ЭКСПЕРИМЕНТАЛЬНЫЙ КОНВЕРТЕР С ЯЗЫКА С В HDL НА ОСНОВЕ ДИАЛОГОВОГО ВЫСОКОУРОВНЕВОГО ОПТИМИЗИРУЮЩЕГО РАСПАРАЛЛЕЛИВАТЕЛЯ

Д. В. Дубров[*], Р. Б. Штейнберг[†]
Южный федеральный университет,
Россия, 344006, г. Ростов-на-Дону, ул. Большая Садовая, 105
E-mail: dubrov@sfedu.ru, romanofficial@yandex.ru

В работе рассматривается конвертер, позволяющий генерировать описания электронных схем на языке VHDL. Генерируемые схемы реализуют заданные конвейерные вычисления. В качестве входа поддерживается подмножество языка C: одномерные циклы, операторы присваивания и условные операторы. Поддерживается целочисленная арифметика и регулярные индексные выражения.

AN EXPERIMENTAL C TO HDL CONVERTER BASED ON DIALOG HIGH-LEVEL OPTIMIZING PARALLELIZER / D.V. Dubrov[*], R. B. Steinberg[†] (Southern Federal University, 105 B Sadovaya-street, Rostov-on-Don, 344006, Russia). The converter generating electronic circuit descriptions in VHDL language is considered. The generated circuits implement given pipeline computations. A subset of C language is supported as input: one-dimensional loops, assignment and conditional statements. Integer arithmetics and regular index expressions are supported.

1 Введение

Одним из важных требований, предъявляемых к процессу проектирования современных электронных цифровых устройств (на основе ПЛИС или микросхем заказной логики), является малое время разработки. Часто разработчик ради сокращения цикла проектирования устройства готов пожертвовать качественными характеристиками конечного продукта, такими как быстродействие, площадь кристалла, энергопотребление и т. д. Достижению этой цели может способствовать применение средств, позволяющих спроектировать цифровое устройство по описанию алгоритма его работы на языке программирования высокого уровня, такого как C. Существующие в настоящее время немногочисленные программные комплексы, имеющие данную возможность, характеризуются узким классом входных алгоритмов, эффективно реализуемых ими аппаратно, а также высокой стоимостью.

Проект «Диалоговый высокоуровневый оптимизирующий распараллеливатель» (ДВОР, [2]), разрабатываемый в Южном федеральном университете, представляет собой программный

[*]Южный федеральный университет, Российская Федерация, 344006, г. Ростов-на-Дону, ул. Большая Садовая, 105, (Southern federal university, Russian Federation, 344006, Rostov-on-Don, Bolshaya Sadovaya St., 105) dubrov@sfedu.ru

[†]Южный федеральный университет, Российская Федерация, 344006, г. Ростов-на-Дону, ул. Большая Садовая, 105, (Southern federal university, Russian Federation, 344006, Rostov-on-Don, Bolshaya Sadovaya St., 105) romanofficial@yandex.ru

инструментальный комплекс, ориентированный на разработку распараллеливающих и оптимизирующих компиляторов с языков программирования высокого уровня (C, Фортран), систем полуавтоматического распараллеливания. ДВОР включает в себя набор лексических анализаторов, поддержку внутреннего представления программ, а также большую библиотеку оптимизирующих преобразований, использующих граф программных зависимостей и решётчатый граф, и ориентированных на различные целевые параллельные архитектуры.

Основной задачей настоящей работы является разработка на базе ДВОР генератора электронных схем на языке описания оборудования (HDL) для параллельного выполнения конвейеризуемых участков исходной программы на языке C. Данная задача включает в себя следующие этапы:

1. Получение внутреннего представления программы в системе ДВОР.
2. Построение графа вычислений для каждого конвейеризуемого участка программы [4]. Сюда также входит вычисление расписания конвейера.
3. Генерация HDL-описаний участков электронной схемы, реализующих построенные графы вычислений, во внутреннем представлении программы на HDL.
4. Преобразование внутреннего представления результирующей HDL-программы в текст на языке описания оборудования. В настоящее время поддерживается вывод на языке VHDL.

