Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Автоматизация проектирования вычислительных машин и систем

ОТЧЁТ
к лабораторной работе № 2
на тему

ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ УЗЛОВ ПОСЛЕДОВАТЕЛЬНОГО ТИПА НА ЯЗЫКЕ VHDL В СРЕДЕ VIVADO. ВАРИАНТ № 12

Студенты: гр. 050502
Т.С. Жук
А.В. Крачковский
Е.В. Кравченко

Проверил: В.В. Шеменков

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Приобрести навыки проектирования функциональных узлов последовательного типа (автоматов с памятью) и структурного описания устройств на языке VHDL.

2 ИСХОДНЫЕ ДАННЫЕ К РАБОТЕ

Для выполнения работы используем САПР Vivado 2018.2 и редактор кода Visual Studio Code с расширением (extension) Modern VHDL для подсветки синтаксиса языка VHDL. Для работы с логической диаграммой устройста использованием кроссплатформенное приложение Drawio.

В соответствии с полученным вариантом задания необходимо спроектировать устройство на языке VHDL, а также создать тестовый модуль для моделирования его работы. При этом моделирование должно как для конечного устройства, так и для его компонентов (триггеров).

Для создания тестового модуля необходимо воспользоваться графическим редактором тестовых воздействий. При этом тестовые воздействия должны быть полными. Для заданного устройства был взят пример моделирования из даташита. После создания тестового модуля необходимо выполнить поведенческое (Behavioural Simulation) моделирование работы устройства для всех способов его описания.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать модуль ‘entity’ заданного триггера на языке VHDL;
2. Разработать модуль ‘architecture’ для заданного триггера на языке VHDL;
3. Разработать модуль поведенческого моделирования (Behavioural Simulation) работы реализованного триггера на языке VHDL;
4. Провести поведенческое моделирование и проверить корректность работы реализованного триггера;
5. Разработать модуль ‘entity’ заданного устройства на языке VHDL;
6. Разработать модуль ‘architecture’ для заданного устройства на языке VHDL;
7. Разработать модуль поведенческого моделирования (Behavioural Simulation) работы реализованного устройства на языке VHDL;
8. Провести поведенческое моделирование и проверить корректность работы реализованного устройства.

3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Согласно заданному варианту необходимо разработать синхронный 4-х разрядный десятичный счётчик (SYNCHRONOUS 4-BIT UP/DOWN DECADE COUNTERS WITH 3-STATE OUTPUTS) согласно логической диаграмме представленной на рисунке ниже.

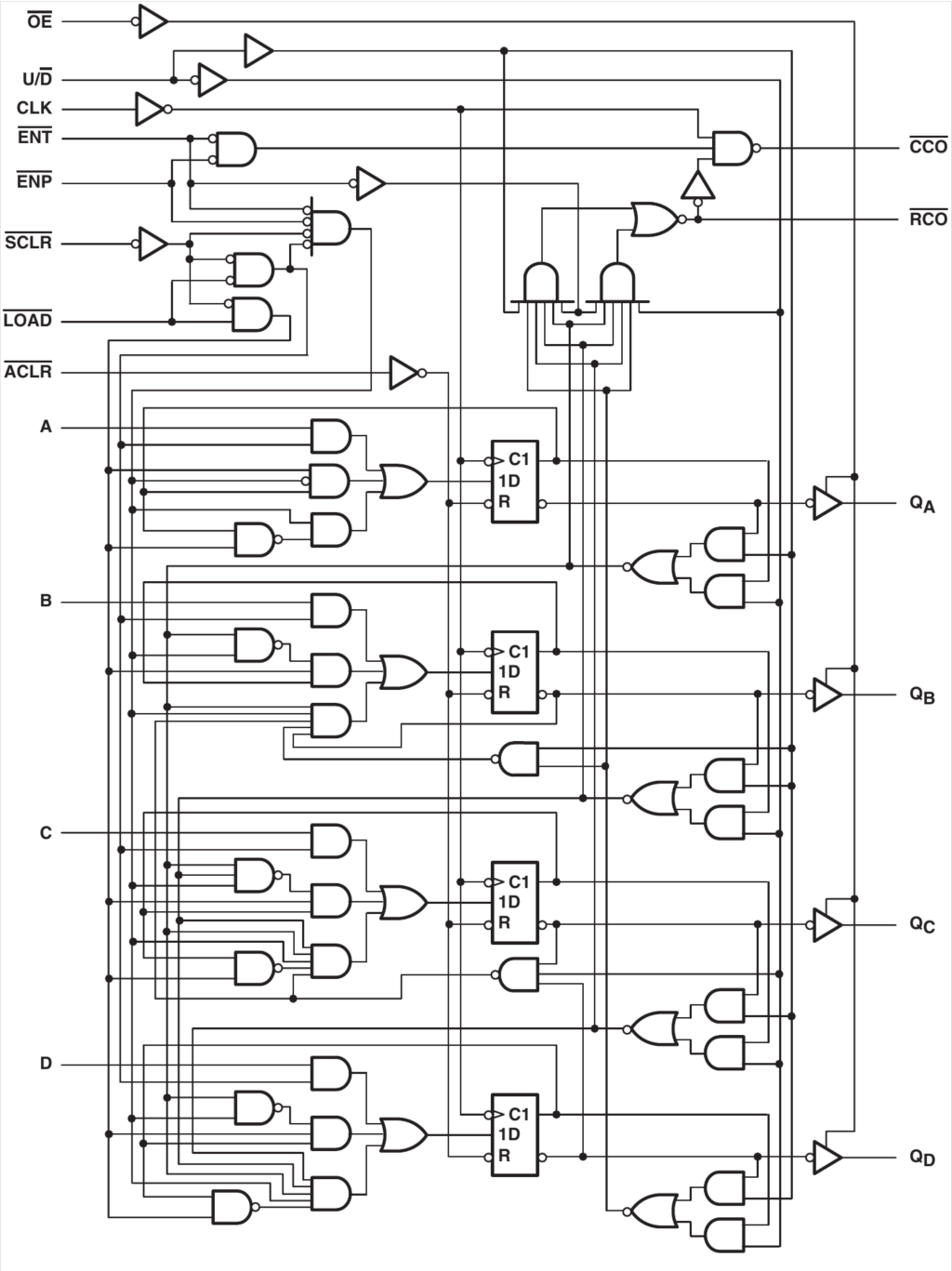


Рисунок 3.1 – Исходная логическая диаграмма заданного устройства

Для выполнения лабораторной работы необходимо провести детальный анализ логической диаграммы представленной на рисунке 3.1, выделить логические блоки и вспомогательные сигналы, а также

объединить входные сигналы ‘A, B, C, D’ и выходные сигналы ‘Qa, Qb, Qc, Qd’ в соответствующие входную и выходную шины ‘D’ и ‘Q’. Результаты данных преобразований приведены на рисунке ниже.

