

第 9 章 DSP Builder 设计初步

利用 EDA 技术完成硬件设计的途径有多种，前面介绍的是利用 QuartusII 来完成的，最为典型的设计流程，包括设计项目编辑（如用 VHDL）、综合、仿真、适配、编程。但是对于一些特定的设计项目，这个流程就会显得很不方便，甚至无能为力。例如涉及算法类（如 DSP 模块）及模拟信号处理与产生方面的系统设计。

Altera 自 2002 年推出的 DSP Builder 则很好地解决了这些问题。

DSP Builder 可以帮助设计者完成基于 FPGA 的不同类型的应用系统设计。除了图形化的系统建模外，DSP Builder 还可以自动完成大部分的设计过程和仿真，直至把设计文件下载至 FPGA 开发板上。利用 Matlab 与 DSP Builder 进行模块设计也是 SOPC 技术的一个组成部分。本章以两个简单的电路模型设计为示例，详细介绍 Matlab、DSP Builder、QuartusII 三个工具软件联合开发的设计流程。

9.1 Matlab/DSP Builder 及其设计流程

DSP Builder 是一个系统级（或算法级）设计工具，它架构在多个软件工具之上，并把系统级（算法仿真建模）和 RTL 级（硬件实现）两个设计领域的设计工具连接起来，都放在了 Matlab/Simlink 图形设计平台上，而将 QuartusII 作为底层设计工具置于后台，最大程度地发挥了对种工具的优势。DSP Builder 依赖于 MathWorks 公司的数学分析工具 Matlab/Simlink，以 Simulink 的 Blockset 出现。可以在 Simulink 中进行图形化设计和仿真，同时又通过 SignalCompiler 把 Matlab/Simulink 的模型设计文件（.mdl）转成相应的硬件描述语言 VHDL 设计文件（.vhd），以及用于控制综合与编译的 tcl 脚本。对于综合以及此后的处理都由 QuartusII 来完成。

由于在 FPGA 上设计一个算法模型的复杂性，设计的性能（包括面积、速度、可靠性、设计周期）对于不同的应用目标将有不同的要求，涉及的软件工具也不仅仅是 Simulink 和 QuartusII，DSP Builder 针对不同情况提供了两套设计流程，即自动流程和手动流程。

图 9-1 是基于 Matlab、DSP Builder、QuartusII 等工具完成设计的流程框图。如图 9-1 所示，设计流程的第一步是在 Matlab/Simulink 中进行设计输入。即在 Matlab 的 Simulink 环境中建立一个 mdl 模型文件，用图形方式调用 DSP Builder 和其他 Simulink 库中的图形模块，构成系统级或算法级设计框图，或称 Simulink 设计模型。在第二步，是利用 Simulink 的图形化仿真、分析功能，分析此设计模型的正确性，完成模型仿真。在这两步中，与一般的 Matlab Simulink 建模过程几乎没有什么区别，所不同的是，设计模型库采用 DSPBuilder 的 Simulink 库而已，同样也涉及到其他 EDA 软件。

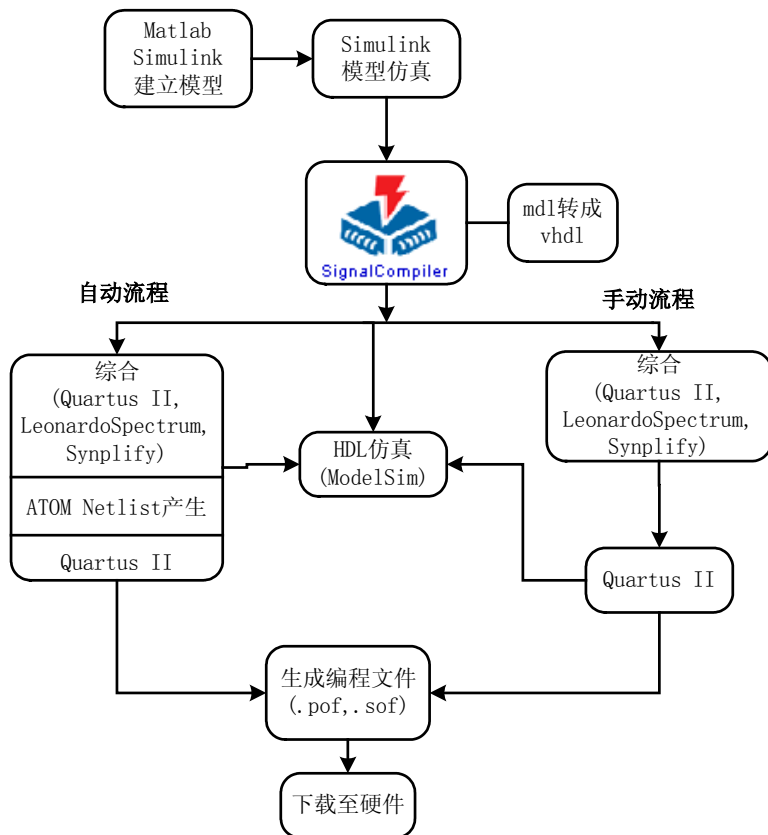


图 9-1 基于 Matlab、DSP Builder、QuartusII 等工具

第三步是 DSP Builder 设计实现的关键一步。由于 EDA 工具软件（诸如 QuartusII、ModelSim）不能直接处理 Matlab 的.mdl 文件，这就需要有一个转换过程。通过 SignalCompiler 把 Simulink 的模型文件（后缀为.mdl）转化成通用的硬件描述语言，VHDL 文件。转换获得的 HDL 文件是基于 RTL 级的，即可综合的 VHDL 描述。

此后的步骤是对以上顶层设计产生的 VHDL 的 RTL 代码和仿真文件进行综合、编译适配以及仿真。为了针对不同用户的设计目的和设计要求，DSP Builder 提供了两种不同的设计流程，主要可以分为自动流程和手动流程。

如果采用自动流程，几乎可以忽略硬件的具体实现过程，选择让 DSP Builder 自动调用 QuartusII 等 EDA 软件，完成综合（Synthesis）、网表（ATOM Netlist）生成和 QuartusII 适配，直至在 Matlab 中完成 FPGA 的配置下载过程。

但是，如果希望使用其它第 3 方的 VHDL 综合器和仿真器（除 Synplify、LeonardoSpectrum 和 QuartusII 综合器及 ModelSim 外），或是希望完成特定的适配设置，如逻辑锁定、时序驱动编译、ESB 特定功能应用等，可以选用手动流程设计。在此流程中，设计者可以灵活地指定综合、适配条件；手动地调用 VHDL 综合器进行综合，调用 QuartusII 进行适配，调用 ModelSim 或者 QuartusII 进行仿真，最后用 QuartusII 产生相应的编程文件用于 FPGA 的配置。

