Лабораторная работа 2
По курсу: «Компьютерная схемотехника»

Проектирование счетчика команд

1. **Цель лабораторной работы**:
овладеть знаниями и практическими навыками по проектированию таких комбинационных устройств с памятью, как счетчик команд (ICTR). Лабораторная работа также служит для овладения навыками программирования и отладки описания триггеров и логических схем на языке VHDL.

2. **Теоретические сведения**

Счетчик команд предназначен для вычисления текущего адреса команды в процессоре. В зависимости от вида текущей команды и признака условия, счетчик команд должен выдавать различные адреса новой команды. Если текущая команда – не команда перехода, или если признак условия не истинный, то адрес следующей команды равен текущему адресу плюс длина текущей команды до k байт. Если это команда перехода и условие истинно, то следующий адрес – адрес перехода, который, например, определяется на основе кода поля адреса текущей команды. И при начальной установке состояние ICTR должно быть установлено в ноль.

Таким образом, типичный ICTR должен обладать функциями счетчика с инкрементом 1,2,…,k, с возможностями сброса и начальной установки. Для выполнения функций ICTR необходим малоразрядный сумматор с инкрементом SM, регистр RG младшей части адреса и счетчик - регистр CTR, хранящий старшие разряды адреса и считающий импульсы переполнения SM (рис.2.1).

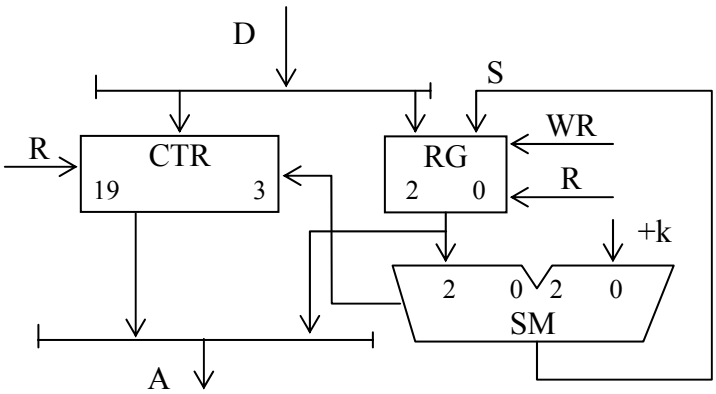


Рис. 2.1. Структура ICTR

Операции ICTR можно закодировать словом F, так как это выполнено в примере, показанном в табл. 2.1. Кодировка F выбирается в каждом конкретном случае особо. Следует отметить, что целесообразно, чтобы часть кодовых комбинаций F совпадала с кодами приращений, как это указано в табл.2.1. В этом случае упрощается схема управления ICTR.

Таблица 2.1

Описание операции	MO	F_2 F_1 F_0	Q^{t+1}	D_T	R
Без изменения	M	0 0 0	Q^t	S	0
Инкремент на 1	+1	0 0 1	S	S	0
То же на 2	+2	0 1 0	S	S	0
То же на 3	+3	0 1 1	S	S	0
То же на 4	+4	1 0 0	S	0	0

Сброс	R	1 0 1	S	X	1
Запись	WR	1 1 0	S	D	1
Безразлично	-	1 1 1	X	X	X

В табл. 2.1 указаны мнемоническое обозначение (МО) операции, значение Q^{t+1} равное состоянию *i*-го разряда сумматора, D_T означает Тип операции ICTR задается управляющим трехбитовым кодом *F*,

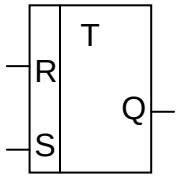
3. **Элементная база**

При выполнении лабораторной работы предлагается реализовать LSM на базе CPLD или ПЛИС. В первом случае в качестве базового элемента используется элемент PLM с триггером (PLMT), а во втором – LUT и D - триггеры. Программирование и включение PLM и LUT с помощью VHDL были подробно описаны в лабораторной работе 1. Рассмотрим программирование D – триггеров.