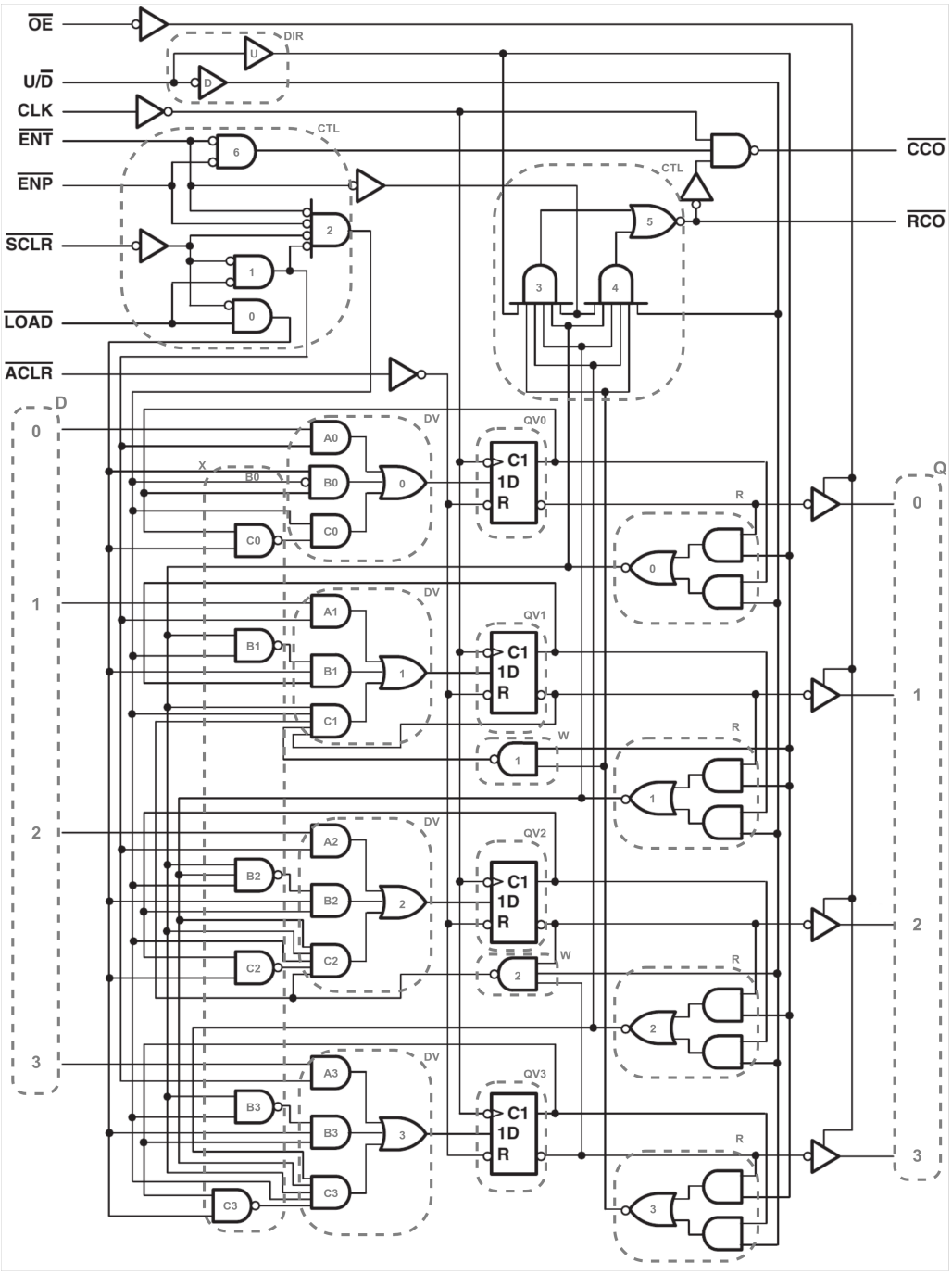


Рисунок 3.2 – Преобразованная логическая диаграмма заданного устройства

Функциональная таблица разрабатываемого устройства представлена на рисунке ниже.

FUNCTION TABLE

INPUTS								OPERATION
$\overline{OE}$	$\overline{ACLR}$	$\overline{SCLR}$	$\overline{LOAD}$	$\overline{ENT}$	$\overline{ENP}$	U/$\overline{D}$	CLK	
H	X	X	X	X	X	X	X	Q outputs disabled
L	L	X	X	X	X	X	X	Asynchronous clear
L	H	L	X	X	X	X	↑	Synchronous clear
L	H	H	L	X	X	X	↑	Load
L	H	H	H	L	L	H	↑	Count up
L	H	H	H	L	L	L	↑	Count down
L	H	H	H	H	X	X	X	Inhibit count
L	H	H	H	X	H	X	X	Inhibit count

Рисунок 3.3 – Функциональная таблица разрабатываемого устройства

Пример симуляции работы устройства представлен на странице 6 приложения А.

4 ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения лабораторной работы создаём проект с помощью встроенного менеджера проектов в САПР Vivado. С помощью графического интерфейса в режиме диалоговых окон создаём и добавляем в проект исходные файлы:

1. ‘DFlipFlop.tex’ – с описанием используемого в устройстве триггера;
2. ‘DFlipFlop_Behavioural.tex’ – с описанием модуля поведенческого моделирования работы описанного триггера;
3. ‘Counter.tex’ – с описанием модуля заданного устройства;
4. ‘Counter_Behavioural.tex’ – с описанием модуля поведенческого моделирования работы заданного устройства.

Исходный текст разработанных модулей представлен в приложениях с Б по Д.

4.1 Разработка модуля ‘entity’ заданного триггера на языке VHDL

Следуя правилам хорошего тона для достижения гибкости и возможности повторного использования кода в ‘entity DFlipFlop’ добавляем параметр ‘generic (INV_C: boolean; INV_R: boolean)’, который будет отвечать за инверсность входов триггера.

Добавляем триггеру порты ввода:

– порт C типа std_logic;
– порт D типа std_logic;
– порт R типа std_logic.

И порты вывода:

– порт Q типа std_logic.

4.2 Разработка модуля ‘architecture’ для заданного триггера на языке VHDL

Для реализации триггера используем атрибуты внутри оператора ‘process’ на языке VHDL:

```
process (C, R) begin
    if (R = '1' and not INV_R) or (R = '0' and INV_R) then
        -- Asynchronous Reset
        Q <= '0';
    elsif C'event and ((C = '1' and not INV_C) or (C = '0' and INV_C))
       then
        -- Synchronous load
        Q <= D;
    end if;
end process;
```

4.3 Разработка модуля поведенческого моделирования (Behavioural Simulation) работы реализованного триггера на языке VHDL

Для проведения поведенческого моделирования триггера во всех возможных комбинациях используем 4 entity DFlipFlop, которые будут иметь различные параметры ‘INV_C’ и ‘INV_R’, а также воспользуемся вспомогательной функцией ‘function to_std_logic(value: integer) return std_logic’ из лабораторной работы №1. Непосредственно само моделирование проводится в цикле:

```
for int_r in 0 to 1 loop
    for tmp_0 in 0 to REPEATS loop
        for int_d in 0 to 1 loop
            for int_c in 0 to 1 loop
                UUT_C <= to_std_logic(int_c);
                UUT_D <= to_std_logic(int_d);
                UUT_R <= to_std_logic(int_r);
                wait for DELAY;
            end loop;
        end loop;
    end loop;
end loop;
```

4.4 Проведение поведенческого моделирования и проверка корректности работы реализованного триггера

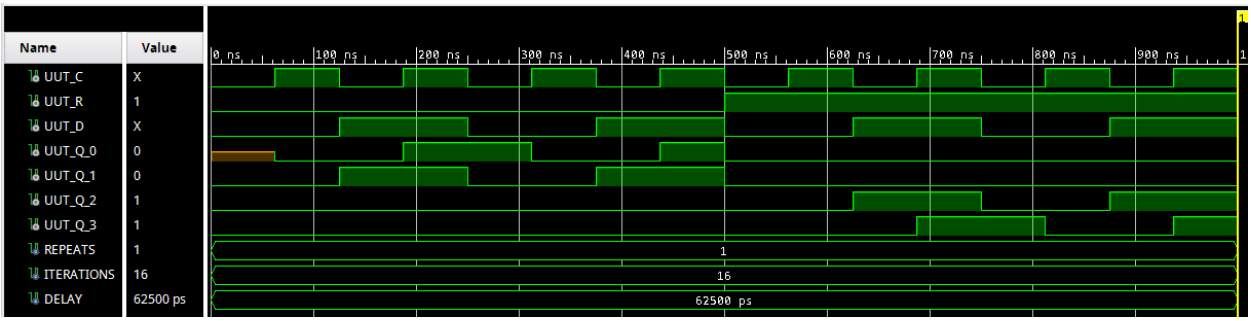


Рисунок 4.1 – Временная диаграмма поведенческого моделирования работы триггера

Заметим, что поведенческое моделрование отображает поведение триггера при любых сочетаниях значений ‘INV_C’ и ‘INV_R’.

4.5 Разработка модуля ‘entity’ заданного устройства на языке VHDL

Добавляем параметр ‘generic (WIDTH: positive := 4)’, чтобы использовать его как константу для размера шин входных и выходных портов.