采用手动流程时，除了行为级仿真验证和设计输入外，其它过程与标准的基于 VHDL 的 EDA 设计流程是完全一致的。首先由基于 Matlab 的 DSP Builder 设计流程得到 VHDL 文件（由 Simulink 模型文件*.mdl 通过 SignalCompiler 转换而成），送入综合器进行综合。综合器可以是 Synplify Pro，也可以是 LeonardoSpectrum，或者采用 Altera 自己的 QuartusII 综合器。在综合时，可能需要对综合器进行配置或者提供综合的约束条件，由于这个过程操作可能比较繁琐，所以 DSP Builder 的 SignalCompiler 相应提供了一个接口，针对设计，自动产生一个 TCL 脚本与综合器 Synplify 或者 LeonardoSpectrum 相接。综合器在综合操作后，会产生一个网表文件，以供下一个流程使用。这里产生的网表文件称为 ATOM 网表文件，主要是 EDIF 网表文件（.edf 电子设计交换格式文件）或 VQM（.vqm Verilog Quartus Mapping File），它们是一种参数可设置的，并含有具体器件系列硬件特征（如逻辑宏单元 LCs、I/O 单元、乘积项、M4K、嵌入式系统块 ESB 等）的网表文件。QuartusII 可以利用这些 ATOM 网表文件针对选定的具体器件进行适配，包括布线、布局、结构优化等操作，最后产生时序仿真文件和 FPGA 目标器件的编程与配置文件。在这一步，设计者可以在 QuartusII 中完成对 Pin（引脚）的锁定，更改一些约束条件等。

如果用 DSP Builder 产生的设计模型只是庞大设计中的一个子模块，可以在设计中调用 DSP Builder 产生的 VHDL 文件，以构成完整的设计。同时，还可以使用 QuartusII 强大的 LogicLock 功能和 SignalTap 测试技术。在 DSP Builder 设计流程的最后一步，可以在 DSP Builder 中直接下载到 FPGA 用户开发板上，或者通过 QuartusII 完成硬件的下载、测试。

在图 9-1 的流程中，其中的 VHDL 仿真流程在设计中是不可或缺的。与 DSP Builder 可以配合使用的 HDL 仿真器是 ModelSim。DSP Builder 在生成 VHDL 代码时，可以同时生成用于测试 DSP 模块的 TestBench（测试平台）文件，DSP Builder 生成的 TestBench 文件采用 VHDL 语言，测试向量与该 DSP 模块在 Simulink 中的仿真激励相一致。通过 ModelSim 仿真生成的 TestBench 可以验证生成的 VHDL 代码与 Simulink 中设计模型的一致性。另外，DSP Builder 在产生 TestBench 的同时，还产生了针对 ModelSim 仿真的 Tcl 脚本来简化用户的操作，如包含了来自 Simulink 平台上进行仿真的激励信号信息等，从而掩盖 ModelSim 仿真时的复杂性。

在大部分情况下，QuartusII 对来自 DSP Builder 的设计模块适配后，需要再次验证适配后网表与 Simulink 中建立的 DSP 模型的一致性。这就需要再次使用 ModelSim 进行仿真，这时仿真采用 QuartusII 适配后带延时信息的网表文件（EDIF 格式、或者 VHDL、Verilog 格式），即为时序仿真。

两种设计流程归纳起来有如下几个步骤：

自动流程：1、MATLAB/Simulink 建模；2、系统仿真；3、DSP Builder 完成 VHDL 转换、综合、适配、下载。
4、嵌入式逻辑分析仪实时测试

手动流程：1、MATLAB/Simulink 建模；2、系统仿真；3、DSP Builder 完成 VHDL 转换、综合、适配；4、Modelsim 对 TestBench 功能仿真；5、QuartusII 直接完成适配（进

行优化设置); 6、QuartusII 完成时序仿真; 7、引脚锁定; 8、下载/配置与嵌入式逻辑分析仪等实时测试; 9、对配置器件编程, 设计完成。

考虑到实用的目的, 本章重点介绍手动设计流程。

9.2 正弦信号发生器设计

本节中, 以一个简单的可控正弦波发生模块的设计为例, 详细介绍 DSP Builder 基于手动流程的使用方法。图9-2所示是一个简单的正弦波发生器, 主要由4个部分构成: InCount 是阶梯信号发生模块, 产生一个按时钟线性递增的地址信号, 送往 SinLUT。SinLUT 是一个正弦函数值的查找表 (LUT: Look Up Table) 模块, 由递增的地址获得正弦波的量化值输出。由 SinLUT 输出的 8 位正弦波数据经过一个延时模块 Delay 后送往 Product 乘法模块, 与 SinCtrl 相乘。由于 SinCtrl 是 1 位 (bit) 输入, SinCtrl 通过 Product 就完成了对正弦波输出有无的控制。SinOut 是整个正弦波发生器模块的输出, 送往 D/A 即可获得正弦波的输出模拟信号。设计者在利用 DSP Builder 来进行相关设计时, 关键的设计过程大都在 Matlab 的图形仿真环境 Simulink 中进行。

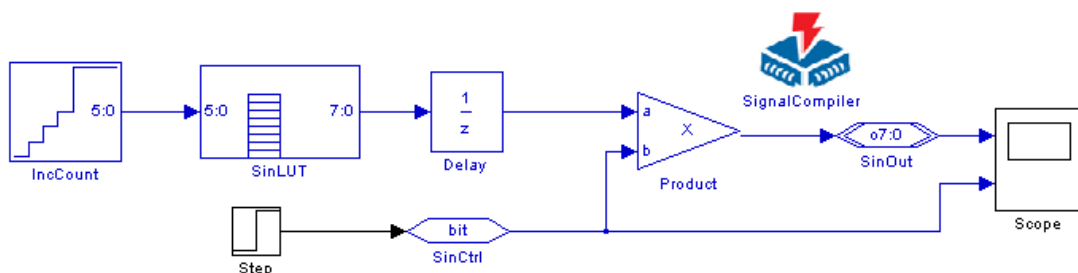


图 9-2 正弦波发生模块原理图

9.2.1 建立设计模型

首先需要建立一个新的设计模型, 步骤如下:

1、打开 Matlab 环境

Matlab 环境界面如图 9-3 所示。可以看到, Matlab 的主窗口界面被分割成三个窗口: 命令窗口 (Command Window)、工作区 (Workspace)、命令历史 (Command History)。在命令窗口中, 可以键入 Matlab 命令, 同时获得 Matlab 对命令的响应信息、出错警告提示等。

