目录

第一章 研究的背景和意义

在当今数字化时代，计算机科学与技术的飞速发展对计算机体系结构提出了更高的要求，尤其是在处理大数据和高性能计算方面。乘法运算作为计算机中的基本运算之一，其效率和性能直接关系到许多应用的执行速度。在很多情况下，计算机或电子设备需要进行大量的乘法运算。例如，在算术逻辑单元（ALU）中，乘法是基本运算之一。为了处理乘法操作，需要一个高效且可靠的乘法器。乘法是相对复杂和耗时的运算，特别是在传统的电子电路中。设计一个快速的乘法器是为了提高计算速度，尤其在对大量数据进行处理时尤为重要。8 位乘 8 位乘法器意味着输入为两个 8 位的数字。这需要一个电路结构来将这两个 8 位数字进行乘法运算，并产生一个 16 位的输出，这个输出是乘法结果的展开形式。在很多计算任务中，需要处理小规模的数据。8 位乘法器提供了一种针对这种规模数据进行乘法运算的方法。本研究致力于设计一套指令系统，利用累加器实现 8*8 位乘法运算，旨在通过优化硬件结构和指令集，提高计算机在乘法运算方面的效率，具有重要的研究意义。

研究背景：

随着计算机科技的发展，设计更高效、更快速的乘法运算方法是计算机体系结构领域长期以来的研究重点之一。计算机体系结构是计算机科学领域中的一个核心分支，关注计算机硬件与软件之间的接口及其相互作用。在计算机体系结构中，乘法运算是一项常见而基础的运算，广泛应用于图形处理、信号处理、加密算法等众多领域。传统的乘法器通常占用大量硬件资源，设计一种高效而精简的指令系统，能够在保证计算准确性的前提下加速乘法运算，对于提高计算机整体性能具有重要作用。

意义：

资源节约和性能优化： 在硬件资源有限的情况下，设计一种有效的乘法实现方式可以节省芯片面积和功耗。通过优化累加器的使用，可以在更短的时钟周期内完成乘法运算，提高计算速度和效率。这种优化有助于设计更紧凑、更经济的硬件结构，降低成本，提高设备性能。

扩展性和适用性： 针对大位宽数据的乘法运算是众多计算任务中的一个子集。设计一套能够扩展到不同位宽且适用于不同场景的指令系统，可以为多种应用提供高效的运算支持。

教育和学术研究价值： 这样的研究也有助于教育领域，让学生更好地理解计算机体系结构中乘法运算的实现原理。同时，对于学术研究而言，探索不同的乘法实现方式有助于推动计算机体系结构领域的发展。

通过设计一套指令系统，利用累加器来实现 8*8 位乘法运算，可以在硬件实现层面上探索新的方法，为计算机体系结构的发展和优化提供新的思路和可能性。

应用领域拓展： 高效的乘法运算对于图像处理、音频处理、密码学等领域至关重要。通过设计一套通用性强、效率高的指令系统，可以拓展计算机在各种应用领域的适用范围，推动科技创新。

社会影响： 在当前社会日益数字化的背景下，优化计算机硬件结构不仅可以提高计算效率，还有助于减少资源消耗，降低能源浪费，符合可持续发展的理念，具有积极的社会影响。

第二章 相关知识介绍

2.1 二进制乘法的算法

乘法器是由 8 位二进制加法器构成的以时序方式设计的乘法器，通过逐项移位相加原理来实现。用乘数的各位数码，从低位开始依次与被乘数相乘，每相乘一次得到的积称为部分积，将第一次（由乘数最低位与被乘数相乘）得到的部分积右移一位并与第二次得到的部分积相加，将加得的和右移一位再与第三次得到的部分积相加，再将相加的结果右移一位与第四次得到的部分积相加。直到所有的部分积都被加过一次即可得到相应的乘积。

二进制乘法如图所示：

```
   11010101
×  10010011
   11010101    N0与被乘数相乘的部分积，部分积右移一位
    11010101   N1与被乘数相乘的部分积
+  11010101
  1001111111
   1001111111  两个部分积之和，部分积之和右移一位
+  00000000    N2与被乘数相乘的部分积
  0100111111
  0100111111  与前面部分积之和相加,部分积之和右移一
+ 00000000    N4与被乘数相乘的部分积
   ...
   ...          N7与被乘数相乘的部分积
+ 11010101
  1111010010111110   与前面部分积之和相加
  0111101001011111   右移一位得到最后的积
```