На сегодняшний день на вход конвертера можно подавать одномерные циклы, содержащие операторы **if** и операторы присваивания со вхождениями переменных, содержащими регулярные линейные индексные выражения [1, 5]. Поддерживается целочисленная арифметика. В дальнейшем планируется реализация следующих возможностей:

1. Применение автоматических преобразований программ (раскрутка, развёртка циклов и т. д.), позволяющих эффективнее использовать вычислительные ресурсы, с учётом аппаратных ограничений целевой архитектуры.
2. Автоматическая замена во внутреннем представлении всех операций языка C (в т. ч. вещественной арифметики), не синтезируемых в системах проектирования электронных схем на эквивалентные выражения, реализующие данные вычисления при помощи элементарных операций.
3. Автоматическая организация обмена входными/выходными данными между вычислительным устройством и внешним ОЗУ.

2 Примеры работы конвертера

Пример 1:
Пусть дана следующая программа на языке C:

```
int main()
{
  int i;
  int a[10], b[10], c[10], x[10];
  for (i = 0; i < 10; i = i + 1)
    x[i] = (a[i] + b[i]) - (c[i] * b[i]);
}
```

Таблица 1: Время выполнения операций (такты)

Операция	**Время выполнения**
Чтение данных	1
Запись данных	1
+	3
-	3
*	5

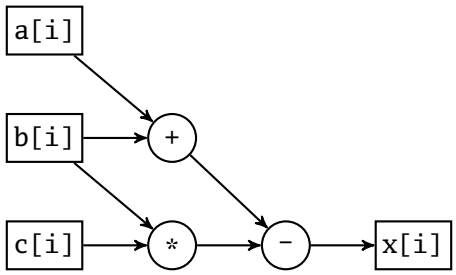

Рис. 1: Граф вычислений программы

Пусть время выполнения каждой операции в тактах задано таблицей 1.

Конвертер рассматривает тело цикла **for** как конвейеризуемый участок. Для него строится граф вычислений, представленный на рис. 1. Вершинами на нём являются операции конвейера: чтения данных (`a[i]`, `b[i]`, `c[i]`), вычислений (`+`, `*`, `-`) и записи (`x[i]`). Дуги графа соответствуют линиям передачи данных между элементарными вычислительными узлами, а также входными/выходными данными.

Для созданного графа вычислений конвертер строит *расписание конвейера*: информацию о том, какая операция должна выполняться на каких тактах. Данная информация включает в себя: *a)* начальные такты выполнения для каждой итерации (таблица 2 в данном примере) и *b) интервал инициализации итераций* (iii) — разницу в тактах между последующими моментами начала одной (любой) операции конвейера (1 в данном примере). Он же определяет частоту подачи входных данных на конвейер (в данном примере — на каждом такте). Таким образом, каждая операция должна выполняться на своём начальном такте, на начальном такте + iii, начальном такте + 2 iii и т. д.

На основе расписания конвейера можно получить временну́ю диаграмму управляющих сигналов электронной схемы, задающих моменты выполнения различных операций. Для данного примера временная диаграмма представлена на рис. 3.