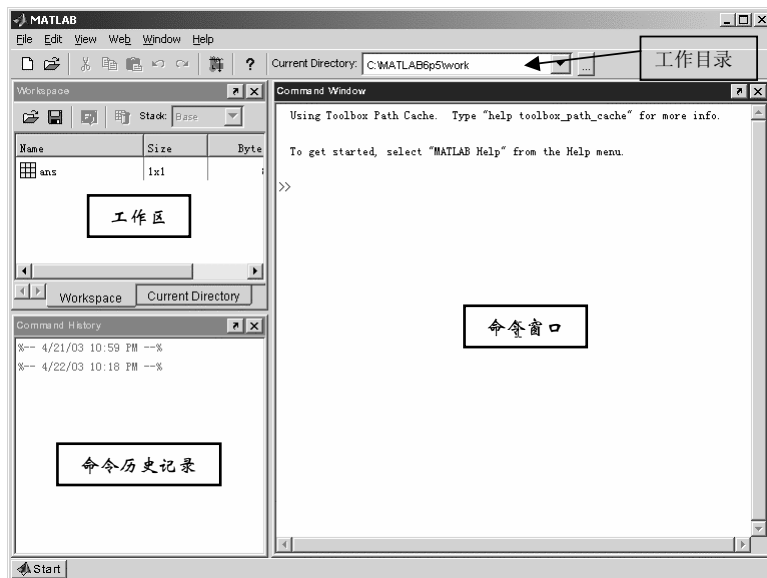


图 9-3 Matlab 界面

2、建立工作库

在建立一个新的设计模型前，最好先建立一个新的文件夹，作为 work（工作）目录，并把 Matlab 当前的 work 目录切换到新建的文件夹下。可以使用 Windows 在外部建立，也可以使用 Matlab 命令来直接完成这些操作，例如在 Matlab 主窗口中的命令窗口中键入：

```
cd e:/
mkdir /myprj/sinwave
cd /myprj/sinwave
```

其中 e:/myprj/sinwave 是新建的文件夹，是用作 Matlab 工作目录的。Mkdir 是一个建立新目录的 Matlab 命令，cd 是切换工作目录的 Matlab 命令。具体过程可以参见图 9-4（打开 simulink）。通过改变 Matlab 主界面中的“Current Directory”的制定，同样可以改变 Matlab 的当前工作目录。

3、了解 simulink 库管理器

当成功地把 Matlab 当前目录切换到新建的设计目录后（即键入 cd /myprj/sinwave），键入命令：pwd，之后可以在 Matlab 命令窗口键入“simulink”命令，以开启 Matlab 的图形化建模仿真环境 simulink。详见图 9-4（打开 simulink）。图 9-5 是 simulink 的库管理器 (Library Browser)。

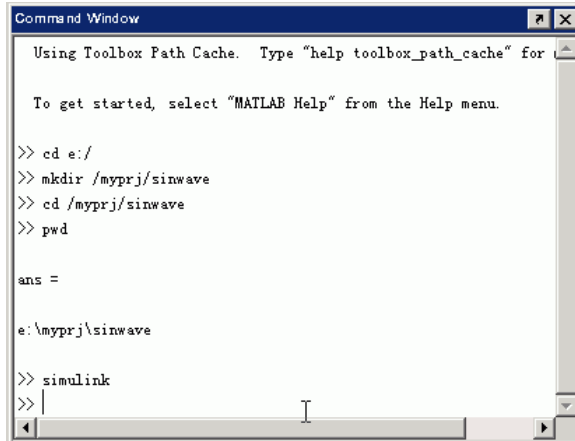


图 9-4 打开 simulink

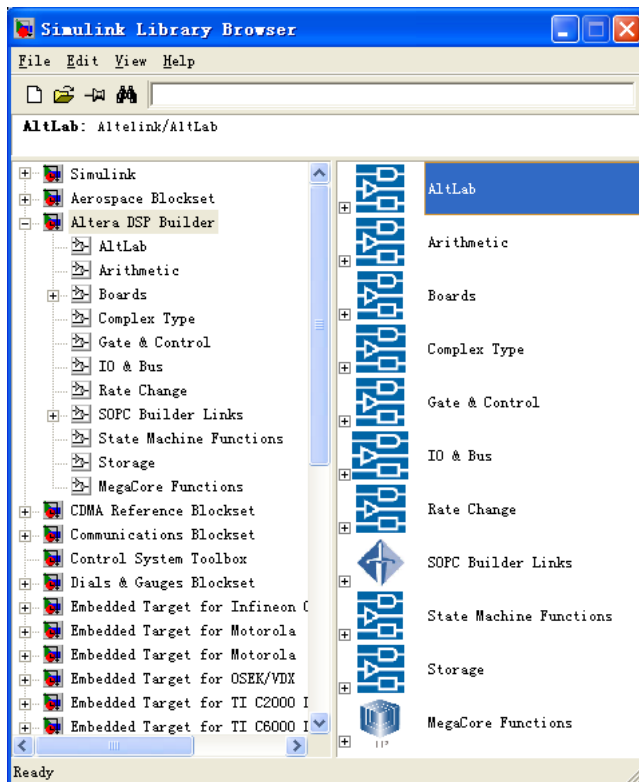


图 9-5 simulink 库管理器

在库管理器的左侧是 simulink library 列表，其中“Simulink”库是 simulink 的基本模型库。在库管理器的右侧是选中的 library 中的组件、子模块列表。当安装完 DSP Builder 后，在 Simulink 的库管理器中可以看到“Altera DSP Builder”字样出现在 library 列表中。在以下的 DSP Builder 应用中，主要是使用该库中的组件、子模型来完成各项设计，再使用

“Simulink”库来完成模型的仿真实验。

注意，只有来自 Altera DSP Builder 元件库中的元件模块构成的电路系统或算法模型能被 DSP Builder 转化为 VHDL 程序

4. simulink 的模型文件

在打开 simulink 库管理器后，需要新建一个 simulink 的模型文件（后缀为 mdl），在 simulink 的库管理器中选择“File”菜单（图 9-6），在出现的菜单项中选择“New”，在弹出的子菜单项中选择新建模型“model”。图 9-6 右边显示的就是新模型窗口。