Триггер представляет собой элемент с памятью. Т.е. уровень его выходного сигнала зависит от состояния триггера, запомненного в предыдущие моменты времени. Такое поведение нельзя описать параллельным оператором присваивания. В большинстве случаев поведение триггера описывается в операторе процесса.

Операторы процесса положены в основу языка VHDL. Такой оператор представляет собой некоторую подпрограмму, в теле которой размещается цепочка последовательных операторов. После запуска процесса при выполнении заданного условия его последовательные операторы исполняются друг за другом и после исполнения своего последнего оператора процесс останавливается. Операторы процесса со списком чувствительности запускаются по каждому событию изменения сигналов, входящих в этот список. Например, в следующем процессе описывается RS-триггер.

```
process(R,S) begin
     if R='1' and S='0'  then
        Q<='1' after td;
     elsif  R='0' and S='1' then
        Q<='0' after td;
     elsif  R='1' and S='1' then
       report "некорректные данные RS -триггера";
     end if;
end process;
```

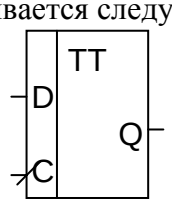


При запуске этого процесса по изменению сигнала R или S если R='1', то триггер устанавливается в 1, а если S='1', то триггер устанавливается в 0. При других запусках процесса триггер не изменяет своего состояния, т.е. его поведение соответствует поведению RS- триггера. Причем если R='1' и S='1', то симулятором должно выдаваться сообщение о некорректной работе триггера.

Приведенный в этом примере триггер относится к числу асинхронных триггеров. Такие триггеры сейчас очень редко используются при проектировании микросхем с их описанием на VHDL, т.к. в схемах с их применением очень трудно достичь ожидаемого уровня работоспособности и повторяемости результатов проектирования. Поэтому в проектах, как правило, применяются синхронные триггеры и регистры на их основе.

Синхронный триггер меняет свое состояние на новое только по фронту или спаду синхросигнала, приходящего на его синхровход. В вычислительной технике наибольшее распространение получили D - триггеры. Такой триггер описывается следующим процессом.

```
 process(C) begin
      if C='1' and C'event then
           Q<=D;
      end if;
 end process;
```

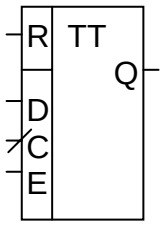


В этом процессе запись в триггер Q происходит в случае события перехода сигнала C из 0 в 1, т.е. по фронту этого сигнала. В логическом выражении условия использован

атрибут сигнала C'event, который равен true, если в момент запуска процесса произошло изменение этого сигнала.

Триггеры, применяемые в ПЛИС и CPLD, имеют также вход разрешения записи E, вход R асинхронной установки в ноль (вход сброса) или вход S асинхронной установки в единицу. Такой триггер с асинхронным сбросом описывается в следующем процессе.

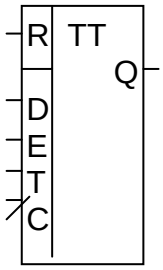
```
process(C,R) begin
     if R='1' then
          Q<='0';
      elsif C='1' and C'event then
         if E='1' then
            Q<=D;
          end if;
     end if;
end process;
```



Следует отметить, что аналогичным образом описывают и регистры. Так, если в приведенных примерах сигнал Q – это вектор из n бит, то такой процесс описывает n-разрядный регистр.

Для построения счетчиков иногда применяются T-триггеры, которые можно описать следующим процессом.

```
process(C,R) begin
     if R='1' then
          Q<='0';
      elsif C='1' and C'event then
         if E='1' then
            Q<= D;
          elsif T='1' then
            Q<= not Q;
         end if;
     end if;
end process;
```



В этом триггере по сигналу асинхронного сброса R триггер устанавливается в ноль, по сигналу E выполняется синхронная начальная установка триггера с входа D, а по сигналу T - инвертируется его состояние.