Добавляем устройству порты ввода:

– порт OE типа std_logic;
– порт UD типа std_logic;
– порт CLK типа std_logic;
– порт ENT типа std_logic;
– порт ENP типа std_logic;
– порт SCLR типа std_logic;
– порт LOAD типа std_logic;
– порт ACLR типа std_logic;
– порт D типа std_logic_vector(WIDTH - 1 downto 0)
– порт R типа std_logic.

И порты вывода:

– порт CCO типа std_logic;
– порт RCO типа std_logic;
– порт Q типа std_logic_vector(WIDTH - 1 downto 0).

4.6 Разработка модуля ‘architecture’ для заданного устройства на языке VHDL

Анализ логической схемы устройства, проведённый в разделе ‘Теоретические сведения’ позволяет эффективно описывать устройство на языке VHDL с использованием вспомогательных внутренних (опорных) сигналов, указанных на рисунке 3.2.

```
-- Internal signals: DIR, CTL
signal DIR_U: std_logic;
signal DIR_D: std_logic;
signal CTL:   std_logic_vector(6 downto 0);
-- Internal signals: XB, DVC
signal XB:    std_logic_vector(WIDTH – 1 downto 0);
signal DVC:   std_logic_vector(WIDTH – 1 downto 0);
-- INternal signals XB, DV[A, B, C], DV, QV, R
signal XC:    std_logic_vector(WIDTH – 1 downto 0);
signal DVA:   std_logic_vector(WIDTH – 1 downto 0);
signal DVB:   std_logic_vector(WIDTH – 1 downto 0);
signal DV:    std_logic_vector(WIDTH – 1 downto 0);
signal QV:    std_logic_vector(WIDTH – 1 downto 0);
signal R:     std_logic_vector(WIDTH – 1 downto 0);
-- Internal signals: W
signal W:     std_logic_vector(2 downto 1);
```

Данный подход, выработанный в результате многократного переосмысления самых громоздких логических блоков позволяет, используя возможности языка VHDL ‘for ... generate’, ускорить процесс описания устройства и минимизировать количество написанного кода, что в свою очередь упрощает поиск ошибок в коде и саму вероятность их появления.

```
GENERATOR: for N in 0 to WIDTH - 1 generate
    XC(N) <= not (QV(N) and CTL(0));
    DVA(N) <= D(N) and CTL(1);
    DVB(N) <= XB(N) and CTL(0) and QV(N);
    DV(N) <= DVA(N) or DVB(N) or DVC(N);
    DFF: entity work.DFlipFlop(custom)
        generic map (INV_C => INV_C, INV_R => INV_R)
        port map (C => CLK, D => DV(N), R => ACLR, Q => QV(N));
    R(N) <= not ((not QV(N) and DIR_U) or (QV(N) and DIR_D));
end generate;
```

4.7 Разработка модуля поведенческого моделирования (Behavioural Simulation) работы реализованного устройства на языке VHDL

За основу для моделирования поведения устройства берём временную диаграмму из документации Texas Instruments на 7й странице приложения А.

Таким образом, написание модуля сводится к переносу значений входных сигналов из указанного выше источника (а точнее начальных значений сигналов и их изменений во времени) на сигналы моделирования такт за тактом. С использованием операторов ‘for ... loop’ и ‘if’ в операторе ‘process’ если это возожно и позволяет уменьшить объём написанного кода.

Например, за моделирование синхронного сброса и загрузки отвечает следующий участок кода:

```
-- Sync clear (2 CLK)
UUT_CLK  <= '1'; wait for DELAY;
UUT_SCLR <= '1'; UUT_LOAD <= '0';
UUT_UD   <= '1'; UUT_D    <= "0111";
UUT_CLK  <= '0'; wait for DELAY;
-- Sync load (2 CLK)
UUT_CLK  <= '1'; wait for DELAY;
UUT_SCLR <= '1'; UUT_LOAD <= '1';
UUT_ENP  <= '0'; UUT_ENT  <= '0';
UUT_UD   <= '1'; UUT_D    <= (others => IDC);
UUT_CLK  <= '0'; wait for DELAY;
```

4.8 Проведение поведенческого моделирования и проверка корректности работы реализованного устройства

Как можно заметить, получаемая нами временная диаграмма поведенческого моделирования работы устройства на рисунке ниже полностью соответсвует исходной временной диаграмме от Texas Instruments, это свидетельствует о том, что реализованное устройство работает корректно.

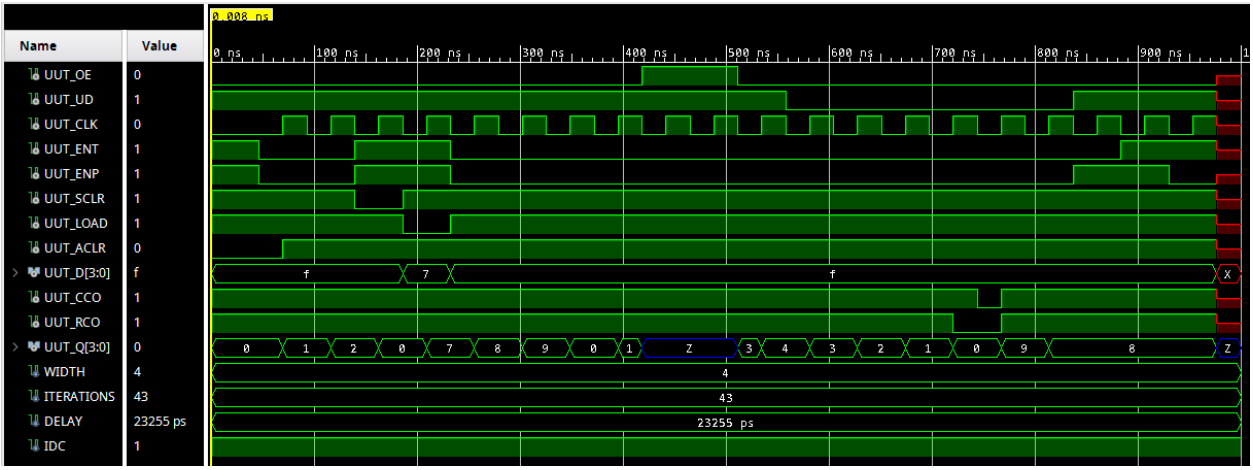


Рисунок 4.2 – Временная диаграмма поведенческого моделирования работы устройства

5 ВЫВОДЫ

В процессе выполнения работы мы приобрели навыки проектирования функциональных узлов последовательного типа (автоматов с памятью) и структурного описания устройств на языке VHDL, а также изучили принцип работы атрибутов в языке описания аппаратуры VHDL.

Полученные знания были применены для решения задач, возникших в ходе работы:

1. Разработки модуля ‘entity’ заданного триггера на языке VHDL;
2. Разработки модуля ‘architecture’ для заданного триггера на языке VHDL;
3. Разработки модуля поведенческого моделирования (Behavioural Simulation) работы реализованного триггера на языке VHDL;
4. Проведения поведенческого моделирования и проверки корректности работы реализованного триггера;
5. Разработки модуля ‘entity’ заданного устройства на языке VHDL;
6. Разработки модуля ‘architecture’ для заданного устройства на языке VHDL;
7. Разработки модуля поведенческого моделирования (Behavioural Simulation) работы реализованного устройства на языке VHDL;
8. Проведения поведенческого моделирования и проверки корректности работы реализованного устройства.