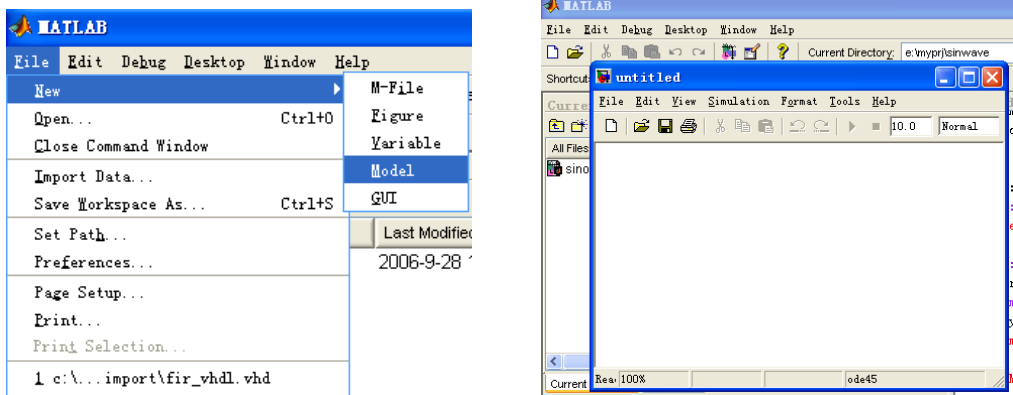


图 9-6 建立新模型

5、放置 SignalCompilder

点击 simulink 库管理器左侧的库内树形列表中的 Altera DSP Builder 条，使之展开 DSP Builder 库，这时会出现一长串树形列表，对 DSP builder 库的子模块（Block）进行了分组，再次点击其中的 AltLab 项，展开 AltLab，选中库管理器右侧的 SignalCompilder 组件，按住鼠标右键拖动 SignalCompilder 到新模型窗口中（图 9-7）。

注意：当在 simulink 库管理器中选中模块（Block）后，在库管理器上方的提示栏中会显示对应模块的说明，有简单的功能介绍。

6、放置 Increment Decrement

参照图 9-2，将 Increment Decrement 模块放置到新模型中。Increment Decrement 模块（图 9-8）是 DSP Builder 库中 Arithmetic（算术）模块。选中 Altera DSP Builder 库中的 Arithmetic 条，则在库管理器的右侧，可以看到 Increment Decrement 模块。按照放置 SignalCompilder 的方法，把 Increment Decrement 模块拖到新建模型窗口中。

7. 设置 IncCount

用鼠标点击在新建模型窗口中的 Increment Decrement 模块下面的文字“Increment Decrement”，就可以修改文字内容，也就是可以修改模块名字。在这里不妨将其修改为“IncCount”（图 9-8）。然后按照图 9-2 中描述的含义，把 IncCount 模块做成一个线性递增的地址发生器，这就需要为 IncCount 模块的参数进行相应的设置：双击新建模型中的

IncCount 模块，打开 IncCount 的模块参数设置对话框：“Block Parameters: IncCount”，如图 9-9 所示。在参数设置对话框的上半部分是该模块的功能描述和使用说明；对话框的下半部分是参数设置部分，对于 Increment Decrement 模块共有下面几种参数可以设置：

- 总线类型 (Bus Type)；
- 输出位宽 (Number of bits)；
- 增减方向 (Direction)；
- 开始值 (Starting Value)；
- 是否使用控制输入 (Use Control Inputs)；
- 时钟相位选择 (Clock Phase Selection)；

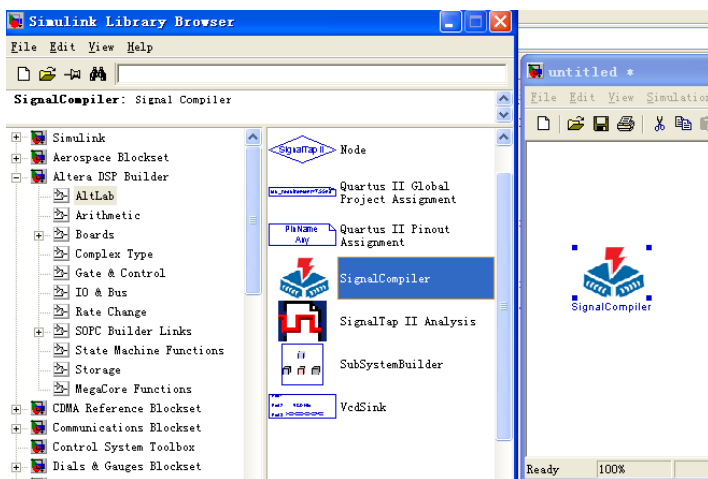


图 9-7 放置 SignalCompiler

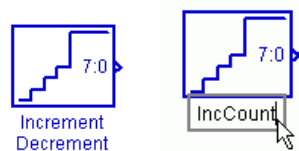


图 9-8 递增递减模块改名为 IncCount

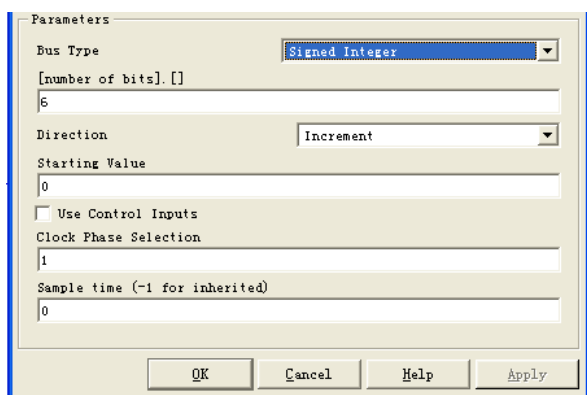


图 9-9 设置递增递减模块



图 9-10 LUT 模块

对于总线类型 (Bus Type)，在下拉列表框中共有三种选择：

- 1、有符号整数 (Signed Integer)；如 9 等于二进制的 1001，等于-7。
- 2、有符号小数 (Signed Fractional)；
- 3、无符号整数 (Unsigned Integer)。例如 9 等于二进制的 1001，等于 9。

在这里选择“signed Integer”，即有符号整数。对于输出位宽，由于在后面接着的正弦查找表（Sin LUT）的地址为 6 位，所以输出位宽设为 6。

IncCount 是一个按时钟增 1 的计数器，Direction 设为 Increment（增量方式）。

Use Control Inputs 项可以不选。因为如果选中此项，此模块会出现两个输入端，分别是复位端和时钟使能端。通常在 simulink 图中的元件的复位和时钟使能端以及都是分别默认接于低电平和高电平使能的，而时钟端是按全局时钟方式相连的。

Clock Phase Selection 可设置为 1（二进制），其他设置采用 Increment Decrement 模块的默认设置。设置完的对话框如图 9-10。然后点击“OK”按钮确认。

注意，若对 DSP Builder 库中模块设置参数值不了解，可以在相应模块的参数设置对话框中点击“Help”按钮（或按 F1 键），呼出相应帮助信息，以便了解详细的模块参数说明。

8、放置正弦查找表（SinLUT）

在 Altera DSP Builder 库的 Gate 库中找到查找表模块：LUT（图 9-10）。把 LUT 拖到新建模型窗口，按照 IncCount 的做法把新调入的 LUT 模块的名字修改成“SinLUT”。

