗУ. В этом случае каждое ОЗУ имеет меньше «пользователей» и допускается параллельное обращение к массивам. Если есть ссылки на внешний массив, то добавляются все атрибуты доступа к внешней памяти: адрес чтения (записи), старт чтения, данные и готовность результата в виде соответствующих портов ввода-вывода. Если массивы большие, то они организуются во внешней по отношению к ПЛИС памяти.

Для каждого регистра синтезируется отдельное выражение вида

```
if («условие 1») then

      Rg1 <= 0;

else

      if («условие 2») then

            Rg1 <= Rg1 + 1;

      ……..  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

      end if;

end if;
```

где каждое «условие i» представляет собой в общем случае логическую сумму произведений «состояние k»*(«внутреннее условие»), а «состояние k» - состояние «k» управляющего автомата; «внутреннее условие» включает значения внутренних переменных данной процедуры.

Rg1 – регистр, представляющий переменную.

Получается, что к каждой внутренней переменной функции аппаратно поставлен в соответствие регистр вместе с соответствующим операционным элементом для реализации только тех операций, в которых данный регистр выступает в качестве приемника информации. Для ряда выражений, с числом параметров более двух, вводятся дополнительные промежуточные регистры хранения данных. В традиционной реализации CPU+RAM память можно рассматривать как коммутатор, обеспечивающий разновременный последовательный доступ к переменным. Аппаратная реализация, используя коммутационные ресурсы ПЛИС, позволяет получить одновременный доступ ко многим переменным.

Благодаря тому, что операционные блоки совмещены с регистрами и независимы, появляется возможность распараллеливания обработки данных, т.е. одновременное исполнение многих операций. Степень параллелизма определяется только логической связью между результатами вычисления, вытекающими из исходной программы на языке «Си».

Такая структура касается только простых операций, таких как «сложение», «вычитание» а также операций «сдвиг», «сравнение», «целочисленное умножение». Однако существует ряд аппаратных ресурсов, например параллельный умножитель или делитель чисел с плавающей точкой, и др., достаточно «дорогих» с точки зрения аппаратных затрат внутри ПЛИС. Такие ресурсы нецелесообразно ставить в каждый блок реализации процедуры. Это относится к процедурам, которые не обеспечивают полной загрузки данных ресурсов. Это ставит задачу разделения таких ресурсов между блоками (процедурами). К таким «общим» ресурсам относятся «глобальные» с точки зрения программы переменные и массивы памяти, реализованные внутри ПЛИС, а также внешняя память.

К особенности таких аппаратных ресурсов относится их конвейерная реализация, связанная с тем, что результат исполнения появляется не сразу, а спустя несколько тактов, хотя загружать их можно каждый такт. Например, умножитель чисел с плавающей точкой в семействе «Stratix III» имеет тактовую частоту порядка 300МГц, при этом постоянная «pipeline» задержка составляет 10 тактов. Другой пример касается использования внутренней памяти. Команда на запись в память принимается сразу, однако в данные реально запишутся в память только через 2 такта. При чтении из памяти данные появляются на выходе памяти только на третий такт. Однако использовать эти ресурсы можно каждый такт.

Для разделения ресурса была предложена, и широко применяется следующая распределенная структура арбитра (см. рис.Рисунок 3). Пусть блок деления необходимо использовать трем процедурам. Схема содержит три одинаковых блока управления 1-3, мультиплексор 4, и сам блок 5, например блок деления (Block_DIV).

Помимо входов и выходов, идущих к вызывающему блоку, каждый блок управления имеет вход и выход занятости «Busy_in» и «Busy_out». Первый блок управления 1 имеет наивысший приоритет, на его вход занятости подан ноль. Далее сигнал занятости распространяется по цепочке с понижением приоритета. Назначение каждого блока управления – найти такт – временной интервал, в который можно вставить исходные данные для целевого блока «5», с тем, чтобы обеспечить загрузку блока «5» в каждом такте. Данные передаются через мультиплексор «4». Понятно, что блок «1» не всегда занимает своими данными блок «5», иначе он закроет доступ остальных блоков к разделяемому ресурсу «5». Выходной сигнал занятости блокирует остальные блоки, которые в это время ожидают освобождения входных шин блока «5».