Таблица 2: Начальные такты выполнения операций

Операция	**Начальный такт**
Чтение `a[i]`	0
Чтение `b[i]`	0
Чтение `c[i]`	0
+	1
*	1
-	6
Запись `x[i]`	9

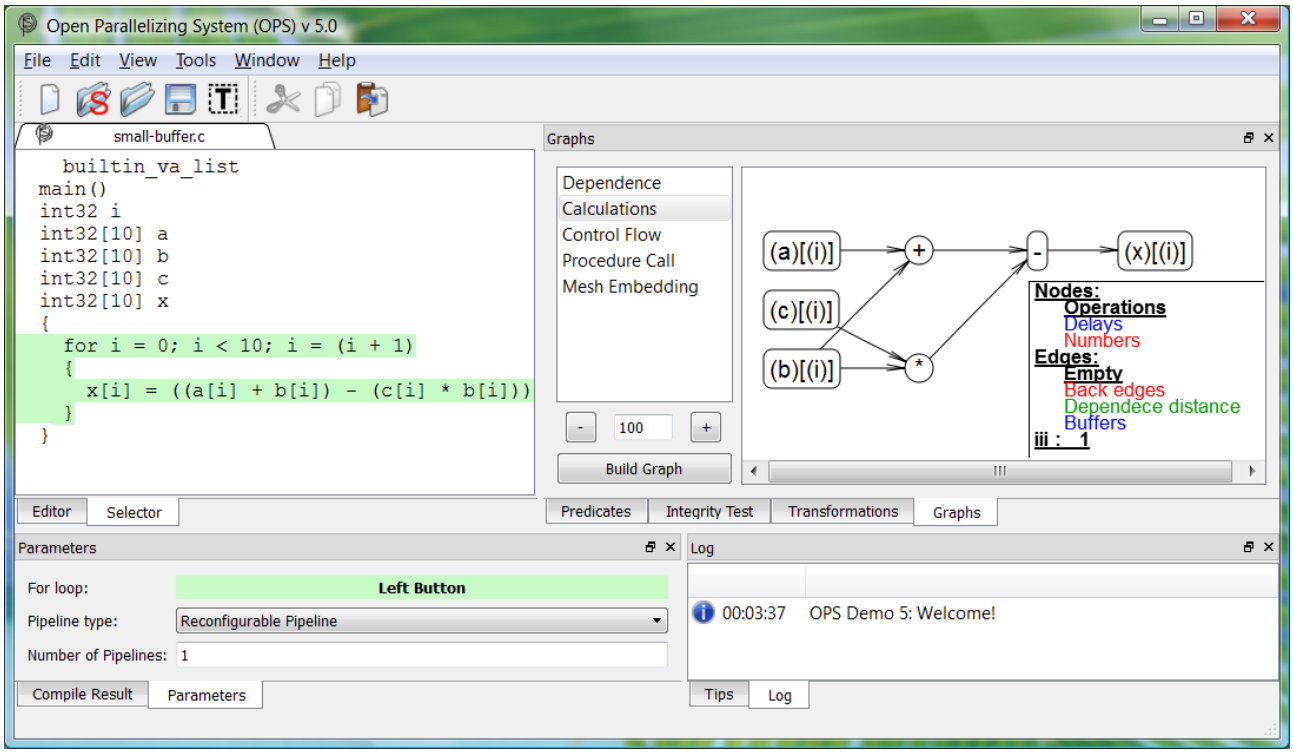

Рис. 2: Окно программы OPS Demo 5 [2] с графом вычислений примера 1

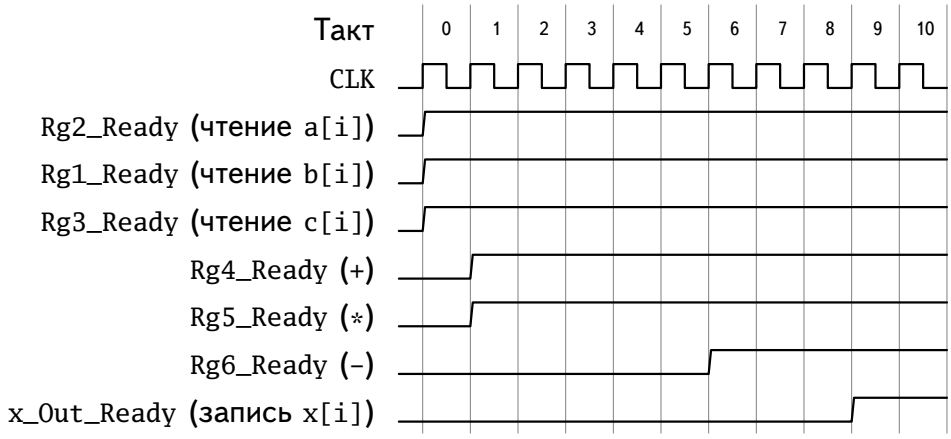

Рис. 3: Временна́я диаграмма сигналов готовности участков конвейера

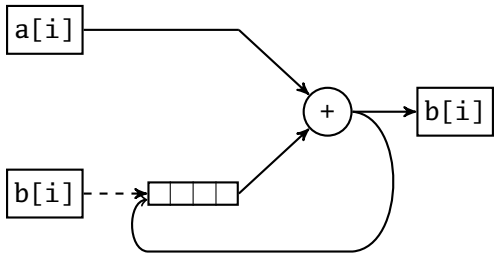

Рис. 4: Граф вычислений программы

Следующим этапом работы конвертера является генерация текста на языке VHDL, который приведён в приложении. При настройках конвертера по умолчанию генерируется описание синхронного устройства. Объект `Sub1` задаёт интерфейс: сигналы входных данных `a`, `b`, `c`, выходных `d`, сигнал готовности выходных данных `Out_Ready`, сигнал запуска конвейера `Start`, сигналы тактового генератора (`CLK`) и сброса (`RST`). В архитектуре объекта (`Sub1_Synth`) описаны операторы процесса, реализующие обработку данных на каждом этапе конвейера, а также управляющий автомат, задающий его расписание. Внутри архитектуры также описаны вспомогательные сигналы для связи вычислительных элементов (`Rg1` – `Rg6`), сигналы готовности (`Rg1_Ready` и т. д.) и счётчики (`x_Out_Ctr` и т. д.), используемые для реализации управляющего автомата. ∗

Пример 2:
Пусть теперь в программе имеется цикл с истинной информационной зависимостью:

```
int main()
{
  int i;
  int a[100], b[100];
  for (i = 4; i < 100; ++ i)
    b[i] = a[i] + b[i - 4];
}
```