双击 SinLUT 模块，打开模块参数设置对话框：“Block Parameters SinLUT”，如图 9-11 所示。把输出位宽（Output[number of bits]）改为 8，查找表地址线位宽（LUT Address Width）设为 6。总线数据类型 Bus Type 选择为有符号整数 Signed Integer；在“MATLAB Array”编辑框中输入计算查找表内容的计算式。在此可以直接使用 sin（正弦）函数，在这里 sin 函数的调用格式为：

$\sin(\text{[起始值:步进值:结束值]})$

SinLUT 是一个输入地址为 6 位、输出值位宽为 8 的正弦查找表模块，且输入地址总线为有符号数，可以设置起始值为 0、结束值为 2π 、步进值为 $\frac{2\pi}{2^6}$ 。计算式可写成：

$$127*\sin[0:2*\pi/2^6]:2*\pi]) \quad 9-1$$

其中 pi 即为常数 π 。上式的数值变化范围是 -127 - +127，总值是 256，恰好是 8 位二进制数最大值。但应注意，如果改变地址线宽，如 8，以上的 2 的 6 次方要改成 2 的 8 次方，即：

$$127*\sin[0:2*\pi/2^8]:2*\pi]) \quad 9-2$$

如果将 SinLUT 模块的总线数据类型设置为无符号整数 Unsigned Integer，且输出位宽（Output[number of bits]）改为 10，若想得到完整满度的波形输出，式 5-1 应该为下式：

$$511*\sin[0:2*\pi/2^6]:2*\pi]) + 512 \quad 9-3$$

在“Use LPM”处可以选择打勾，表示允许 QuartusII 利用目标器件中的嵌入式 RAM（在 EAB、ESB 或 M4K 模块中）来构成 SinLUT，即将生成的正弦波数据放在嵌入式 RAM 构成的 ROM 中，这样可以节省大量逻辑资源。否则 SinLUT 只能用芯片中的 LCs 来构成。设置好的 SinLUT 参数如图 9-11 所示。

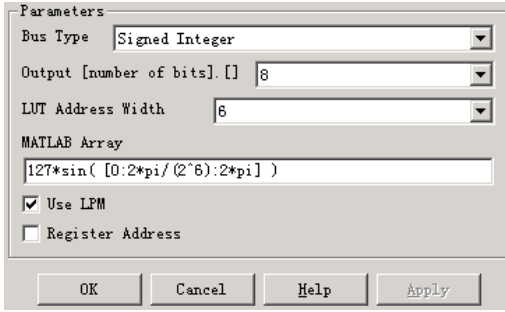


图 9-11 设置 SinLUT

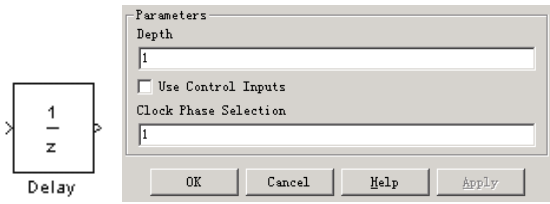


图 9-12 Delay 模块及其参数设置窗

9、放置 Delay 模块

在 simulink 库管理器的 Altera DSP Builder 库中，选中 Storage 库下的 Delay 模块，放置到新建模型窗口。Delay 模块（图 9-12）是一个延时环节。在这里可以不修改其默认参数设置。其具体的默认参数设置见图 9-12。

在 Delay 模块的参数设置对话框中，参数“Depth”是描述信号延时深度的参数。当 Depth 为 1，信号传输函数为 $1/z$ ，表现为通过 Delay 模块的信号延时 1 个时钟周期；当 Depth

为整数 n ，其传输函数为 $\frac{1}{z^n}$ ，表现为通过 Delay 模块的信号将延时 n 个时钟周期。Delay

模块在硬件上可以采用寄存器（锁存器）来实现，这也就是为什么把 Delay 模块放在 Storage（存储单元）库中的原因。

Clock Phase Selection 参数主要是控制采样的。当设置为 1 时表示，每一主频脉冲后，数据都能通过，如果设置为 01，则每隔一个脉冲通过一个数据；若设 0011，表示每隔两个脉冲通过两个脉冲；0100 表示 Delay 在每隔第 2 个时钟时被使能通过，而在第 1、3、4 个时钟时被禁止通过，等等。

到现在为止，已经在新建模型中放置了 4 个模块，先按照图 9-2 把他们连接起来。把鼠标的指针移动到上述几个模块的输入输出端口上，鼠标指针就会变成十字形，这时按住鼠标左键，拖动鼠标就可以连线了。

10、放置端口 SinCtrl

在 simulink 库管理器的 Altera DSP Builder 库中，选中 IO & Bus 库，找到 Input 模块，放置在新建模型窗口中。修改 AltBus 模块的名字为 SinCtrl。SinCtrl 是一个 1 位输入端口。双击 SinCtrl 模块，打开模块参数设置对话框，设置 SinCtrl 的 Bus Type 参数为“Single Bit”，Node Type 参数为“Input Port”，如图 9-13 所示。此端口模块将在产生的 VHDL 文件中变成端口模式为“IN”，数据类型为 STD_LOGIC 的端口信号。

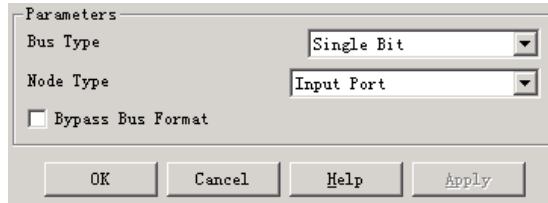


图 9-13 设置 SinCtrl

11、放置 Product 模块

在 simulink 库管理器的 Altera DSP Builder 库中，选中 Arithmetic 库，找到 Product 模块（图 9-14），用鼠标拖放到新建模型窗口中。在这里 Product 有两个输入，一个是经过一个 Delay 的 SinLUT 查表输出，另一个是外部一位端口 SinCtrl。按算法逻辑来看，实现了 SinCtrl 对 SinLUT 查找表输出的控制。双击 Product 模块，打开模块参数设置对话框，设置 Product 的参数如图 9-14。其中 Pipeline（流水线）参数指定该乘法器模块使用几级流水线，即乘积延时几个时钟周期后出现。选中“Use LPM”，表示允许采用 LPM 模块。“Use Dedicated Circuitry”选项用于对 FPGA 中的专用模块的选择，如 Stratix, CycloneII 等器件中的专用 DSP 模块。