图 2-1 二进制乘法图

2.2 设计原理

由加法器构成的时序逻辑方式的乘法器的原理是，通过逐项移位相加原理来实现，从被乘数的最低位开始，若为，若为 1，则乘数左移与上一次和相加；若

为 0，左移后以全零相加，直至被乘数的最高位。

在下图中。ARICTL 是乘法运算控制电路，它的 START 信号的上升沿与高电平有两个功能，即 16 位寄存器清 0 和被乘数 A 向移位寄存器 SREG8B 加载；它的低电平则作为乘法使能信号。

CLK 为乘法时钟信号，当被乘数加载于 8 位右移寄存器 SEG8B 后，在时钟同步下由低位至高位逐位移出，当其为 1 时，与门 ANDARITH 打开，8 位乘数 B 在同一节拍进入 8 位加法器，与上一节拍锁存在 16 位锁存器 REG26B 中的高 8 位进行相加，其和在下一时钟节拍的上升沿被锁存进此锁存器；而当被乘数的移出位为 0 时，与门全 0 输出。如此往复，直至 8 个时钟脉冲后，乘法运算过程中止，此时 REG16B 的输出值即为最后乘积。原理图如下：

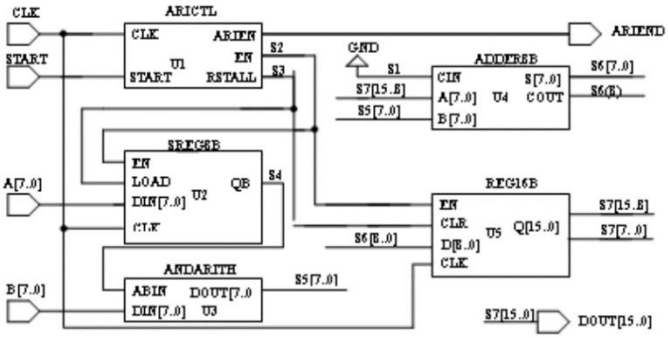

图 2-2 乘法原理图

第三章 整体方案

3.1 电路方案

此电路由五部分组成它们，分别是控制器，锁存器，寄存器，乘法器，加法器。

1.控制器是一个乘法器的控制模块，用来接受实验系统上的连续脉冲。

2.锁存器起锁存的作用，它可以锁存 8 位乘数。

3.移位寄存器起移位的作用，便于被乘数可以逐位移出。

4.乘法器功能类似一个特殊的与非门。

5.加法器用于 8 位乘数和高 8 位相加。

电路整体方案如下：

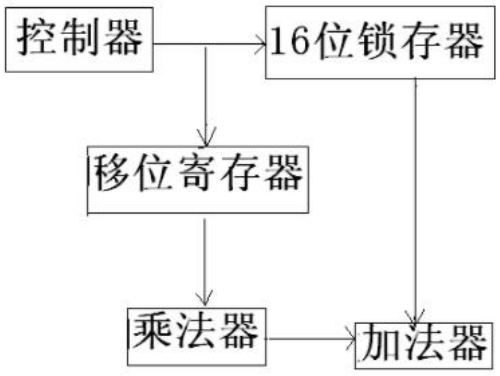

图 3-1 乘法器系统框图

3.2 仿真方案

本次设计采用 Quartus II design 软件和 Modelsim 联合仿真。

Quartus II 作为一种可编程逻辑的设计环境，由于其强大的设计能力和直观易用的接口，越来越受到数字系统设计者的欢迎。

ModelSim 是业界最优秀的 HDL 语言仿真软件，它能提供友好的仿真环境，是业界唯一的单内核支持 VHDL 和 Verilog 混合仿真的仿真器。它采用直接优化

的编译技术、Tcl/Tk 技术、和单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段，是 FPGA/ASIC 设计的首选仿真软件。