Здесь имеется циклически порождённая информационная зависимость с расстоянием в 4 итерации. Граф вычислений для данной программы представлен на рис. 4. Указанная информационная зависимость реализуется на нём при помощи буфера в 4 операнда, организованного по принципу FIFO. Первые 4 такта на вход операции подаются данные `b[0]`...`b[3]` по дуге инициализации, изображённой пунктирной линией. В дальнейшем данные считываются из буфера, куда поступают по обратной дуге графа вычислений (от выхода операции сложения).

Сгенерированный конвертером текст VHDL-описания схемы приведён в приложении. ∗

3 Заключение

Задача синтеза электронных устройств по описаниям на традиционных языках программирования в последнее время привлекает всё больший интерес со стороны производителей. Рассмотренный в настоящей работе конвертер позволяет строить описания конвейерных устройств, выполняющих заданные вычисления на языке C. Реализация конвертера основана на базе Диалогового высокоуровневого автоматического распараллеливателя [2] и использует разработанные авторами алгоритмы расчёта временны́х характеристик вычислительных конвейеров [4, 5]. На текущий момент реализована поддержка ограниченного набора конструкций языка C, в будущем планируется дальнейшее расширение возможностей конвертера.

Список литературы

[1] *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.

[2] Открытая распараллеливающая система. `http://ops.rsu.ru/`.

[3] *Сергиенко А. М.* VHDL для проектирования вычислительных устройств. — К.: ЧП «Корнейчук», ООО «ТИД «ДС», 2003. — 208 с.

[4] *Штейнберг Р. Б.* Вычисление задержки между стартами конвейеров с учётом времени пересылки данных // Труды третьей междунар. конф. «Параллельные вычисления и задачи управления» (PACO‘2006, Москва, 2–4 октября 2006). — С. 542–564.