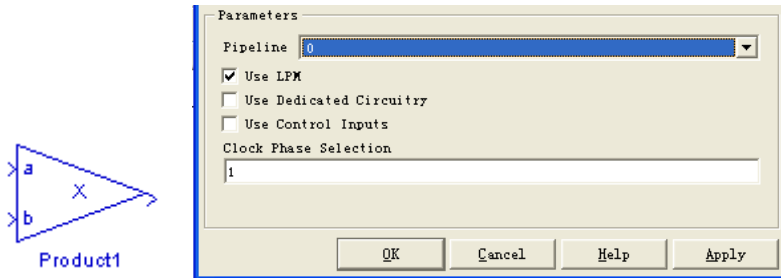


图 9-14 设置乘法单元

12、放置输出端口 SinOut

在 simulink 库管理器的 Altera DSP Builder 库中，选中 IO & Bus 库，找到 Output 模块（图 9-15），放置在新建模型窗口中。修改模块的名字为 SinOut。

SinOut 是一个 8 位输出端口，接向 FPGA 的输出端口，与外面的 8 位 D/A 转换器相接，通过 D/A 把 8 位数据转换成 1 路模拟信号。双击 SinOut 模块，打开模块参数设置对话框，设置 SinOut 的 Bus Type 参数为“Signed Integer”，修改“number of bits”参数为 8，如图 9-15 所示。该模块在 VHDL 文件中将变成 OUT 端口模式的标准位矢量：STD_LOGIC_VECTOR(7 DOWNTO 0)。

如果选择 AltBus 模块，则其中的“Saturate”项如果选择“On”，则当输出大于有待表达的最大的正值或负值，则此输出即扩位到此最大的正值或负值。若此选项取为“Off”，则最高位 MSB 被截去。此选项对输入端口或常数节点类型是无效的。

“Round”项若选择“On”，则输出略去了所有高位的 0 位；若选为“Off”，则最低位 LSB 被截去。此选项不适用于输入和常数类型。

“Bypass Bus Format”项为“On”时，表示在 Simulink 中使用浮点数进行仿真。

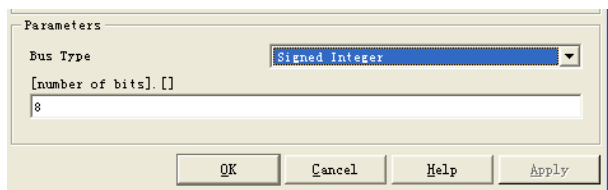


图 9-15 设置 SinOut

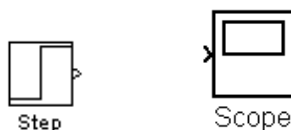


图 9-16 Step 模块 图 9-17 Scope 模型

13. 设计文件存盘

放置完 SinOut 模块后，按照图 9-2，把新建模型窗口中的 DSP Builder 模块连接起来。这样就完成了一个正弦波发生器的 DSP Builder 模型设计。在进行仿真验证和 SignalCompiler 编译之前，先对设计进行存盘操作：点击新建模型窗口的“File”菜单，在下拉菜单中选择“Save”项，取名并保存。在这个例子中，对新建模型取名为 Sinout，模型文件为 sinout.mdl。在保存完毕后，新建模型窗口的标题栏就会显示模型名称。

注意，对模型文件取名，尽量用英文字母打头，不使用空格，不用中文，文件名不要过长。且只有对文件存盘后，才能使用 SignalCompiler 进行编译，把 mdl 文件转换为 VHDL 文件。不过现在模型的正确性还是未知的，需要进行仿真验证。

9.2.2 Simulink 模型仿真

Matlab 的 Simulink 环境具有强大的图形化仿真验证功能。用 DSP Builder 模块设计好一个新的模型后，可以直接在 simulink 中进行算法级、系统级仿真验证。对一个模型进行仿真，需要施加合适的激励、一定的仿真步进和仿真周期，添加合适的观察点和观察方式。

1、加入仿真步进模块

首先加入一个 Step 模块，以模拟 SinCtrl 的按键使能操作。在 Simulink 库管理器中，展开 simulink 库，选中 Sources 库，把 Sources 库中的 Step 模块拖放到 sinout 模型窗口中（如图 9-16）。参照图 9-2，把 step 模块与 SinCtrl 输入端口相接。

注意，凡是来自 Altera DSP Builder 库以外的模块，SignalCompiler 都不能将其变成硬件电路，即不会影响产生的 VHDL 程序，但在启动 Simulink 仿真后能影响后面产生的仿真激励文件。Step 模块的情况正是如此。

2、添加波形观察模块

在 simulink 的库管理器中，展开 simulink 库，选中其中的 Sinks 库，把 Scope（示波器，图 9-17）模块拖放到 sinout 模型窗口中。双击该模块，打开的是一个 Scope 窗口（图 9-18）。图中所示只有一个信号的波形观察窗口，而如若希望可以多观察几路信号，自然可以通过调用多个 Scope 模块的方法来实现，这里介绍通过修改 scope 参数来增加同一 scope 中观察窗。

3、Scope 参数设置

用鼠标点击 Scope 模块窗口上侧工具栏的第二个工具按钮：“Parameters” 参数设置。打开 Scope 参数设置对话框，见图 9-19。在 Scope 参数设置对话框中共有两个选项页：“General”（通用）和“Data History”（数据历史）。在“General”选项页中，改变“Number of axes”参数为 2。在点击“OK”按钮确认后，可以看到 Scope 窗口增加了两个波形观察窗。每个观察窗都可以分别观察信号波形，而且相对独立。然后将 SinCtrl 的信号接向 Scope 的另一端（图 9-21），以作信号比较。需要注意的是，此路信号在生成的 VHDL 文件中不会有相应语句，因为没有接上输出端口模块：AltBus 或 Output 模块。