1.使用 Quartus II 和 Modelsim 的联合仿真功能

2.用 VHDL 语言描述单元电路。

3.利用软件对单元电路进行编译、仿真调试。

图 3-2 仿真软件图

第四章 硬件部分设计

4.1 硬件部分系统框图

以下为硬件部分的系统框图：

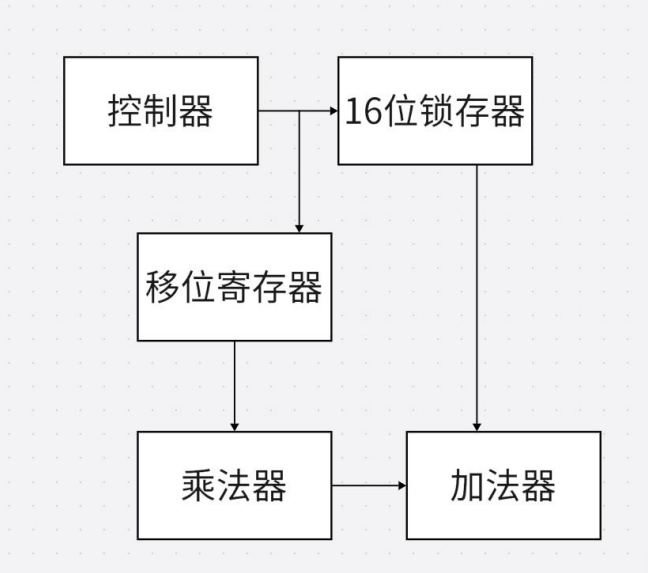

图 4-1 硬件系统框图

4.2 控制器设计

控制器是一个乘法器的控制模块，用来接受实验系统上的连续脉冲。 ARICTL 用于生成时钟控制信号。该模块具有时钟输入 CLK、启动信号 START，并产生时钟输出 CLKOUT、复位停止信号 RSTALL 以及算术操作结束信号 ARIEND。

输入端口：时钟信号 CLK 和启动信号 START。

输出端口：时钟输出信号 CLKOUT、复位停止信号 RSTALL 以及算术操作结束信号 ARIEND。

在 START 为高电平时，CLKOUT 在计数器小于 8 时为高电平，之后为低电平，同时 ARIEND 在计数器小于 8 时为低电平，之后为高电平。

用于控制算术运算的时钟脉冲，例如乘法器。在乘法器中，需要一定数量的时钟周期来完成乘法操作，用于生成正确的时钟信号以控制乘法器

的运行。

以下为控制器硬件图：

图 4-2 控制器硬件图

4.3 移位寄存器设计

移位寄存器起移位的作用，便于被乘数可以逐位移出。模块具有时钟输入信号 CLK、加载使能信号 LOAD、移位使能信号 EN、8 位数据输入 DIN 和一个单独的位输出 QB。