ПРИЛОЖЕНИЕ А

(*обязательное)*

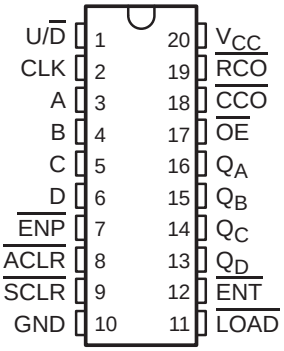
Документация Texas Instruments к разрабатываемому устройству

SN54ALS569A, SN74ALS568A, SN74ALS569A
SYNCHRONOUS 4-BIT UP/DOWN DECADE AND BINARY COUNTERS WITH 3-STATE OUTPUTS

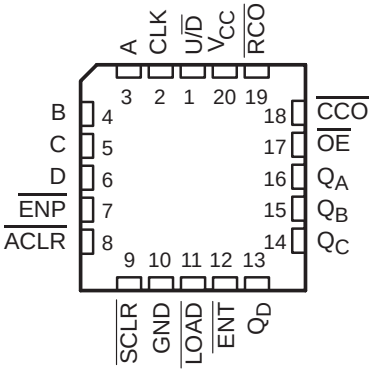
SDAS229A – APRIL 1982 – REVISED JANUARY 1995

- **3-State Q Outputs Drive Bus Lines Directly**
- **Counter Operation Independent of 3-State Output**
- **Fully Synchronous Clear, Count, and Load**
- **Asynchronous Clear Is Also Provided**
- **Fully Cascadable**
- **Package Options Include Plastic Small-Outline (DW) Packages, Ceramic Chip Carriers (FK), and Standard Plastic (N) and Ceramic (J) 300-mil DIPs**

SN54ALS569A . . . J PACKAGE
SN74ALS568A, SN74ALS569A . . . DW OR N PACKAGE
(TOP VIEW)



SN54ALS569A . . . FK PACKAGE
(TOP VIEW)



description

The SN74ALS568A decade counter and ′ALS569A binary counters are programmable, count up or down, and offer both synchronous and asynchronous clearing. All synchronous functions are executed on the positive-going edge of the clock (CLK) input.

The clear function is initiated by applying a low level to either asynchronous clear ($\overline{\text{ACLR}}$) or synchronous clear ($\overline{\text{SCLR}}$). Asynchronous (direct) clearing overrides all other functions of the device, while synchronous clearing overrides only the other synchronous functions. Data is loaded from the A, B, C, and D inputs by holding load ($\overline{\text{LOAD}}$) low during a positive-going clock transition. The counting function is enabled only when enable P ($\overline{\text{ENP}}$) and enable T ($\overline{\text{ENT}}$) are low and $\overline{\text{ACLR}}$, $\overline{\text{SCLR}}$, and $\overline{\text{LOAD}}$ are high. The up/down (U/$\overline{\text{D}}$) input controls the direction of the count. These counters count up when U/$\overline{\text{D}}$ is high and count down when U/$\overline{\text{D}}$ is low.

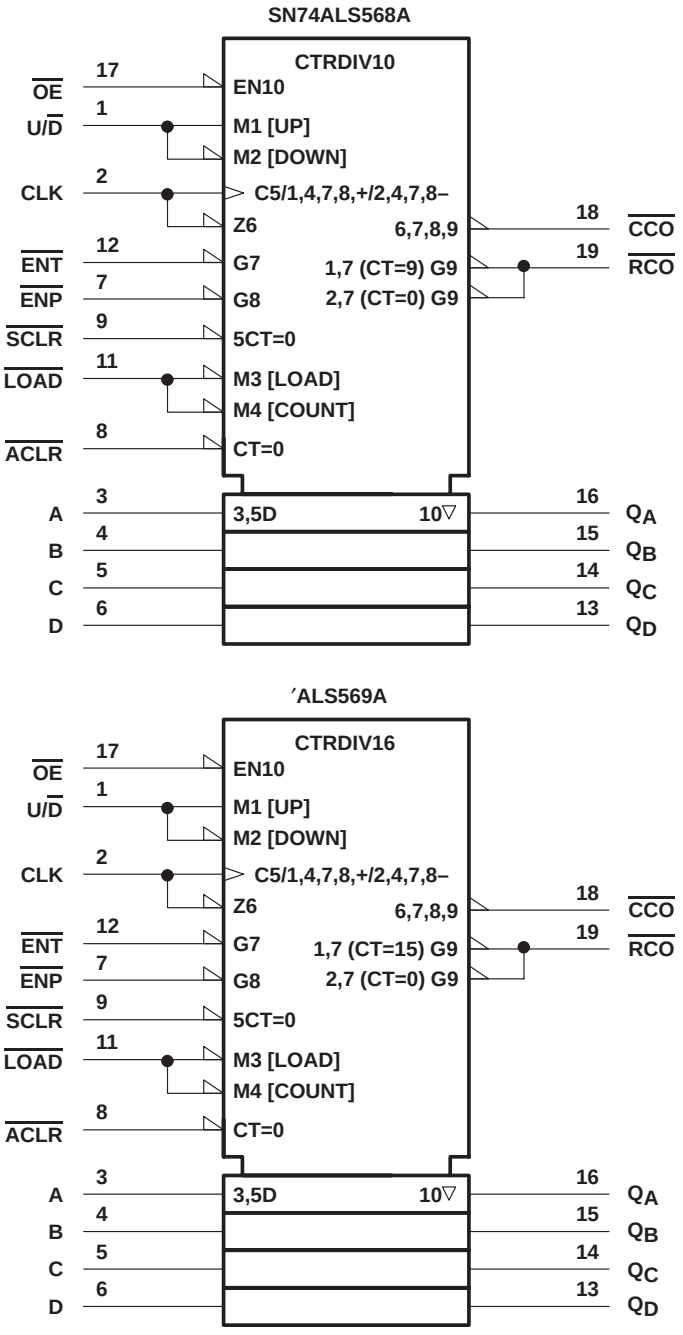
A high level at the output-enable ($\overline{\text{OE}}$) input forces the Q outputs into the high-impedance state, and a low level enables those outputs. Counting is independent of $\overline{\text{OE}}$. $\overline{\text{ENT}}$ is fed forward to enable the ripple-carry output ($\overline{\text{RCO}}$) to produce a low-level pulse while the count is zero (all Q outputs low) when counting down or maximum (9 or 15) when counting up. The clocked carry output ($\overline{\text{CCO}}$) produces a low-level pulse for a duration equal to that of the low level of the clock when $\overline{\text{RCO}}$ is low and the counter is enabled (both $\overline{\text{ENP}}$ and $\overline{\text{ENT}}$ are low); otherwise, $\overline{\text{CCO}}$ is high. $\overline{\text{CCO}}$ does not have the glitches commonly associated with a ripple-carry output. Cascading is normally accomplished by connecting $\overline{\text{RCO}}$ or $\overline{\text{CCO}}$ of the first counter to $\overline{\text{ENT}}$ of the next counter. However, for very high-speed counting, $\overline{\text{RCO}}$ should be used for cascading since $\overline{\text{CCO}}$ does not become active until the clock returns to the low level.

The SN54ALS569A is characterized for operation over the full military temperature range of –55°C to 125°C. The SN74ALS568A and SN74ALS569A are characterized for operation from 0°C to 70°C.

FUNCTION TABLE

INPUTS								OPERATION
$\overline{OE}$	$\overline{ACLR}$	$\overline{SCLR}$	$\overline{LOAD}$	$\overline{ENT}$	$\overline{ENP}$	U/$\overline{D}$	CLK	
H	X	X	X	X	X	X	X	Q outputs disabled
L	L	X	X	X	X	X	X	Asynchronous clear
L	H	L	X	X	X	X	↑	Synchronous clear
L	H	H	L	X	X	X	↑	Load
L	H	H	H	L	L	H	↑	Count up
L	H	H	H	L	L	L	↑	Count down
L	H	H	H	H	X	X	X	Inhibit count
L	H	H	H	X	H	X	X	Inhibit count