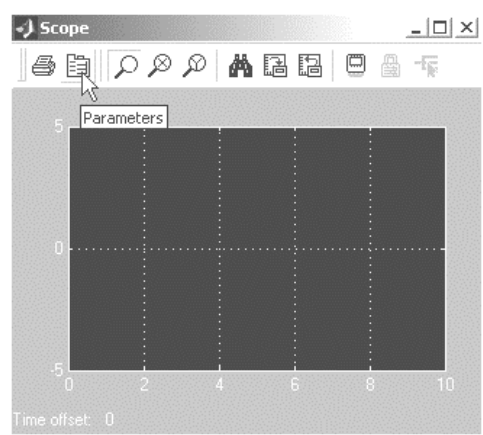


图 9-18 Scope 初始显示

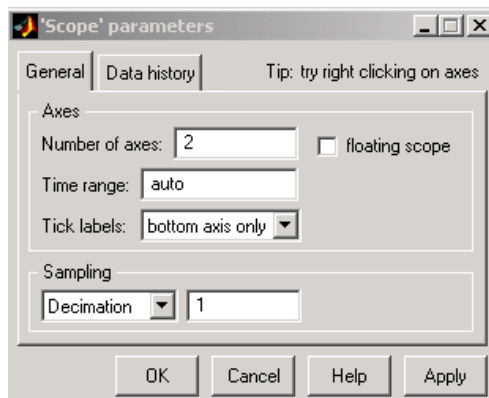


图 9-19 设置 Scope 参数

4、设置仿真激励

按图 9-2 连接 sinout 模型的全图，准备开始仿真。在仿真前还需要设置一下与仿真相关的参数。先设置模型的仿真激励。在 sinout 模型图中只有一个输入端口 SinCtrl，需要设置与此相连的 Step 模块：双击放置在 sinout 模型窗口中的 Step 模块，设置对输入端口 SinCtrl 施加的激励。在打开的 Step 模块参数设置对话框中，可以看到下列参数（详见图 9-20）：

- 步进间隔（step time）；
- 初始值（Initial value）；
- 终值（Final value）；
- 采样时间（Sample time）

在此设置“step time”为 50；“Initial value”为 50，即初始时，不输出正弦波；“Final Value”为 1；“Sample time”为 1。把最后两项选择：“Interpret vector parameters as 1-D”和“Enable zerocrossing detection”都设为打勾。

在 sinout 模型编辑窗中(图 9-21)，点击“Simulation”菜单，在下拉菜单中选择“Simulation parameters”菜单项，如图 9-22 所示。随后，将弹出 sinout 模型的仿真参数设置对话框：

“Simulation Parameters : sinout”。

仿真参数设置对话框中共有 5 个选项页：Solver、Workspace I/O、Diagnostics、Advanced、Real-Time Workshop。其中“Solver”选项页中完成仿真时，基本的时间设置、步进间隔和

方式、输出选项。在 `sinout` 模型中，可设置“Start time”为 0.0，“Stop time”为 500。其他设置按照默认。选项页的设置也按照默认设置。然后，点击“OK”按钮确认。

为了能更好的在波形观察窗中区分不同信号，可以在 `sinout` 模型中对连接线进行命名：双击对应的连接线，就会出现一个可以输入文本的小框，在框中输入信号的名称。

6、启动仿真

`sinout` 模型编辑窗中，选择“Simulation”菜单，选“Start”项，开始仿真（图 9-23）。等待仿真结束，双击 `Scope` 模块，打开 `scope` 观察窗。图 9-24 显示了仿真结果，`SinOut` 信号是 `sinout` 模型的输出（`scope` 观察窗中模拟了 D/A 的输出波形），`SinCtrl` 信号是 `sinout` 模型的输入，可以看出 `SinOut` 受到了 `SinCtrl` 的控制。当 `SinCtrl` 为 1 时，`SinOut` 波形是正弦波；当 `SinCtrl` 为 0 时，输出为 0。在 `Scope` 观察窗中，可以使用工具栏中的按钮来放大缩小波形，也可以在波形上单击右键使用“Autoscale”，使波形自动适配波形观察窗，用鼠标左键，可以放大波形。图 9-25 是改成无符号输出的波形。



图 9-20 设置 Step

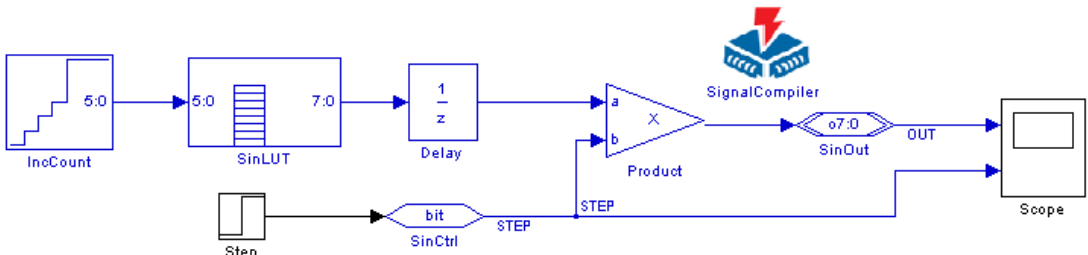


图 9-21 `sinout` 全图

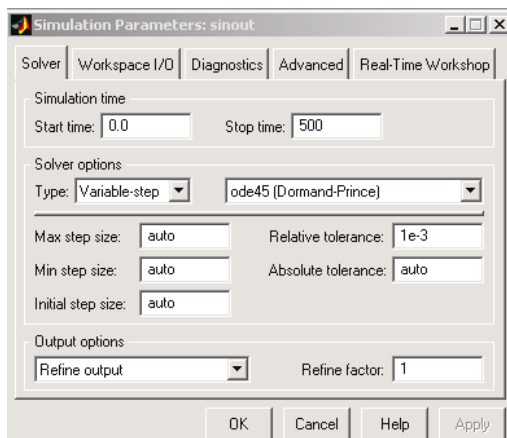


图 9-22 simulink 仿真设置

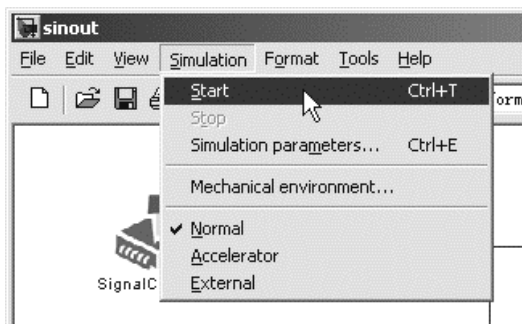


图 9-23 simulink 仿真 Start

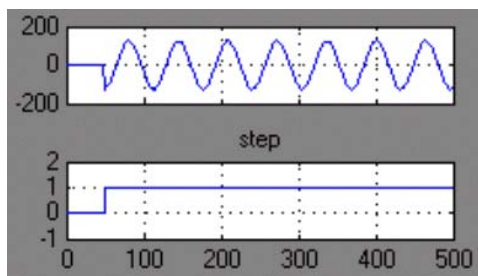


图 9-24 有符号输出波形（系统级仿真）

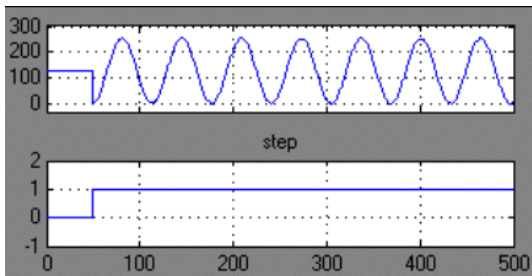


图 9-25 无符号输出波形（系统级仿真）

7、设计成无符号数据输出

由图 9-24 可以看出，输出的正弦波是有符号数据，它在正负 127 间变化，但一般的 D/A 器件的输入数据都是无符号的正数。因此，为了能在硬件系统上 D/A 的输出也能观察到此波形，必须对此输出作一些改进，以便输出无符号数。