Часто триггер вводится в проект с помощью оператора вставки компонента. Следующий оператор вставляет компонент FDRE - триггера с асинхронным сбросом и разрешением записи.

```
FF: FDRE port map (Q=>Q, D=>D, C=>CLK, CE=>E, R=>R);
```

В этом операторе использовано поименованное связывание имен портов триггера с входными и выходным сигналами. Оригинальная модель этого компонента описана в библиотеке UNISIM, применяемой при проектировании ПЛИС фирмы Xilinx. Но компонент такой модели не может быть непосредственно вставлен и связан с битовыми сигналами. Поэтому в данной лабораторной работе предлагается использовать такой же компонент, описанный в библиотеке CNetlist_Lib, описанный с использованием битовых сигналов.

При использовании моделей триггеров из библиотеки CNetlist_Lib, проект транслируют вместе с файлом в CNetlist_Lib.VHD, а в декларативной части архитектуры необходимо объявить этот компонент как:

```
component FDRE  is port (Q:out  bit;
      D :   in   bit;
      C :   in   bit;
      CE:   in   bit;
      R :   in   bit);
end component;
```

Использование вставки компонента триггера в проекте имеет то преимущество, что компилятор – синтезатор обязан вставить в результирующую логическую схему компонент триггера именно заданного типа, а не произвольного триггера. Во многих случаях это уменьшает вероятность ошибок проектирования.

В качестве логических элементов можно применять такие же элементы, которые применяются в предыдущей лабораторной работе.

4. **Примеры описания ICTR**

Рассмотрим пример проектирования ICTR для параметра k=4 и числа входов логических схем N=4, структура которого показана на рис.2.1, который функционирует согласно таблице 2.1. Объявление объекта для такого ICTR выглядит следующим образом.

```
use  Cnetlist.all;
entity ICTR is
 port(CLK : in BIT;  -- синхровход
      R : in BIT;  -- сброс
      WR: in BIT;  -- сигнал записи
      D : in BIT_VECTOR(19 downto 0);  -- адрес перехода
       F : in BIT_VECTOR(2 downto 0);   -- функция
       A : out BIT_VECTOR(19 downto 0)  -- выходной адрес
       );
end ICTR;
```

4.1**. Поведенческая модель ICTR.**

Поведенческая модель объекта представляет собой алгоритм его функционирования, который может быть затем автоматически преобразован в соответствующую логическую схему с регистрами. Поведение ICTR зависит не только от входных данных, но и от состояния ICTR, изменяемого в предыдущие моменты времени. Поэтому устройство ICTR невозможно описать только операторами параллельного присваивания и необходимо использовать операторы процесса.

Поведенческую модель ICTR можно описать в виде следующего описания архитектуры.

```
architecture BEH of ICTR is
 signal RG: BIT_VECTOR(2 downto 0);-- регистр мл.части адреса
 signal SM: BIT_VECTOR(3 downto 0);--  сумматор мл.части адреса
 signal CTR:BIT_VECTOR(19 downto 3);-- счетчик ст.части адреса
 signal CTRi:integer;                  -- он же
begin
-- описание сумматора ------------------------------------
  SUM:SM<= INT_TO_BIT( (BIT_TO_INT(RG)+ BIT_TO_INT(F)),4);
-- описание RG -------------------------------------------
  R_3:process(R,C)
  begin
       if R='1' then
            RG<="000";
       elsif CLK='1' and CLK'event then
            case F is
                 when "001"|"010"|"011"|"100"=>RG<=SM(2 downto 0);
                 when "101" => RG<="000";
                 when "110" => RG<=D(2 downto 0);
                 when others=> null;
            end case;
       end if;
  end process;
-- описание счетчика -------------------------------------
  CT:process(CLK,R)
  begin
       if R='1' then
            CTRi<=0;
       elsif CLK='1' and CLK'event then
            if F="101" then
                 CTRi<=0;
            elsif F="110" then
                 CTRi<= BIT_TO_INT(D(19 downto 3));
            elsif (F(2) = '0' or F="100") and SM(3) ='1' then
```

```
                CTRi<= CTRi+1;
            end if;
        end if;
    end process;
    CTR<= INT_TO_BIT(CTRi,17);
    A<=CTR&RG;  -- выходной адрес
end BEH;
```

В операторе, отмеченном меткой SUM, описан сумматор SM (см. рис.2.1) . При этом векторы бит преобразуются в целые числа, а затем целый результат преобразуется в битовый вектор. 3 младшие разряда суммы SM(2 downto 0) подаются на вход регистра RG, а старший разряд SM(3)– как разряд переполнения - на счетный вход счетчика CTR.