输入端口：时钟信号 CLK、加载使能信号 LOAD、移位使能信号 EN 以及 8 位的数据输入 DIN。

输出端口：单个位 QB。

CLK, LOAD, EN 是输入端口，分别表示时钟信号、加载信号和使能信号。

DIN 是输入端口，是一个 8 位的标准逻辑向量，表示要加载到寄存器的输入数据。

QB 是输出端口，表示寄存器的最低位。

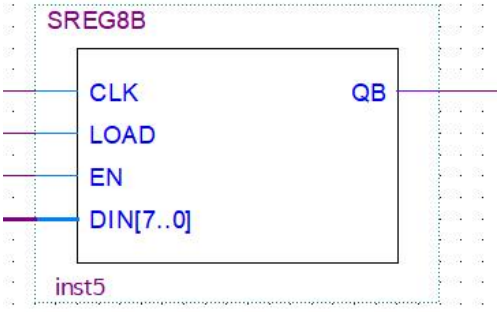

图 4-3 移位寄存器硬件图

4.4 乘法器设计

乘法器功能类似一个特殊的与门。

ABIN 是输入端口，是一个标准逻辑信号，用于与输入数据进行逻辑与运算。

DIN 是输入端口，是一个 8 位的标准逻辑向量，表示待进行逻辑与运算的数据。

DOUT 是输出端口，是一个 8 位的标准逻辑向量，表示逻辑与运算的结果。

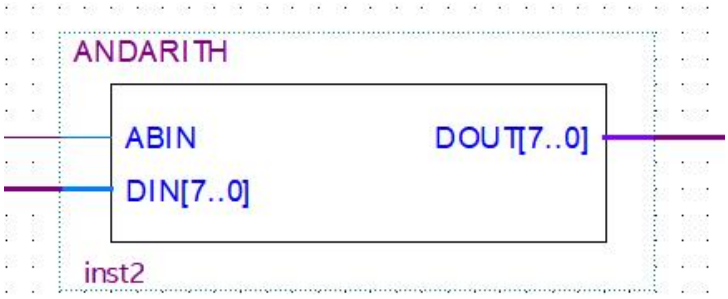

图 4-4 乘法器硬件图

4.5 加法器设计

加法器用于 8 位乘数和高 8 位相加。该模块接受两个 8 位的输入向量 A 和 B，以及一个进位输入信号 CIN。输出包括一个 8 位的和 S1 和一个进位输出信号 COUT。

输入端口：进位输入信号 CIN 和两个 8 位的数据输入向量 A 和 B。

输出端口：一个 8 位的和 S1 和一个进位输出信号 COUT。

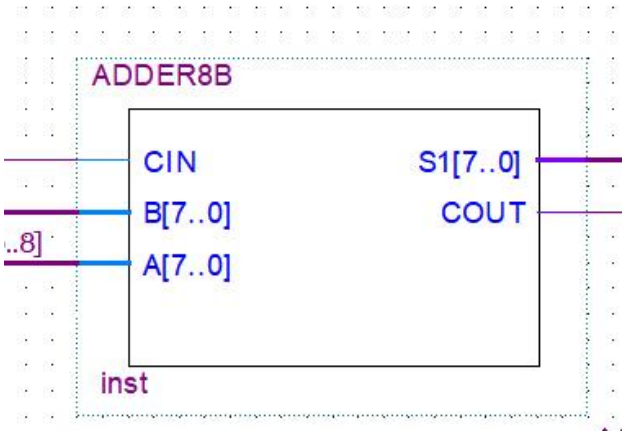

图 4-5 加法器硬件图

4.6 锁存器设计

锁存器起锁存的作用，它可以锁存 8 位乘数。该寄存器有一个时钟信号 CLK、清零信号 CLR、使能信号 EN 以及一个 9 位数据输入 D，并输出一个 16 位的结果 Q。