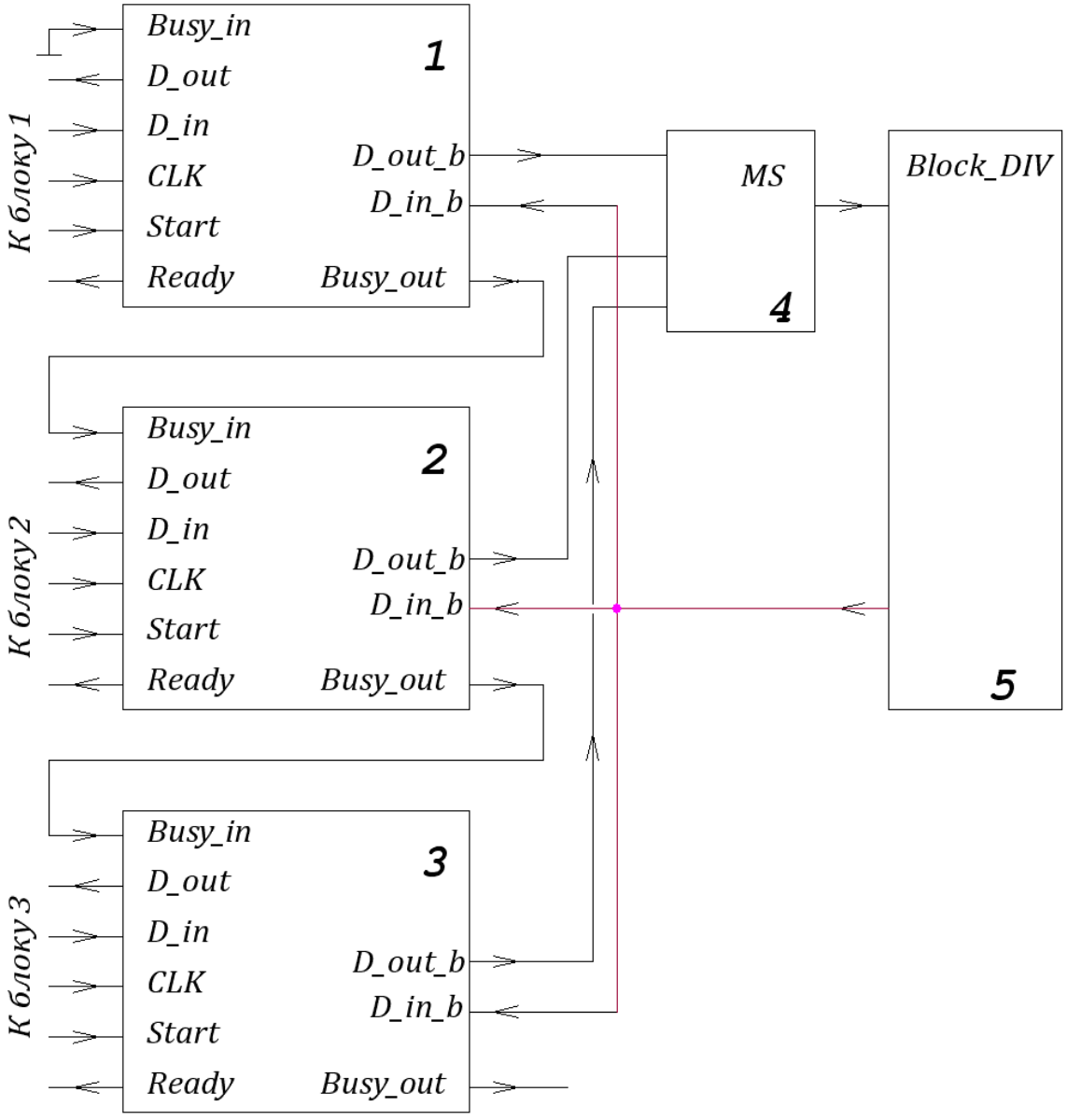

Рис. 3. Структура разделения ресурса

Каждый управляющий блок содержит автомат управления, который, в случае успешной передачи данных, «знает», когда на выходе блока «6» появляются данные, соответствующие именно его запросу. В этот момент производится запись результата из шины «D_in_b» в регистр блока управления и передача данных по шине «D_out» с выдачей в вызывающий блок соответствующего сигнала готовности «Ready».