В процессе R_3 описан трехразрядный регистр. В него по фронту синхросерии записывается ноль, 3 младших разряда входного данного D или сумма SM в зависимости от управляющего слова F, семантика которого указана в таблице 2.1.

Логика управления регистром реализована оператором **case**. Этот оператор разрешает выполнение одной из цепочек последовательных операторов в зависимости от значения выражения селектора F. Главное правило составления оператора **case** - никакие два значения в выражениях альтернатив **when** не должны быть равны друг другу, т.е. множества альтернатив не перекрываются. Последней альтернативой может быть ключевое слово **others**, которое указывает на неперечисленные альтернативы. Если слово **others** не применяется, то в альтернативах должны быть перечислены все возможные значения, принимаемые в селекторе F. Пустой оператор – оператор **null** – не выполняет никаких действий и может быть вставлен в любом месте программы как последовательный оператор.

В операторе процесса CT описано поведение 17- разрядного счетчика CTR. При этом состояние счетчика представлено целым числом CTRi. В зависимости от управляющего слова F, к состоянию счетчика прибавляется разряд переноса SM(3), состояние обнуляется или принимается новое значение с шины D.

Результирующий адрес A формируется с помощью операции конкатенации из разрядов регистра RG и счетчика CTR.

Следует отметить, что приведенный пример описания архитектуры, описан синтезируемым стилем. Но так как в нем состояние счетчика CTRi задано целым числом без ограничений его диапазона, то в результирующей схеме его разрядность будет равна 32. если объявить этот сигнал как:

```
signal CTRi:integer range(0 to 2**17-1);
```

то этот счетчик будет 17-разрядным.

4.2. Структурная модель ICTR на базе PLMT.

Рассмотрим проектирование ICTR на базе PLMT. У всех PLM одинаковое объявление объекта, такое, как приведено в лабораторной работе 1. Отличие лишь в количестве входов: PLM_4 – 4-входовая, а PLM_3 – 3-входовая PLM. Отдельные триггеры имеют интерфейс триггера FDRE, описанного выше.

Модель имеет структуру верхнего уровня, показанную на рис.2.1. В ней 3 узла будем проектировать раздельно как отдельные объекты. При этом учитываем, что поведение этих узлов соответствует трем операторам описанной выше архитектуры ICTR(BEH)

Проектирование SM

SM представляет собой 3-разрядный сумматор числа R с числом F в диапазоне от 0 до 4 с выходом переноса Q(3). Таблица 2.2 представляет собой таблицу истинности одного разряда сумматора.

Таблица 2.2

Ri Fi Qi	Q_{i+1} Si	Ri Fi Qi	Q_{i+1} Si
0 0 0	0 0	1 0 0	0 1
0 0 1	0 1	1 0 1	1 0
0 1 0	0 1	1 1 0	1 0
0 1 1	1 0	1 1 1	1 1

PLM, которые вычисляют i-й разряд переноса – PLM_Q и разряд суммы – PLM_S, приведены ниже при C = Ri, B=Fi, A=Qi.

```
architecture PLM_Q of PLM_3 is
    begin
     Y<=( (not C and B and A)
        or (C and not B and A) or ( C and B )
        )              after td;  --задержка элемента
    end PLM_Q;

architecture PLM_S of PLM_3 is
    begin
     Y<=((not C and not B and A) or (not C and B and not A)
        or ( C and not B and not A) or (C and B and A)
        )              after td;  --задержка элемента
    end PLM_S;
```