输入端口：时钟信号 CLK、清零信号 CLR、使能信号 EN 以及 9 位的数据输入 D。

输出端口：16 位的结果 Q。

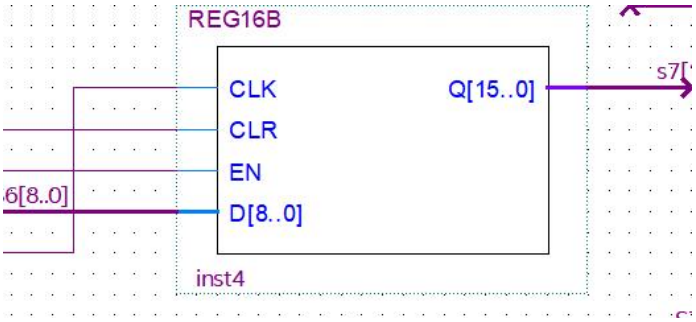

图 4-6 锁存器硬件图

第五章 软件部分设计

5.1 控制器代码

以下是控制器部分的代码：

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4  ENTITY ARICTL IS
5  PORT (           CLK,START:IN STD_LOGIC;
6         CLKOUT,RSTALL,ARIEND:OUT STD_LOGIC);
7  END ENTITY ARICTL;
8  ARCHITECTURE ART5 OF ARICTL IS
9  SIGNAL CNT4B:STD_LOGIC_VECTOR(3DOWNTO 0);
10 BEGIN
11    RSTALL<=START;
12    PROCESS (CLK,START) IS
13    BEGIN
14      IF START = '1' THEN CNT4B<= "0000";
15      ELSIF CLK'EVENT AND CLK = '1' THEN
16        IF CNT4B<8 THEN
17          CNT4B<=CNT4B+1;
18        END IF;
19      END IF;
20    END PROCESS;
21    PROCESS (CLK,CNT4B,START) IS
22    BEGIN
23      IF START = '0' THEN
24        IF CNT4B<8 THEN
25          CLKOUT <='1';
26          ARIEND <='0';
27        ELSE
28          CLKOUT <='0';
29          ARIEND <='1';
30        END IF;
31      ELSE
32        CLKOUT <='1';
33        ARIEND <='0';
34      END IF;
35    END PROCESS;
36 END ARCHITECTURE ART5;
37
```

图 5-1 控制器代码

ENTITY ARICTL 声明了一个实体（entity）的名称为 ARICTL。

PORT 子句定义了实体的端口：

CLK 是输入端口，表示时钟信号。

START 是输入端口，表示启动信号。

CLKOUT、RSTALL 和 ARIEND 是输出端口，分别表示计时器的时钟输出、复位信号和结束信号。

ARCHITECTURE ART5 OF ARICTL 定义了一个体（architecture），命名为 ART5，属于实体 ARICTL。

SIGNAL CNT4B : STD_LOGIC_VECTOR(3 DOWNTO 0); 声明了一个信号 CNT4B，用于计数。

RSTALL <= START; 将输入的启动信号直接赋给输出的复位信号。

第一个进程处理计数器的逻辑：

当启动信号 START 为 '1' 时，将计数器清零。

当时钟信号 CLK 上升沿发生时，计数器递增，直到计数到 8。

第二个进程处理时钟输出和结束信号的逻辑：

当启动信号 START 为 '0' 时，如果计数器未达到 8，时钟输出 CLKOUT 为 '1'，结束信号 ARIEND 为 '0'。

当计数器达到 8 时，时钟输出 CLKOUT 变为 '0'，结束信号 ARIEND 变为 '1'。

当启动信号 START 为 '1' 时，时钟输出 CLKOUT 为 '1'，结束信号 ARIEND 为 '0'。

这段代码实现了一个简单的计时控制器，通过控制时钟输出和结束信号来实现计时。

5.2 移位寄存器代码

以下是移位寄存器部分代码：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SREG8B IS
    PORT ( CLK,LOAD,EN : IN STD_LOGIC;
                DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                 QB : OUT STD_LOGIC  );
END SREG8B;
ARCHITECTURE ART2 OF SREG8B IS
    SIGNAL REG8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (CLK, LOAD)
    BEGIN
        IF  LOAD = '1' THEN  REG8 <= DIN;
        ELSIF CLK'EVENT AND CLK = '1' THEN
            IF EN = '1' THEN
            REG8(6 DOWNTO 0) <= REG8(7 DOWNTO 1);
            END IF;
        END IF;
    END PROCESS;
    QB <= REG8(0);
END ART2;

```

图 5-2 移位寄存器代码

PORT 子句定义了实体的端口：

CLK 是输入端口，表示时钟信号。

LOAD 是输入端口，表示加载信号，用于将输入数据加载到寄存器中。

EN 是输入端口，表示使能信号，用于控制寄存器中的数据移位。

DIN 是输入端口，是一个 8 位的标准逻辑向量，表示要加载到寄存器中的数据。

QB 是输出端口，表示寄存器的最低位。

ARCHITECTURE ART2 OF SREG8B 定义了一个体（architecture），命名为 ART2，属于实体 SREG8B。

SIGNAL REG8 : STD_LOGIC_VECTOR(7 DOWNTO 0); 声明了一个信号 REG8，用于保存 8 位的寄存器内容。

PROCESS (CLK, LOAD) 处理时钟和加载信号的逻辑：

当加载信号 LOAD 为 '1' 时，将输入数据 DIN 加载到寄存器中。

当时钟信号 CLK 上升沿发生时，如果使能信号 EN 为 '1'，则进行寄存器中的数据移位操作，即将寄存器中的数据向右移动一位。

QB <= REG8(0); 将寄存器中的最低位赋给输出端口 QB。

这段代码实现了一个可加载和可移位的 8 位寄存器。在时钟上升沿时，如果加载信号 LOAD 为 '1'，则将输入数据加载到寄存器中；如果使能信号 EN 为 '1'，则进行寄存器中的数据移位。最终，寄存器的最低位被输出到 QB 端口。