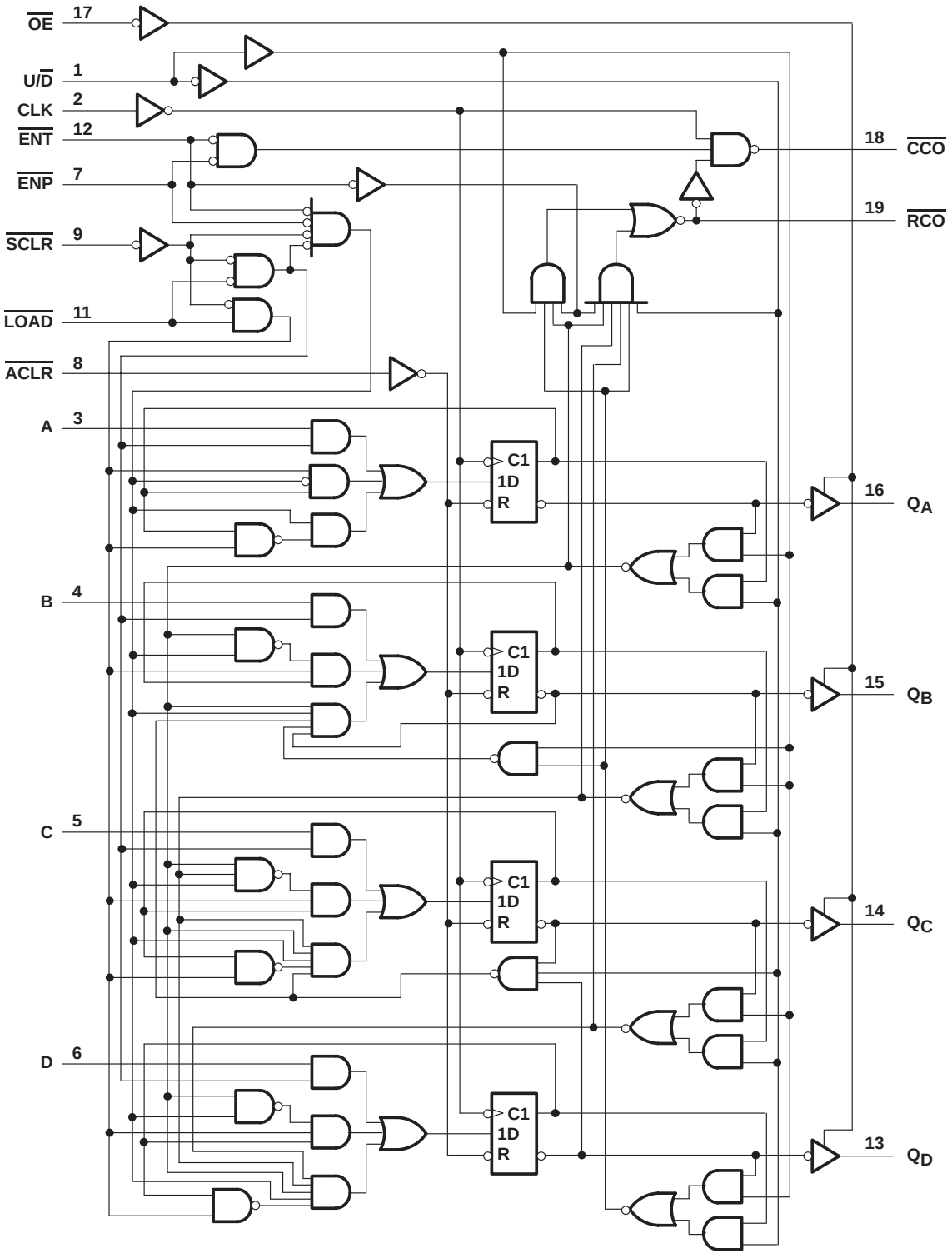
logic symbols†



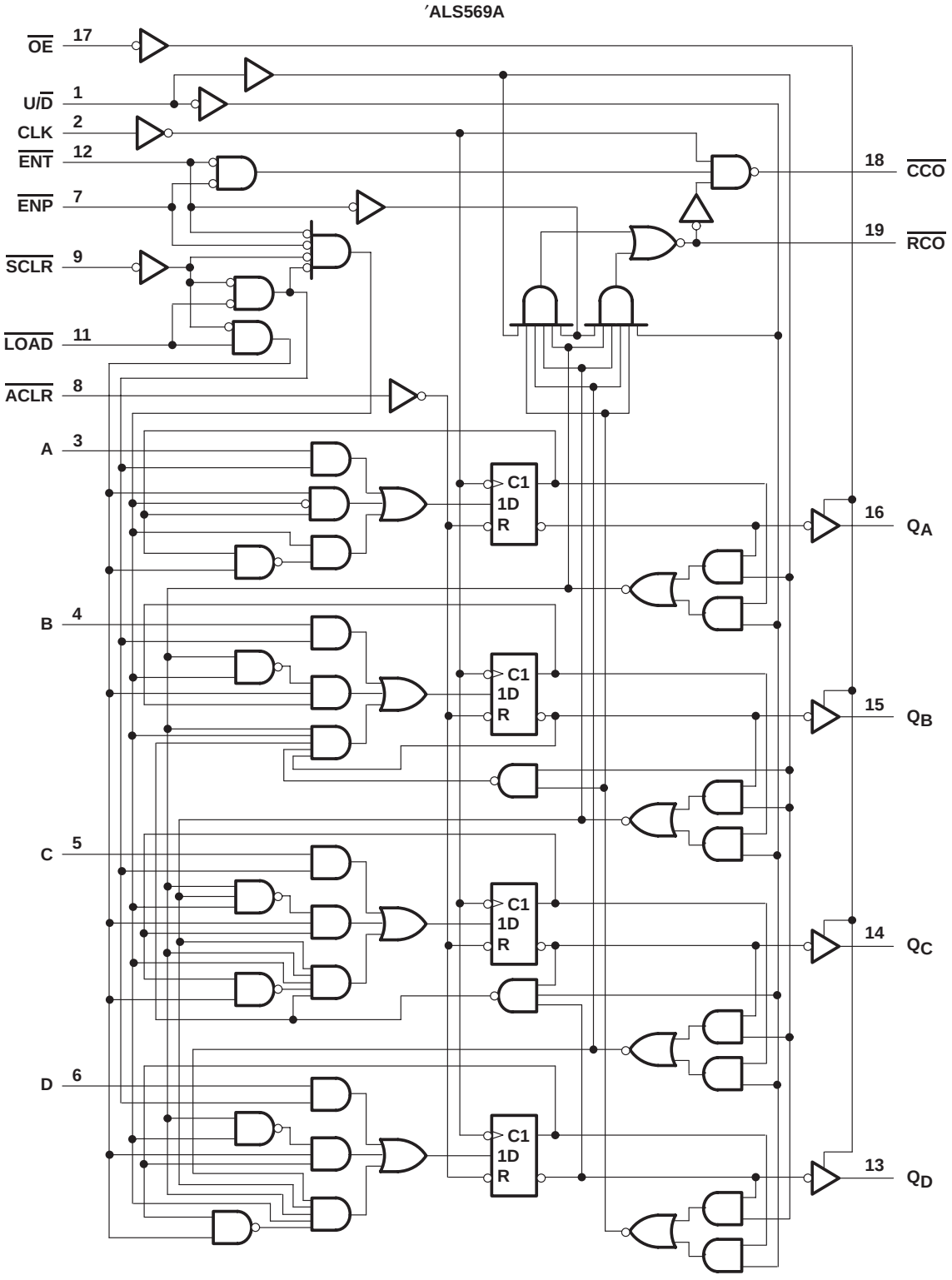
† These symbols are in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

logic diagrams (positive logic)