Можно принять во внимание, что $Q_0 = 0$, $Q_1 = R_0F_0$. Тогда первые разряды переноса и суммы можно вычислить в следующих PLM при $D = R_1$, $C = F_1$, $B = R_0$, $A = F_0$.

```
architecture PLM_Q1 of PLM_4 is
    begin
    Y<=((not D and C and B and A)
        or (D and not C and B and A) or ( D and C )
        )              after td;  --задержка элемента
    end PLM_Q1;
architecture PLM_S1 of PLM_4 is
    begin
Y<=(( not D and not C and B and A) or (D and not C and not B)
        or (D and not C and not A) or (not D and C and not B)
        or (not D and C and not A) or (D and C and B and A)
        )              after td;  --задержка элемента
    end PLM_S1;
```

На основе этих PLM строим следующую структурную модель сумматора.

```
    entity SM_3 is port(
                R:in bit_vector(2 downto 0);-- данное с регистра
                F:in bit_vector(2 downto 0);-- инкремент
                S:out bit_vector(2 downto 0);-- сумма
                Q:out bit);
    end entity;
architecture PLM of SM_3 is
   constant gnd:bit:='0';  -- нулевой бит
   signal qi2: bit;  -- перенос в 2 разряд
begin
      S0:entity PLM_3(PLM_S) port map  -- 0 разряд суммы
              (a=>gnd,b=>F(0),c=> R(0),    Y =>S(0));
      S1:entity PLM_4(PLM_S1) port map -- 1 разряд суммы
              (a=>F(0),b=>R(0),c=> F(1),d=>R(1),Y =>S(1));
      S2:entity PLM_3(PLM_S) port map  -- 2 разряд суммы
              (a=>qi2,b=>F(2),c=> R(2),    Y =>S(2));
      Q2:entity PLM_4(PLM_Q1) port map  -- перенос в 2 разр.
              (a=>F(0),b=>R(0),c=> F(1),d=>R(1),Y =>qi2);
      Q3:entity PLM_3(PLM_Q) port map  -- выходной перенос
              (a=>qi2,b=>F(2),c=> R(2),Y =>Q);
end PLM;
```

Таким образом, вместо шести PLM в сумматоре применяется только пять.

Проектирование RG.

Узел RG представляет собой регистр, на входе которого M подключен мультиплексор, пропускающий данное с выхода сумматора SM, входное данное D или выдающий ноль в зависимости от кода F. Если обусловить, что запись в регистр выполняется в каждом такте, то формирование сигнала разрешения записи в триггер не нужно. Тогда выход S сумматора подается на вход регистра при условиях: F=000|001|010|011|100 , а вход D – при условии F=110, а в остальных случаях – ноль. Все разряды мультиплексора одинаковые, поэтому он реализуется на PLM одного типа при $E=S_i$, $D = D_i$, $C = F_2$, $B=F_1$, $A=F_0$:

```
architecture PLM_MX of PLM_5 is
    begin
Y<=((E and not C) or (E and C and not B and not A)-- пропуск S
        or (D and C and B and not A) -- пропуск D
        )             after td;  --задержка элемента
    end PLM_MX;
```

Описание объекта регистра выглядит следующим образом.

```
    entity RG_3 is port(CLK:in bit;   --синхросигнал
                R:in bit;-- асинхронный сброс
                F:in bit_vector(2 downto 0); -- управление
                D:in bit_vector(2 downto 0); -- входное данное
                S:in bit_vector(2 downto 0); -- сумма
                RG:out bit_vector(2 downto 0)-- выход регистра
                );
    end entity;
architecture PLM of RG_3 is
    component FDRE  is port (Q:out  bit;
          D, C, CE, R:in bit);
    end component;
    constant one:bit:='1';
    signal M:bit_vector(2 downto 0);-- выход мультиплексора
begin
MX_RG:for i in 0 to 2 generate
    MUX: entity PLM_5(PLM_MX) port map  -- мультиплексор
       (E=>S(i), D=>D(i),C=>F(2),B=>F(1),A=>F(0),Y=>M(i));
    REG: FDRE  port map     --регистр
       (C=>CLK,R=>R,CE=>one,D=>M(i),Q=>RG);
    end generate;
end PLM;
```