[5] *Штейнберг Р. Б.* Вычисление задержки в стартах конвейеров для суперкомпьютеров со структурно-процедурной организацией вычислений // *Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, «Наука и Освита»*. — 2003. — № 4. — С. 105–112.

[6] *Ben-Asher Y., Rotem N., Shochat E.* Finding the best compromise in compiling compound loops to Verilog // *Journal of Systems Architecture: the EUROMICRO Journal*. — 2010. — Vol. 56, no. 9. — Pp. 474–486.

[7] Dialogue-based Optimizing Parallelizing Tool and C2HDL Converter / B. Steinberg, A. Abramov, E. Alymova et al. // Proceedings of IEEE East-West Design & Test Symposium (EWDTS’09). Moscow, Russia, September 18–21. — Pp. 216–218.

Приложения

VHDL-описание схемы для примера 1

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

constant CNWidthOfInt: integer = 16;
constant CNLastOfInt: integer = (CNWidthOfInt - 1);

entity Sub1 is
  port(
    CLK: in std_logic;
    RST: in std_logic;
    Start: in std_logic;
    a: in signed(CNLastOfInt downto 0);
    a_In_Ready: in std_logic;
    b: in signed(CNLastOfInt downto 0);
    b_In_Ready: in std_logic;
    c: in signed(CNLastOfInt downto 0);
    c_In_Ready: in std_logic;
    x: out signed(CNLastOfInt downto 0);
    Out_Ready: out std_logic);
end Sub1;

architecture Sub1_synth of Sub1 is
  signal x_Out_Ready: std_logic;
  signal x_Out_Ctr: unsigned(3 downto 0);
  signal Rg1: signed(CNLastOfInt downto 0);
  signal Rg1_Ready: std_logic;
  signal Rg1_Ctr: unsigned(0 downto 0);
  signal Rg2: signed(CNLastOfInt downto 0);
  signal Rg2_Ready: std_logic;
  signal Rg2_Ctr: unsigned(0 downto 0);
  signal Rg3: signed(CNLastOfInt downto 0);
  signal Rg3_Ready: std_logic;
  signal Rg3_Ctr: unsigned(0 downto 0);
  signal Rg4: signed(CNLastOfInt downto 0);
  signal Rg4_Ready: std_logic;
  signal Rg4_Ctr: unsigned(0 downto 0);
  signal Rg5: signed(CNLastOfInt downto 0);
  signal Rg5_Ready: std_logic;
  signal Rg5_Ctr: unsigned(0 downto 0);
  signal Rg6: signed(CNLastOfInt downto 0);
  signal Rg6_Ready: std_logic;
  signal Rg6_Ctr: unsigned(2 downto 0);
begin
  process (CLK, RST, Rg1_Ready) is
  begin
```

```
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg1 <= 0;
  elsif (Rising_edge(CLK) and Rg1_Ready = '1') then
    Rg1 <= b;
  end if;
end process;
process (CLK, RST, Rg2_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg2 <= 0;
  elsif (Rising_edge(CLK) and Rg2_Ready = '1') then
    Rg2 <= a;
  end if;
end process;
process (CLK, RST, Rg3_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg3 <= 0;
  elsif (Rising_edge(CLK) and Rg3_Ready = '1') then
    Rg3 <= c;
  end if;
end process;
process (CLK, RST, Rg4_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg4 <= 0;
  elsif (Rising_edge(CLK) and Rg4_Ready = '1') then
    Rg4 <= (Rg2 + Rg1);
  end if;
end process;
process (CLK, RST, Rg5_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg5 <= 0;
  elsif (Rising_edge(CLK) and Rg5_Ready = '1') then
    Rg5 <= (Rg3 * Rg1);
  end if;
end process;
process (CLK, RST, Rg6_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg6 <= 0;
  elsif (Rising_edge(CLK) and Rg6_Ready = '1') then
    Rg6 <= (Rg4 - Rg5);
  end if;
end process;
process (CLK, RST, x_Out_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    x <= 0;
```