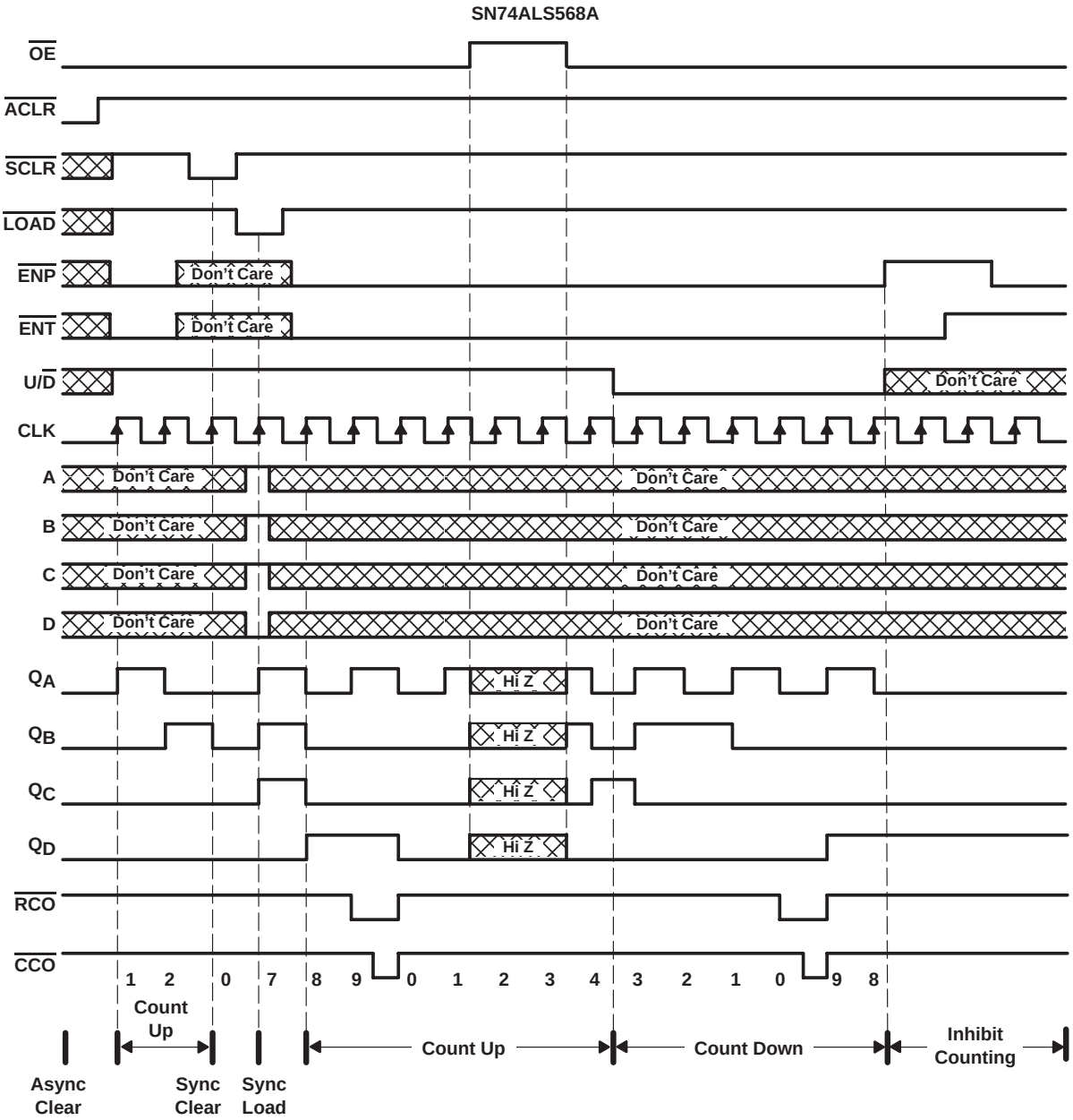
SN74ALS568A



logic diagrams (positive logic) (continued)

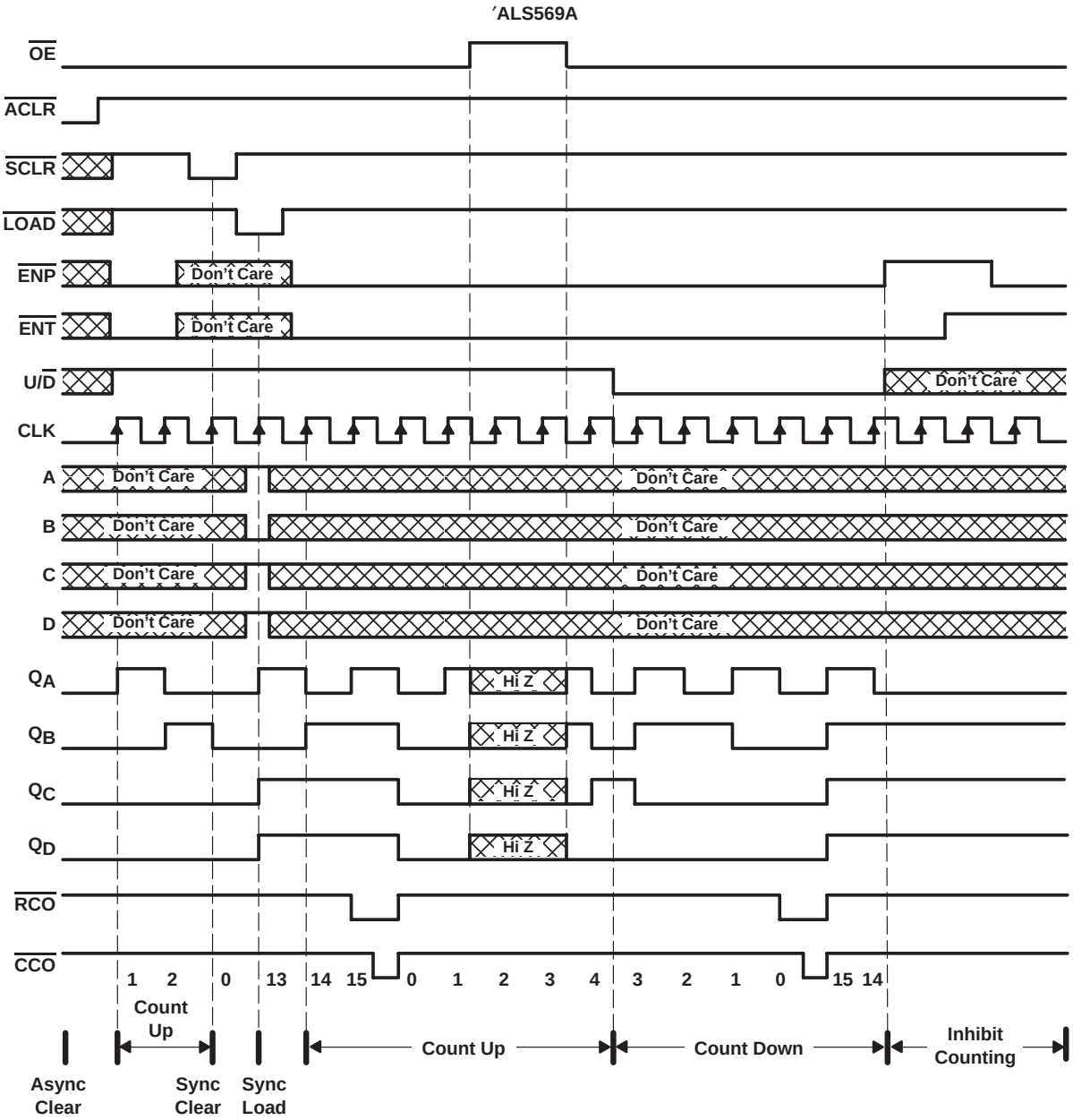


typical load, count, and inhibit sequences



TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

typical load, count, and inhibit sequences (continued)



ПРИЛОЖЕНИЕ Б

(*обязательное)*

Исходный текст модуля DFlipFlop

Файл ‘DFlipFlop.vhd’ содержит следующий код:

```
 1  ----------------------------------------------------------------------------------
 2  -- Company:
 3  -- Engineer:
 4  --
 5  -- Create Date: 11.10.2023 17:13:03
 6  -- Design Name:
 7  -- Module Name: DFlipFlop - custom
 8  -- Project Name:
 9  -- Target Devices:
10  -- Tool Versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------


20
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Category: SYNCHRONOUS 1-BIT D-FLIP-FLOP WITH ASYNCHRONOUS RESET
26  -- Implementation: custom DFlipFlop
27  entity DFlipFlop is
28      generic (
29          INV_C: boolean;
30          INV_R: boolean
31      );
32      port (
33          -- Input pins
34          C: in  std_logic;
35          D: in  std_logic;
36          R: in  std_logic;
37          -- Output pins
38          Q: out std_logic
39      );
40  end entity;
41
42  architecture custom of DFlipFlop is begin
43      process (C, R) begin
44          if (R = '1' and not INV_R) or (R = '0' and INV_R) then
45              -- Asynchronous Reset
46              Q <= '0';
47          elsif C'event and ((C = '1' and not INV_C) or (C = '0' and
                INV_C)) then
48              -- Synchronous load
49              Q <= D;
```

```
50          end if;
51      end process;
52  end architecture;
```