Проектирование CTR. Счетчик адреса можно представить в виде регистра и сумматора, прибавляющего к содержимому Q регистра сигнал переноса C_3 с сумматора SM. У этого сумматора должны быть функции генерации нуля при F=101 для обнуления счетчика и пропуска входного данного D при F=110, необходимого при переходе по новому адресу. Каждая схема формирования разряда сумматора D_T должна состоять из PLM формирования сигнала переноса C_i в следующий разряд и PLM сигнала возбуждения триггера .

PLM переноса описывается в следующей архитектуре при $A=C_{i-1}$, $B=Q_i$.

```
architecture PLM_C of PLM_3 is
    begin
     Y<=B and A after td;
    end PLM_C;
```

PLM возбуждения триггера описывается в следующей архитектуре при $F=Q_i$, $E=C_i$, $D = D_i$, $C = F_2$, $B=F_1$, $A=F_0$:

```
architecture PLM_CT of PLM_6 is
    begin
     Y<=(((F and not E) or (not F and E)) and       --XOR
                  (not C or (C and not B and not A)))--сумма
     or (D and C and B and not A) -- пропуск D
```

```
                after td;  --задержка элемента
    end PLM_CT;
```

Тогда объект счетчика описывается следующим образом.

```
    entity CTRG is port(CLK:in bit;   --синхросигнал
                R:in bit;-- асинхронный сброс
                C3:in bit;-- перенос из сумматора
                F:in bit_vector(2 downto 0);-- управление
                D:in bit_vector(19 downto 3);-- входное данное
                CTR:out bit_vector(19 downto 3)-- выход счетчика
                );
    end entity;
architecture PLM of CTRG is
    component FDRE  is port (Q:out  bit;
          D, C, CE, R:in bit);
    end component;
    constant one:bit:='1';
    signal c:bit_vector(20 downto 3); -- переносы
    signal Q:bit_vector(19 downto 3); -- выход регистра
    signal Dt:bit_vector(19 downto 3);-- вход регистра
begin
    c(3)<=C3;
  CT:for i in 3 to 19 generate
       CARRY: entity PLM_3(PLM_C) port map  -- переносы
          (C=>one,B=>C(i),A=>Q(i),Y=>c(i+1));
       SM: entity PLM_6(PLM_CT) port map  -- сумматор
    (F=>Q(i),E=>C(i), D=>D(i),C=>F(2),B=>F(1),A=>F(0),Y=>Dt(i));
       REG: FDRE  port map     --регистр
          (C=>CLK,R=>R,CE=>one,D=>Dt(i),Q=>Q(i));
    end generate;
    CTR<=Q;         --выходной сигнал
end PLM;
```

Теперь, когда архитектуры всех составляющих блоков подготовлены, можно составить описание архитектуры ICTR в целом. Оно выглядит следующим образом.

```
    architecture PLM of ICTR is
     component SM_3 is port(R:in bit_vector(2 downto 0);-- данное
                 F:in bit_vector(2 downto 0);-- инкремент
                S:out bit_vector(2 downto 0);-- сумма
                Q:out bit);
     end  component;
     component  RG_3 is port(CLK:in bit;   --синхросигнал
                R:in bit;-- асинхронный сброс
                F:in bit_vector(2 downto 0);-- управление
                D:in bit_vector(2 downto 0);-- входное данное
                S:in bit_vector(2 downto 0); -- сумма
                RG:out bit_vector(2 downto 0) );-- выход регистра
     end component ;
     component  CTRG is port(CLK:in bit;   --синхросигнал
                R:in bit;-- асинхронный сброс
                C3:in bit;-- перенос из сумматора
                F:in bit_vector(2 downto 0);-- управление
                D:in bit_vector(19 downto 3);-- входное данное
                CTR:out bit_vector(19 downto 3) );-- выход счетчика
     end component;
     signal RG: BIT_VECTOR(2 downto 0); -- регистр мл.части адреса
     signal SM: BIT_VECTOR(2 downto 0); -- сумматор мл.части адреса
     signal CTR:BIT_VECTOR(19 downto 3);-- счетчик ст.части адреса
     Signal C3:BIT;                      --перенос
 begin
    -- Сумматор ----------------------------
     U_SM:SM_3 port map(R=>RG,F=>F,-- инкремент
          S=>SM,                  -- сумма
          Q=>C3);                  -- перенос
    -- Регистр  ----------------------------
```

```
 U_RG:RG_3 port map(CLK=>CLK,R=>R,F=>F,-- управление
       D=>D(2 downto 0),       -- входное данное
       S=>SM,                  -- сумма
       RG=>RG);                -- выход регистра
-- Счетчик старших разрядов адреса ---------------
 U_CT:CTRG port map(CLK=>CLK,R=>R,C3=>C3,-- перенос
       F =>F,                   -- управление
       D  =>D(19 downto 3),     -- входное данное
       CTR=>  A(19 downto 3)); -- выход счетчика
 A(2 downto 0)<=RG;
end PLM;
```