Каждый блок управления «1-3» содержит свой таймер, который блокирует остальные блоки управления, если данный блок управления не получил сигнал освобождения в течение относительно длительного интервала (128 тактов).

Арбитр разделения ресурсов не имеет централизованного управления. Каждый автомат в блоке управления добивается своей задачи автономно, с учетом очереди, задаваемой соединением «занятость выход» - «занятость вход».

В качестве разделяемого ресурса может выступать любой блок, в том числе и блок, реализующий одну из функций исходной программы для разделения этой функции между потоками.

Рассмотрим важный для встраиваемых приложений вопрос потребления энергии. Для традиционной процессорной реализации программы каждая команда микропроцессора связана с циклами обращения к памяти при чтении и выполнении инструкций, при этом в работе всегда находится достаточно мощный аппарат управления и АЛУ микропроцессора. Во время исполнения происходит переключение во всех разрядах как внешних шин, так и десятков регистров в процессоре.

В случае аппаратной реализации дополнительных обращений к памяти нет, и совершается только та операция, которая нужна, например i=i+1 – однотактное приращение. Если на каждый такт производится одно переключение, то суммарная частота переключений во всем 32-разрядном слове равна f+f/2+f/4+…+f/32~2f. Отсюда вытекает минимальная энергия, необходимая для совершения данной операции.

Для каждой, даже самой простой операции, процессор к «полезному» рассеиванию энергии при выполнении операции прибавляет свою постоянную часть, связанную с выборкой, дешифрацией и исполнением команды,

$$Ecpu = Eком+Eн,$$

где Еком – энергия, затраченная на выполнение команды в идеальном случае,

Ен – дополнительная энергия, рассеиваемая процессором.

Причем, второе слагаемое, как правило, на порядки превышает первое. Можно говорить о «КПД» процессора, а именно сравнивать гипотетическую энергию, затраченную на операцию в идеальных условиях, для аппаратной реализации команды при той же технологии, и «накладные расходы» процессора. С этой позиции «КПД» процессора едва приближается к 10%.

При аппаратной реализации программы вычисления распределены по кристаллу в своих операционных элементах, энергия выделяется распределено по площади кристалла, и только в тех блоках, которые активизированы в данный момент.

В докладе приводится ряд схем дополнительной экономии энергии, связанных с управлением синхронизацией. Данный способ имеет ряд особенностей при реализации в ПЛИС, связанных с корректностью переключения источника синхроимпульсов и ограничениями, накладываемыми допустимыми областями синхронизации.

Приведенные в докладе способы аппаратной реализации представляют собой, по сути, способы ручной компиляции исходного кода. Исходная программа на языке «Си», как документ, рассматривается в качестве исходных данных для разработки аппаратного обеспечения. Хотя это и удобно для организации документирования процесса разработки аппаратного обеспечения в инженерной практике, однако остро ставит задачу разработки транслятора с языка высокого уровня в язык аппаратной реализации, а ПЛИС.

Таким образом, аппаратная реализация программного обеспечения систем реального времени в ПЛИС позволяет:

1. Обеспечить ускорение выполнения программы функционирования в десятки раз по сравнению с микропроцессорной реализацией по сопоставимому технологическому уровню, при сохранении или снижении энергопотребления.
2. Довести начало реакции системы на прерывание до 5нс, обеспечить реакцию на несколько событий одновременно. Собственно, ничего не прерывается, просто вызывается независимый блок (процедура) обработки прерываний.
3. Наращивать функциональность без ущерба для быстродействия, не за счет разделения (временного) ресурса, а за счет добавления параллельно работающих аппаратных средств.
4. Осуществить синтез в одном кристалле ПЛИС как системы сбора и обработки данных, так и классических регуляторов, обеспечивающих управление объектами.