5.3 乘法器代码

以下是乘法器代码：

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   ENTITY ANDARITH IS
4       PORT ( ABIN : IN STD_LOGIC;
5              DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
6              DOUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
7   END ANDARITH;
8   ARCHITECTURE ART1 OF ANDARITH IS
9   BEGIN
10      PROCESS(ABIN, DIN)
11      BEGIN
12          FOR I IN 0 TO 7 LOOP
13              DOUT(I) <= DIN(I) AND ABIN;
14          END LOOP;
15      END PROCESS;
16  END ART1;
17
```

图 5-3 乘法器代码

ABIN 是输入端口，是一个标准逻辑信号，用于与输入数据进行逻辑与运算。

DIN 是输入端口，是一个 8 位的标准逻辑向量，表示待进行逻辑与运算的数据。

DOUT 是输出端口，是一个 8 位的标准逻辑向量，表示逻辑与运算的结果。

ARCHITECTURE ART1 OF ANDARITH 定义了一个体（architecture），命名为 ART1，属于实体 ANDARITH。

PROCESS (ABIN, DIN) 定义了一个进程，敏感于 ABIN 和 DIN 的变化。

FOR I IN 0 TO 7 LOOP 是一个循环，遍历 DIN 中的每一位。

DOUT(I) <= DIN(I) AND ABIN; 对于每一位，进行与门操作，将结果赋给输出端

口 DOUT 中的对应位。

这段代码描述了一个 AND 门阵列，它对输入的 8 位数据向量 DIN 中的每一位和输入信号 ABIN 进行逻辑与运算，将结果存储在输出向量 DOUT 中。每一位的与运算是并行进行的，这样可以在一次时钟周期内完成所有位的运算。这种结构在数字逻辑电路中常被用于实现各种算术逻辑单元（ALU）的一部分。

5.4 加法器代码

以下是加法器部分代码：

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4   ENTITY ADDER8B IS
5       PORT(  CIN:IN STD_LOGIC;
6              B, A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7            S1 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
8             COUT: OUT STD_LOGIC );
9   END ADDER8B;
10  ARCHITECTURE ART3 OF ADDER8B IS
11    SIGNAL SINT:STD_LOGIC_VECTOR(8 DOWNTO 0);
12  BEGIN
13            SINT <= '0'&A+B;
14            S1<=SINT(7 DOWNTO 0);
15            COUT<=SINT(8);
16  END ARCHITECTURE ART3;
17
```