Эта архитектура была перекомпилирована в графическое представление струкутры, которое показано на рис.2.2. Сравнение этой структуры со структурой на рис.2.1. указывает на их идентичность.

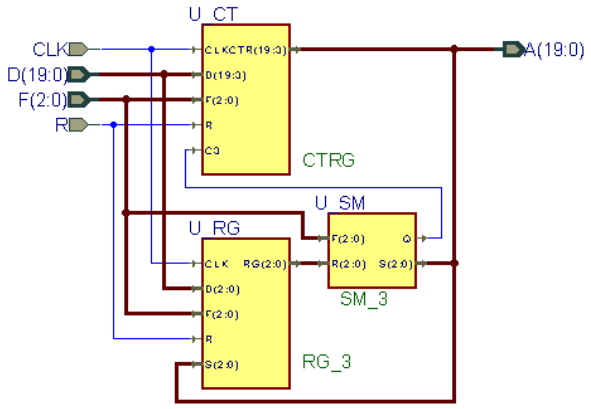


Рис.2.2. Структура ICTR, полученная из архитектуры ICTR(PLM)

4.4. Испытательный стенд для ICTR

Рассмотрим испытательный стенд - объект `ICTR_TB` - для архитектуры `ICTR(LUT)`, у которой эталонной моделью является архитектура `ICTR(BEH)`. Испытания заключаются в подаче кода операции F с генератора случайных чисел на входы обеих моделей и сравнении выходных адресных последовательностей. Для такого сравнения используем следующий объект компаратора.

```
entity COMPARATOR is
 generic(n:positive:=20;  -- разрядность векторов
         del:time:=20 ns);
 port(CLK: in bit;           --синхросерия
      D1 : in BIT_VECTOR(n-1 downto 0); --1-й вектор
       D2 : in BIT_VECTOR(n-1 downto 0); --2-й вектор
       Q : out BOOLEAN);  -- результат сравнения векторов
end COMPARATOR;
architecture BEH of COMPARATOR is
begin
 process    begin
       wait until CLK='1';
       wait for del;
       Q<=true;
       if   D1/=D2  then
            Q<= false;
       end if;
       assert D1=D2 report "ошибка сравнения";
 end process;
end BEH;
```

В нем с помощью настроечной переменной `n` задается разрядность сравниваемых векторов. Благодаря этому, он может быть использован в других проектах для сравнения векторов произвольной длины. Компаратор выдает булевский сигнал `false`, если

сравниваемые векторы неодинаковые. При этом сравнение выполняется через `del` наносекунд после фронта синхносигнала , т.е. после того как прошли переходные процессы в тестируемых схемах. При несовпадении векторов на консоль симулятора также выдается сообщение об ошибке. Это сообщение выдается командой **`assert`**, которая является "ловушкой" ошибочных ситуаций при моделировании.

В графическом виде испытательный стенд показан на рис. 2.3.