```
  elsif (Rising_edge(CLK) and x_Out_Ready = '1') then
    x <= Rg6;
  end if;
end process;
Out_Ready <= x_Out_Ready;
process (CLK, RST) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg6_Ctr <= 6;
    Rg6_Ready <= '0';
    Rg1_Ctr <= 0;
    Rg1_Ready <= '0';
    Rg4_Ctr <= 1;
    Rg4_Ready <= '0';
    Rg2_Ctr <= 0;
    Rg2_Ready <= '0';
    x_Out_Ctr <= 9;
    x_Out_Ready <= '0';
    Rg5_Ctr <= 1;
    Rg5_Ready <= '0';
    Rg3_Ctr <= 0;
    Rg3_Ready <= '0';
  elsif Rising_edge(CLK) then
    if (Rg6_Ctr = 0) then
      Rg6_Ctr <= 0;
      Rg6_Ready <= '1';
    else
      Rg6_Ctr <= (Rg6_Ctr - 1);
      Rg6_Ready <= '0';
    end if;
    if (Rg1_Ctr = 0) then
      Rg1_Ctr <= 0;
      Rg1_Ready <= '1';
    else
      Rg1_Ctr <= (Rg1_Ctr - 1);
      Rg1_Ready <= '0';
    end if;
    if (Rg4_Ctr = 0) then
      Rg4_Ctr <= 0;
      Rg4_Ready <= '1';
    else
      Rg4_Ctr <= (Rg4_Ctr - 1);
      Rg4_Ready <= '0';
    end if;
    if (Rg2_Ctr = 0) then
      Rg2_Ctr <= 0;
      Rg2_Ready <= '1';
    else
      Rg2_Ctr <= (Rg2_Ctr - 1);
      Rg2_Ready <= '0';
```

```
      end if;
      if (x_Out_Ctr = 0) then
        x_Out_Ctr <= 0;
        x_Out_Ready <= '1';
      else
        x_Out_Ctr <= (x_Out_Ctr - 1);
        x_Out_Ready <= '0';
      end if;
      if (Rg5_Ctr = 0) then
        Rg5_Ctr <= 0;
        Rg5_Ready <= '1';
      else
        Rg5_Ctr <= (Rg5_Ctr - 1);
        Rg5_Ready <= '0';
      end if;
      if (Rg3_Ctr = 0) then
        Rg3_Ctr <= 0;
        Rg3_Ready <= '1';
      else
        Rg3_Ctr <= (Rg3_Ctr - 1);
        Rg3_Ready <= '0';
      end if;
    end if;
  end process;
end Sub1_synth;