图 5-4 加法器代码

CIN 是输入端口，表示加法器的进位输入。

A 和 B 是输入端口，分别表示要相加的两个 8 位二进制数。

S1 是输出端口，表示 8 位相加的和。

COUT 是输出端口，表示加法器的输出进位。

ARCHITECTURE ART3 OF ADDER8B 定义了一个体（architecture），命名为 ART3，属于实体 ADDER8B。

SIGNAL SINT : STD_LOGIC_VECTOR(8 DOWNTO 0); 声明了一个信号 SINT，用于保存相加的结果和可能的进位。

SINT <= '0' & A + B; 将输入 A 和 B 进行相加，并在最高位前加一个零，将结果保存在 SINT 中。

S1 <= SINT(7 DOWNTO 0); 将相加的结果的低 8 位赋给输出端口 S1。

COUT <= SINT(8); 将相加的结果的最高位赋给输出端口 COUT。

这段代码实现了一个 8 位全加器，可以将两个 8 位的二进制数相加，并考虑了输入的进位。结果被分别赋给输出端口 S1 和 COUT。

5.5 锁存器代码

以下是锁存器部分代码：

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ENTITY REG16B IS
4      PORT ( CLK,CLR,EN: IN STD_LOGIC;
5             D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
6             Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
7  END REG16B;
8  ARCHITECTURE ART4 OF REG16B IS
9      SIGNAL R16S : STD_LOGIC_VECTOR(15 DOWNTO 0);
10 BEGIN
11     PROCESS(CLK, CLR)
12     BEGIN
13      IF CLR = '1' THEN  R16S <= (OTHERS =>'0') ;
14      ELSIF CLK'EVENT AND CLK ='1' THEN
15        IF EN = '1' THEN
16         R16S(6 DOWNTO 0)  <= R16S(7 DOWNTO 1);
17         R16S(15 DOWNTO 7) <= D;
18        END IF;
19      END IF;
20     END PROCESS;
21     Q <= R16S;
22 END ART4;
23
```

图 5-4 锁存器代码

CLK 是输入端口，表示时钟信号。

CLR 是输入端口，表示清零信号。

EN 是输入端口，表示使能信号。

D 是输入端口，是一个 9 位的标准逻辑向量，表示要加载到寄存器中的数据。

Q 是输出端口，是一个 16 位的标准逻辑向量，表示寄存器的内容。

ARCHITECTURE ART4 OF REG16B 定义了一个体（architecture），命名为 ART4，属于实体 REG16B。

SIGNAL R16S : STD_LOGIC_VECTOR(15 DOWNTO 0); 声明了一个信号 R16S，用于保存 16 位寄存器的内容。

PROCESS (CLK, CLR) 处理时钟和清零信号的逻辑：

如果清零信号 CLR 为 '1'，则将寄存器内容清零。

如果时钟信号 CLK 上升沿发生时，如果使能信号 EN 为 '1'，则进行寄存器中的数据移位和加载操作。

Q <= R16S; 将寄存器的内容赋给输出端口 Q。

这段代码实现了一个带有时钟、清零和使能功能的 16 位寄存器。在每个时钟周期内，如果清零信号为 '1'，则将寄存器内容清零；否则，如果使能信号为 '1'，则进行寄存器中的数据移位和加载操作。最终，寄存器的内容被输出到 Q 端口。

第六章 仿真结果和分析

移位寄存器时序仿真：

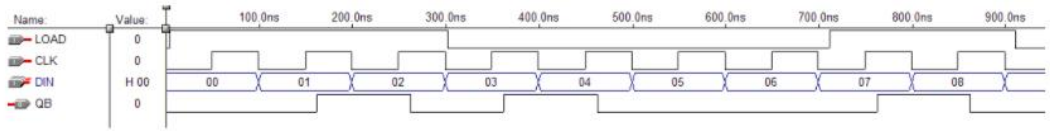

图 6-1 移位寄存器时序仿真

加法器时序仿真：

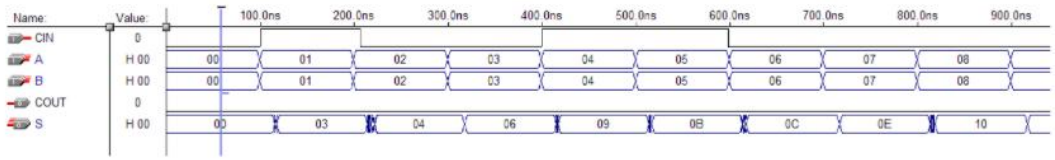

图 6-2 加法器时序仿真

乘法器时序仿真：

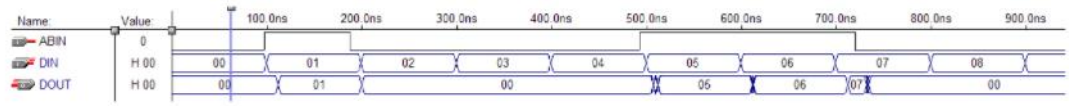

图 6-3 乘法器时序仿真

锁存器时序仿真：

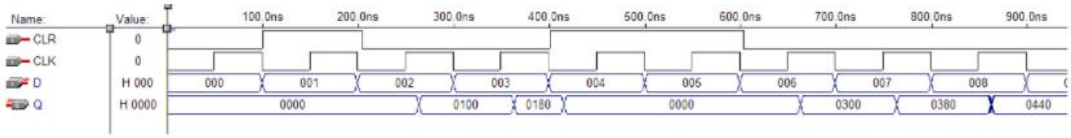

图 6-4 锁存器时序仿真

结果仿真：

图 6-5 结果仿真

分析：A 和 B 为输入八位二进制数,CLK 为时钟信号，这里给了 40ns，D1-D4 为输出，因为在结果考虑可能会接 7 段数码管，所以每个结果会输出 7 位。这边只看后四位，把后四位拼起来就是结果。STAR1 是开始信号，需要一个高电平。以下是结果转 7 段数码管对应代码：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY YIMA IS
   PORT( A    :  IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         LED7S : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END;
ARCHITECTURE art6 OF YIMA IS
BEGIN
   PROCESS(A)
   BEGIN
      CASE A IS
         WHEN "0000" => LED7S <= "0000000" ;
         WHEN "0001" => LED7S <= "0000001" ;
         WHEN "0010" => LED7S <= "0000010" ;
         WHEN "0011" => LED7S <= "0000011" ;
         WHEN "0100" => LED7S <= "0000100" ;
         WHEN "0101" => LED7S <= "0000101" ;
         WHEN "0110" => LED7S <= "0000110" ;
         WHEN "0111" => LED7S <= "0000111" ;
         WHEN "1000" => LED7S <= "0001000" ;
         WHEN "1001" => LED7S <= "0001001" ;
         WHEN "1010" => LED7S <= "0001010" ;
         WHEN "1011" => LED7S <= "0001011" ;
         WHEN "1100" => LED7S <= "0001100" ;
         WHEN "1101" => LED7S <= "0001101" ;
         WHEN "1110" => LED7S <= "0001110" ;
         WHEN "1111" => LED7S <= "0001111" ;
         WHEN OTHERS => NULL ;
      END CASE;
    END PROCESS;
END;
```