ПРИЛОЖЕНИЕ В

(*обязательное)*

Исходный текст модуля DFlipFlop_Behavioural

Файл ‘DFlipFlop_Behavioural.vhd’ содержит следующий код:

```
 1  ----------------------------------------------------------------------------------
 2  -- Company:
 3  -- Engineer:
 4  --
 5  -- Create Date: 11.10.2023 17:15:11
 6  -- Design Name:
 7  -- Module Name: DFlipFlop_Behavioural - simulation
 8  -- Project Name:
 9  -- Target Devices:
10  -- Tool Versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------

20
21
22  library IEEE;
23  -- use IEEE.NUMERIC_STD.ALL;
24  use IEEE.STD_LOGIC_1164.ALL;
25
26  entity DFlipFlop_Behavioural is
27      -- Generics
28      -- generic ();
29      -- Constants
30      constant REPEATS:    integer  := 1;
31      constant ITERATIONS: integer  := (2 ** 3) * (1 + REPEATS);
32      constant DELAY:      time     := 1000000 ps / ITERATIONS;
33  end entity;
34
35  architecture simulation of DFlipFlop_Behavioural is
36      -- Input signals
37      signal UUT_C:   std_logic := 'U';
38      signal UUT_R:   std_logic := 'U';
39      signal UUT_D:   std_logic := 'U';
40      -- Output signals
41      signal UUT_Q_0: std_logic := 'U';
42      signal UUT_Q_1: std_logic := 'U';
43      signal UUT_Q_2: std_logic := 'U';
44      signal UUT_Q_3: std_logic := 'U';
45      -- Utility functions
46      function to_std_logic(value: integer) return std_logic is begin
47          if    value = 0 then return '0';
48          elsif value = 1 then return '1';
49          else                 return 'X';
50          end if;
```

```
51      end function;
52  begin
53      -- Instantiate units under test (UUT-s)
54      -- Units are sorted by their generic flags `INV_[X, Y] -> 0bYX`
           using Gray codes
55      uut_dff_0: entity work.DFlipFlop(custom) -- Gray code: 0b00
56          generic map (INV_C => false, INV_R => false)
57          port map (C => UUT_C, R => UUT_R, D => UUT_D, Q => UUT_Q_0);
58      uut_dff_1: entity work.DFlipFlop(custom)-- Gray code: 0b01
59          generic map (INV_C => true, INV_R => false)
60          port map (C => UUT_C, R => UUT_R, D => UUT_D, Q => UUT_Q_1);
61      uut_dff_2: entity work.DFlipFlop(custom)-- Gray code: 0b11
62          generic map (INV_C => true, INV_R => true)
63          port map (C => UUT_C, R => UUT_R, D => UUT_D, Q => UUT_Q_2);
64      uut_dff_3: entity work.DFlipFlop(custom)-- Gray code: 0b10
65          generic map (INV_C => false, INV_R => true)
66          port map (C => UUT_C, R => UUT_R, D => UUT_D, Q => UUT_Q_3);
67      -- Simulation process
68      process begin
69          for int_r in 0 to 1 loop
70              for tmp_0 in 0 to REPEATS loop
71                  for int_d in 0 to 1 loop
72                      for int_c in 0 to 1 loop
73                          UUT_C <= to_std_logic(int_c);
74                          UUT_D <= to_std_logic(int_d);
75                          UUT_R <= to_std_logic(int_r);
76                          wait for DELAY;
77                      end loop;
78                  end loop;
79              end loop;
80          end loop;
81          UUT_C <= 'X';
82          UUT_D <= 'X';
83          UUT_D <= 'X';
84          wait;
85      end process;
86  end architecture;
```

ПРИЛОЖЕНИЕ Г

(*обязательное)*

Исходный текст модуля Counter

Файл ‘Counter.vhd’ содержит следующий код:

```
 1 ----------------------------------------------------------------------------------

 2 -- Company:
 3 -- Engineer:
 4 --
 5 -- Create Date: 11.10.2023 17:13:20
 6 -- Design Name:
 7 -- Module Name: Counter - custom
 8 -- Project Name:
 9 -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------

20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Category: SYNCHRONOUS 4-BIT UP/DOWN BINARY COUNTERS WITH 3-STATE
       OUTPUTS
26 -- THAT IS A LIE! IF FACT IT IS NOT BINARY BUT DECADE COUNTER (AKA BCD)
27 -- Implementation: custom Counter
28 entity Counter is
29     -- Generics
30     generic (
31         WIDTH: positive := 4
32     );
33     -- Ports
34     port (
35         -- Input pins (control)
36         OE:   in  std_logic;
37         UD:   in  std_logic;
38         CLK:  in  std_logic;
39         ENT:  in  std_logic;
40         ENP:  in  std_logic;
41         SCLR: in  std_logic;
42         LOAD: in  std_logic;
43         ACLR: in  std_logic;
44         -- Input pins (data)
45         D:    in  std_logic_vector(WIDTH - 1 downto 0);
46         -- Output pins (status)
47         CCO:  out std_logic;
48         RCO:  out std_logic;
49         -- Output pins (data)
```

```
50          Q:    out std_logic_vector(WIDTH - 1 downto 0)
51      );
52  end entity;
53
54  --
       |-----------------------------------------------------------------------|

55  -- |                              Function table
                                    |
56  --
       |----|------|------|------|-----|-----|-----|-----|--------------------|

57  -- | __ | ____ | ____ | ____ | ___ | ___ |   _ |     |
                              |
58  -- | OE | ACLR | SCLR | LOAD | ENT | ENP | U/D | CLK | Operation
                 |
59  --
       |----|------|------|------|-----|-----|-----|-----|--------------------|

60  -- | H  |  X   |  X   |  X   |  X  |  X  |  X  |  X  | Q outputs
       disabled |
61  -- | L  |  L   |  X   |  X   |  X  |  X  |  X  |  X  | Asynchronous
       clear |
62  -- | L  |  H   |  L   |  X   |  X  |  X  |  X  | ↑   | Synchronous
       clear  |
63  -- | L  |  H   |  H   |  L   |  X  |  X  |  X  | ↑   | Load
                     |
64  -- | L  |  H   |  H   |  H   |  L  |  L  |  H  | ↑   | Count up
                   |
65  -- | L  |  H   |  H   |  H   |  L  |  L  |  L  | ↑   | Count down
                |
66  -- | L  |  H   |  H   |  H   |  H  |  X  |  X  |  X  | Inhibit count
              |
67  -- | L  |  H   |  H   |  H   |  X  |  H  |  X  |  X  | Inhibit count
              |
68  --
       |----|------|------|------|-----|-----|-----|-----|--------------------|

69  architecture custom of Counter is
70      -- Constants
71      constant INV_C: boolean := false;
72      constant INV_R: boolean := true;
73      -- Internal signals: DIR, CTL
74      signal DIR_U: std_logic;
75      signal DIR_D: std_logic;
76      signal CTL:   std_logic_vector(6 downto 0);
77      -- Internal signals: XB, DVC
78      signal XB:    std_logic_vector(WIDTH - 1 downto 0);
79      signal DVC:   std_logic_vector(WIDTH - 1 downto 0);
80      -- INternal signals XB, DV[A, B, C], DV, QV, R
81      signal XC:    std_logic_vector(WIDTH - 1 downto 0);
82      signal DVA:   std_logic_vector(WIDTH - 1 downto 0);
83      signal DVB:   std_logic_vector(WIDTH - 1 downto 0);
84      signal DV:    std_logic_vector(WIDTH - 1 downto 0);
```

```
85      signal QV:    std_logic_vector(WIDTH - 1 downto 0);
86      signal R:     std_logic_vector(WIDTH - 1 downto 0);
87      -- Internal signals: W
88      signal W:     std_logic_vector(2 downto 1);
89  begin
90      -- DIR
91      DIR_U <= UD;
92      DIR_D <= not UD;
93      -- CTL
94      CTL(0) <= SCLR and LOAD;
95      CTL(1) <= SCLR and not LOAD;
96      CTL(2) <= not ENT and not ENP and SCLR and not CTL(1);
97      CTL(3) <= DIR_U and R(3) and R(2) and R(1) and R(0) and not ENT;
98      CTL(4) <= not ENT and R(0) and R(1) and R(2) and R(3) and DIR_D;
99      CTL(5) <= not (CTL(3) or CTL(4));
100     CTL(6) <= not ENT and not ENP;
101     -- XB
102     XB(0) <= not CTL(2);
103     XB(1) <= not (R(0) and CTL(2));
104     XB(2) <= not (R(0) and R(1) and CTL(2));
105     XB(3) <= not (R(0) and CTL(2));
106     -- DVC
107     DVC(0) <= CTL(2) and XC(0);
108     DVC(1) <= R(0) and CTL(2) and W(2) and W(1) and not QV(1);
109     DVC(2) <= R(1) and R(0) and CTL(2) and XC(2) and W(2);
110     DVC(3) <= R(2) and R(1) and R(0) and CTL(2) and XC(3);
111
112     -- XC, DV[A, B], DV, QV, R
113     GENERATOR: for N in 0 to WIDTH - 1 generate
114         XC(N) <= not (QV(N) and CTL(0));
115         DVA(N) <= D(N) and CTL(1);
116         DVB(N) <= XB(N) and CTL(0) and QV(N);
117         DV(N) <= DVA(N) or DVB(N) or DVC(N);
118         DFF: entity work.DFlipFlop(custom)
119             generic map (INV_C => INV_C, INV_R => INV_R)
120             port map (C => CLK, D => DV(N), R => ACLR, Q => QV(N));
121         R(N) <= not ((not QV(N) and DIR_U) or (QV(N) and DIR_D));
122     end generate;
123
124     -- W
125     W(1) <= not (DIR_U and R(3));
126     W(2) <= not (not QV(2) and DIR_D and not QV(3));
127     -- CCO
128     CCO <= not (not CLK and CTL(6) and not CTL(5));
129     -- RCO
130     RCO <= CTL(5);
131     -- Q
132     OUTPUT: for N in 0 to WIDTH - 1 generate
133         Q(N) <= QV(N) when OE = '0' else 'Z';
134     end generate;
135 end architecture;
```