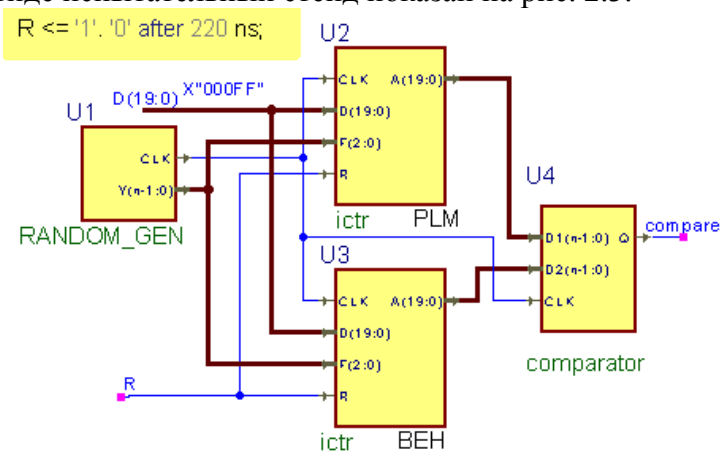


Рис. 2.3. Испытательный стенд для ICTR

5. **Порядок проведения лабораторной работы**

В соответствии с номером варианта, выбирается задание на выполнение лабораторной работы. Параметры задания включают:

- тип логического элемента (PLM или LUT);
- максимальное число термов PLM или количества входов LUT (4 или 5);
- разрядность результирующей схемы LSM;
- перечень операций LSM;
- перечень выходных сигналов.

Выполнение лабораторной работы имеет 3 стадии: разработка поведенческой модели LSM, разработка структурной модели LSM и разработка испытательного стенда с проверкой функционирования LSM.

5.1. **Разработка поведенческой модели ICTR**.

Поведенческая модель ICTR описывается стилем потоков данных с использованием операций с целыми числами и функций из пакета `Cnetlist` . При этом используется редактор и компилятор VHDL, входящие в состав пакета Active HDL.

После того, как ICTR описана в виде архитектуры ICTR(BEH), она тестируется путем ручной подачи входных тестовых значений при моделировании этой архитектуры.

5.2. **Разработка структурной модели ICTR.**

Структурная поведенческая модель ICTR описывается структурным стилем, как это показано выше в примере. При этом используются компоненты из файла `Cnetlist_Lib.VHD` . Для этого также используется редактор и компилятор VHDL, входящие в состав пакета Active HDL.

5.3. **Разработка испытательного стенда и тестирование моделей.**

За образец испытательного стенда берется его пример, описанный в п.4.3. Он дорабатывается под требования конкретного испытуемого объекта.

При тестировании моделей по графикам сигналов определяется правильность функционирования моделей и измеряются задержки сигналов между входами и выходами структурной модели. Полученные графики сигналов переносятся в файл отчета с помощью функций выделения и сохранения в "кармане". По результатам тестирования формулируются выводы по лабораторной работе.

6. Отчет по лабораторной работе.

Отчет по лабораторной работе должен содержать:

- цель работы,
- описание варианта ICTR,
- ход синтеза моделей PLM или содержимого LUT,
- тексты описаний поведенческой и структурной моделей ICTR,
- графики сигналов, снятых на испытательном стенде,
- измеренные задержки сигналов,
- выводы.

7. Вопросы по лабораторной работе.

Каково функциональное назначение ICTR?

Каким образом управляют режимом работы ICTR?

Для чего нужны переменные generic в VHDL?

Покажите два способа вставки компонента в VHDL.

Объясните действие оператора **case**.

Каково назначение команды **assert**?

Почему логические схемы можно описывать стилем потоков данных, а схемы с регистрами – нельзя?

Какими способами можно выполнить описание устройства с триггерами?

Что выполняет функция конкатенации в VHDL?

Чем отличается описание аиснхронных и синхронных триггеров?

Почему в проектах рекомендуют использовать только синхронные триггеры?

Предложите модель JK-триггера.

Почему рекомендуется все триггеры подключать к одному источнику синхросерии?