-- End of file
```

VHDL-описание схемы для примера 2

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

constant CNWidthOfInt: integer = 16;
constant CNLastOfInt: integer = (CNWidthOfInt - 1);

entity Sub1 is
  port(
    CLK: in std_logic;
    RST: in std_logic;
    Start: in std_logic;
    a: in signed(CNLastOfInt downto 0);
    a_In_Ready: in std_logic;
    b: inout signed(CNLastOfInt downto 0);
    b_In_Ready: in std_logic;
    Out_Ready: out std_logic);
end Sub1;

architecture Sub1_synth of Sub1 is
  signal b_Out_Ready: std_logic;
```

```
  signal b_Out_Ctr: unsigned(0 downto 0);
  signal Rg1: signed(CNLastOfInt downto 0);
  signal Rg1_Ready: std_logic;
  signal Rg1_Ctr: unsigned(0 downto 0);
  signal Rg2: signed(CNLastOfInt downto 0);
  signal Rg2_Ready: std_logic;
  signal Rg2_Ctr: unsigned(0 downto 0);
  signal b_Buf_Ctr: unsigned(2 downto 0);
  signal Rg3: signed(CNLastOfInt downto 0);
  signal Rg3_Ready: std_logic;
  signal Rg3_Ctr: unsigned(0 downto 0);
  signal Rg3_Arg1_Buf1: signed(CNLastOfInt downto 0);
  signal Rg3_Arg1_Buf2: signed(CNLastOfInt downto 0);
  signal Rg3_Arg1_Buf3: signed(CNLastOfInt downto 0);
  signal Rg3_Arg1_Buf4: signed(CNLastOfInt downto 0);
  signal Rg3_Arg1_Buf5: signed(CNLastOfInt downto 0);
begin
  process (CLK, RST, Rg1_Ready) is
  begin
    if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
      Rg1 <= 0;
    elsif (Rising_edge(CLK) and Rg1_Ready = '1') then
      Rg1 <= a;
    end if;
  end process;
  process (CLK, RST, Rg2_Ready) is
  begin
    if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
      Rg2 <= 0;
      b_Buf_Ctr <= 4;
    elsif ((Rising_edge(CLK) and Rg2_Ready = '1') and (b_Buf_Ctr /= 0)) then
      Rg2 <= b;
      b_Buf_Ctr <= (b_Buf_Ctr - 1);
    end if;
  end process;
  process (CLK, RST, Rg3_Ready) is
  begin
    if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
      Rg3 <= 0;
    elsif (Rising_edge(CLK) and Rg3_Ready = '1') then
      if (b_Buf_Ctr = 0) then
        Rg3_Arg1_Buf5 <= Rg3_Arg1_Buf4;
      else
        Rg3_Arg1_Buf5 <= Rg2;
      end if;
      Rg3 <= (Rg1 + Rg3_Arg1_Buf5);
      Rg3_Arg1_Buf4 <= Rg3_Arg1_Buf3;
      Rg3_Arg1_Buf3 <= Rg3_Arg1_Buf2;
      Rg3_Arg1_Buf2 <= Rg3_Arg1_Buf1;
      Rg3_Arg1_Buf1 <= Rg3;
```

```
  end if;
end process;
process (CLK, RST, b_Out_Ready) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    b <= 0;
  elsif (Rising_edge(CLK) and b_Out_Ready = '1') then
    b <= Rg3;
  end if;
end process;
Out_Ready <= b_Out_Ready;
process (CLK, RST) is
begin
  if (Rising_edge(CLK) and (RST = '1' or Start = '1')) then
    Rg3_Ctr <= 1;
    Rg3_Ready <= '0';
    Rg1_Ctr <= 0;
    Rg1_Ready <= '0';
    b_Out_Ctr <= 4;
    b_Out_Ready <= '0';
    Rg2_Ctr <= 1;
    Rg2_Ready <= '0';
  elsif Rising_edge(CLK) then
    if (Rg3_Ctr = 0) then
      Rg3_Ctr <= 0;
      Rg3_Ready <= '1';
    else
      Rg3_Ctr <= (Rg3_Ctr - 1);
      Rg3_Ready <= '0';
    end if;
    if (Rg1_Ctr = 0) then
      Rg1_Ctr <= 0;
      Rg1_Ready <= '1';
    else
      Rg1_Ctr <= (Rg1_Ctr - 1);
      Rg1_Ready <= '0';
    end if;
    if (b_Out_Ctr = 0) then
      b_Out_Ctr <= 0;
      b_Out_Ready <= '1';
    else
      b_Out_Ctr <= (b_Out_Ctr - 1);
      b_Out_Ready <= '0';
    end if;
    if (Rg2_Ctr = 0) then
      Rg2_Ctr <= 0;
      Rg2_Ready <= '1';
    else
      Rg2_Ctr <= (Rg2_Ctr - 1);
      Rg2_Ready <= '0';
```

```
      end if;
    end if;
  end process;
end Sub1_synth;

-- End of file
```