ПРИЛОЖЕНИЕ Д
(*обязательное)*

Исходный текст модуля Counter_Behavioural

Файл ‘Counter_Behavioural.vhd’ содержит следующий код:

```
----------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date: 11.10.2023 17:14:02
-- Design Name:
-- Module Name: Counter_Behavioural - simulation
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
-- use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_1164.ALL;

entity Counter_Behavioural is
    -- Generics
    generic (
        WIDTH: positive := 4
    );
    -- Constants
    constant ITERATIONS: integer  := 3 + 4 + 2 + 2 + 13 + 12 + 6 + 1;
    constant DELAY:      time     := 1000000 ps / ITERATIONS;
end entity;

architecture simulation of Counter_Behavioural is
    -- Simulation constants (do not care value)
    constant IDC:    std_logic := '1';
    -- Input signals (control)
    signal UUT_OE:   std_logic := 'U';
    signal UUT_UD:   std_logic := 'U';
    signal UUT_CLK:  std_logic := 'U';
    signal UUT_ENT:  std_logic := 'U';
    signal UUT_ENP:  std_logic := 'U';
    signal UUT_SCLR: std_logic := 'U';
    signal UUT_LOAD: std_logic := 'U';
    signal UUT_ACLR: std_logic := 'U';
    -- Input signals (data)
    signal UUT_D:    std_logic_vector(WIDTH - 1 downto 0) := (others =>
        'U');
```

```
50      -- Output signals (status)
51      signal UUT_CCO:  std_logic := 'U';
52      signal UUT_RCO:  std_logic := 'U';
53      -- Output signals (data)
54      signal UUT_Q:    std_logic_vector(WIDTH - 1 downto 0) := (others =>
            'U');
55  begin
56      -- Instantiate units under test (UUT-s)
57      UUT: entity work.Counter(custom)
58          generic map (WIDTH => WIDTH)
59          port map (
60              OE => UUT_OE,
61              UD => UUT_UD,
62              CLK => UUT_CLK,
63              ENT => UUT_ENT,
64              ENP => UUT_ENP,
65              SCLR => UUT_SCLR,
66              LOAD => UUT_LOAD,
67              ACLR => UUT_ACLR,
68              D => UUT_D,
69              CCO => UUT_CCO,
70              RCO => UUT_RCO,
71              Q => UUT_Q
72          );
73      -- Simulation process
74      process begin
75          -- Async clear (3 CLK)
76          UUT_OE   <= '0'; UUT_ACLR <= '0';
77          UUT_SCLR <= IDC; UUT_LOAD <= IDC;
78          UUT_ENP  <= IDC; UUT_ENT  <= IDC;
79          UUT_UD   <= IDC; UUT_D    <= (others => IDC);
80          for N in 1 to 3 loop
81              if N = 3 then
82                  UUT_ENP  <= '0'; UUT_ENT  <= '0';
83              end if;
84              UUT_CLK  <= '0'; wait for DELAY;
85          end loop;
86          -- Count up (4 CLK)
87          UUT_OE   <= '0'; UUT_ACLR <= '1';
88          UUT_SCLR <= '1'; UUT_LOAD <= '1';
89          -- UUT_ENP  <= '0'; UUT_ENT  <= '0'; (Moved up!)
90          UUT_UD   <= '1'; UUT_D    <= (others => IDC);
91          UUT_CLK  <= '1'; wait for DELAY;
92          UUT_CLK  <= '0'; wait for DELAY;
93          UUT_CLK  <= '1'; wait for DELAY;
94          UUT_SCLR <= '0'; UUT_LOAD <= '1';
95          UUT_ENP  <= IDC; UUT_ENT  <= IDC;
96          UUT_CLK  <= '0'; wait for DELAY;
97          -- Sync clear (2 CLK)
98          UUT_CLK  <= '1'; wait for DELAY;
99          UUT_SCLR <= '1'; UUT_LOAD <= '0';
100         UUT_UD   <= '1'; UUT_D    <= "0111";
101         UUT_CLK  <= '0'; wait for DELAY;
102         -- Sync load (2 CLK)
```

```
103         UUT_CLK  <= '1'; wait for DELAY;
104         UUT_SCLR <= '1'; UUT_LOAD <= '1';
105         UUT_ENP  <= '0'; UUT_ENT  <= '0';
106         UUT_UD   <= '1'; UUT_D    <= (others => IDC);
107         UUT_CLK  <= '0'; wait for DELAY;
108         -- Count up (13 CLK)
109         UUT_OE   <= '0'; UUT_ACLR <= '1';
110         UUT_SCLR <= '1'; UUT_LOAD <= '1';
111         UUT_ENP  <= '0'; UUT_ENT  <= '0';
112         UUT_UD   <= '1'; UUT_D    <= (others => IDC);
113         for N in 1 to 7 loop
114             UUT_CLK <= '1'; wait for DELAY;
115             if N = 4 then
116                 UUT_OE   <= '1'; UUT_ACLR <= '1';
117             elsif N = 6 then
118                 UUT_OE   <= '0'; UUT_ACLR <= '1';
119             end if;
120             if not (N = 7) then
121                 UUT_CLK <= '0'; wait for DELAY;
122             end if;
123         end loop;
124         -- Count down (12 CLK)
125         UUT_UD   <= '0'; UUT_D    <= (others => IDC);
126         for N in 1 to 6 loop
127             UUT_CLK  <= '0'; wait for DELAY;
128             UUT_CLK  <= '1'; wait for DELAY;
129         end loop;
130         -- Inhibit counting (6 CLK)
131         UUT_UD   <= IDC; UUT_D    <= (others => IDC);
132         for N in 1 to 3 loop
133             if N = 3 then UUT_ENP <= '0'; else UUT_ENP <= '1'; end if;
134             if N = 1 then UUT_ENT <= '0'; else UUT_ENT <= '1'; end if;
135             UUT_CLK  <= '0'; wait for DELAY;
136             UUT_CLK  <= '1'; wait for DELAY;
137         end loop;
138         -- The end (1 CLK)
139         UUT_OE   <= 'X'; UUT_ACLR <= 'X';
140         UUT_SCLR <= 'X'; UUT_LOAD <= 'X';
141         UUT_ENP  <= 'X'; UUT_ENT  <= 'X';
142         UUT_UD   <= 'X'; UUT_D    <= (others => 'X');
143         UUT_CLK  <= 'X'; wait for DELAY;
144         wait;
145     end process;
146 end architecture;
```