图 6-6　7 段数码管代码

第七章 结论与展望

7.1 结论

设计出一个乘法器是一项挑战性的工程，它要求深入理解数字逻辑、算术运算、信号处理和电路设计。乘法器的设计不仅是技术层面的挑战，同时也是工程上的挑战。通过本次课程设计，熟悉了 Quartus 软件的使用方法，可以更熟练地使用 Quartus II绘制原理图，通过利用自己所学的知识，设计出具有一定功能的器件，掌握了八位乘法器的设计原理：乘法通过逐项移位相加原理来实现，从被乘数的最低位开始，若为 1，则乘数左移与上一次和相加；若为 0，左移后以全零相加，直至被乘数的最高位。乘法器设计中，优化是一个重要的方面。优化可以包括电路的简化、提高运算速度、减少功耗或面积占用等方面，从而使乘法器更加高效。包括延迟、功耗、面积占用等方面的分析。这些指标对于乘法器在实际应用中的有效性至关重要。

在使用元件例化语句设计顶层文件的程序时，遇到了不少的问题，特别是各元件之间的连接，以及信号的定义，在认真修改之后，程序实现了功能。

7.2 展望

成功设计出乘法器是一个重要的里程碑，这仅仅是我数字电路和电子工程领域探索的开始。以下是我的展望方向，帮助我进一步提高和发展乘法器设计。

优化和改进： 乘法器的优化是一个持续的过程。通过探索更先进的算法、优化电路设计或采用新型技术，进一步提高乘法器的性能和效率。

扩展功能： 考虑增加乘法器的功能，如浮点数乘法、带符号数乘法、高精度乘法等。这些扩展将使乘法器更加多功能化和适用于更广泛的应用场景。

应用领域： 探索乘法器在不同领域的应用。数字信号处理、通信系统、图像处理等领域可能会对乘法器的性能和特性有不同的需求。

硬件实现： 考虑将乘法器集成到特定硬件平台或嵌入式系统中。这将需要考虑电路的实际实现、集成和优化，以适应特定的应用场景。

持续学习和研究： 数字电路领域发展迅速，持续学习和跟进最新的研究成果、技术和标准是非常重要的。阅读最新的论文和书籍，保持对领域的深入了解。

第八章 参考文献

以下是本次设计的参考文献：

[1]、EDA 技术与 FPGA 应用设计 张文爱 编著 电子工业出版社.2012

[2]、CPLD 技术及其应用 宋万杰 西安电子科技大学出版社.1999

[3]、可编程逻辑器件原理、开发与应用 赵曙光 西安电子科技大学出版社.2000

[4]、计算机组成原理实践教程,基于 EDA 平台 易小琳 朱文军 鲁鹏程 北京航空航天大学出版社.2006 年第一版

[5]、现代计算机组成原理 番松 潘明 科学技术出版社.2007 年第一版

[6]、基于 EDA 技术的计算机组成原理实验 方恺晴 湖南大学出版社.2006 年第一版