最简单的方法是将图 9-24 中的波形向上平移 127，即对输出的数据加上 127 即可。

即对图 9-21 中的 SinLUT 的波形数据公式可以改为： $127 * \sin[0:2 * \pi / 2^6:2 * \pi] + 128$ 。此外，图中相关模块的有符号设置全部改为无符号设置。

也可以在原来电路的基础上增加一些模块。图 9-26 就是改进后的电路，其功能即在原输出的有符号数据上加上了 128。原理是将乘法器输出的 8 位有符号数的最高位取反并以无符号数输出。图 9-25 即是产生的仿真波形，可以看出整个波形在 0 以上。以下将对图中出现的新模块的用法作一说明。

8、各模块功能说明

图 9-26 中的新添模块都来自 Altera DSP Builder 的 IO&Bus 库。

SinOut1 与 SinOut 类似，都是总线模块 AltBus，只是设定为有符号内部节点模块（图 9-27），此模块在 VHDL 文件中将变成内部信号 Signal 定义。

ExtractBit 模块的功能是将总线中一指定位提取出来。在这里它将 8 位输出总线的最高

位提取出来，其参数设置如图 9-28 所示。

BusConversion 是总线变换模块，其功能是将总线中指定数位提取出来。在这里它将 8 位输出总线的低 7 位提取出来，即将第 0 位至第 6 位提取输出，其参数设置如图 9-29 所示。

BusConcatenation 是总线合并模块，其功能是将两个总线按要求合并成一个总线，其输出位宽数等于输入的两个总线位数之和，参数设置如图 9-30 所示。

SinOut 模块的设置基本不变，只是将有符号整数改成无符号整数输出（图 9-31）。

Not 模块是反相器模块，设置如图 9-32 所示。

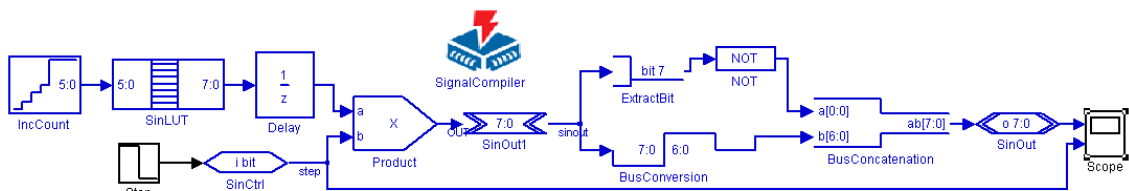


图 9-26 无符号整数 Signed Integer 输出电路

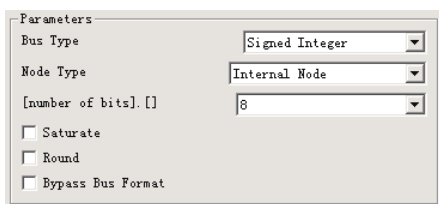


图 9-27 SinOut1 模块设置

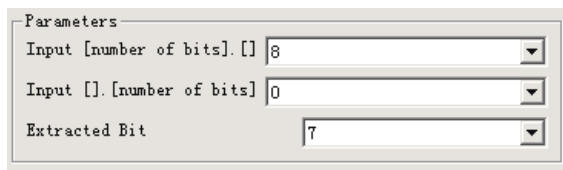


图 9-28 ExtractBit 模块设置

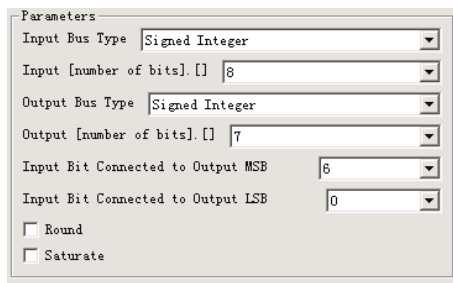


图 9-29 BusConversion 模块设置

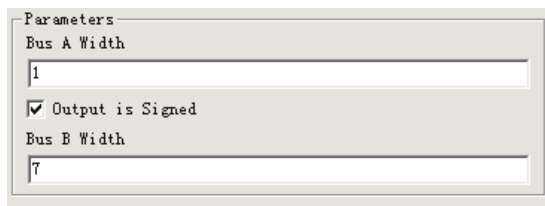


图 9-30 BusConcatenation 模块设置

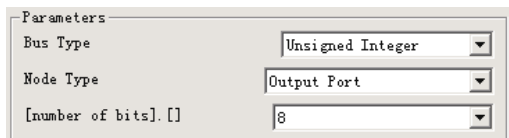


图 9-31 SinOut1 模块设置

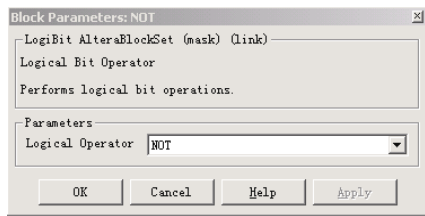


图 9-32 NOT 模块设置

9.2.3 SignalCompiler 使用方法

在 Simulink 中完成仿真验证后，就需要把设计转到硬件上加以实现。这是整个 DSP Builder 设计流程中最关键的一步，据此可以获得针对 FPGA 的 VHDL RTL 代码。

1、分析当前的模型

双击 sinout 模型中的 SignalCompiler 模块，将出现如图 9-33 所示的对话框，点击“Analyze”（分析）按钮后，SignalCompiler 就会对 sinout 模型进行分析，检查模型有无错误，并在 Matlab 主窗口弹出对话框，并给出相关信息。若有错误存在，SignalCompiler 就会停止分析过程，并把错误信息显示在 Matlab 主窗口的“Command Window”命令窗口中；反之，在分析过程结束后，打开 SignalCompiler 窗口（图 9-34）。如果有警告（Warning）存在，同错误一样把警告信息显示在命令窗口。

Simulink 具有较为强大的错误定位能力，对许多错误可以在 simulink 模型中直接定位，用不同的颜色提示有错误的 Simulink 模块（Block）。当 SignalCompiler 分析当前 DSP 模型有错误时，必须去除错误才能继续 DSP Builder 流程。

2、设置 Signal Compiler

在图 9-34 中显示了 Signal Compiler 窗口，大致上可以分为三个功能部分：

- 左侧的项目设置选项“Project Setting Options”
- 右侧的硬件编译流程“Hardware Compilation”
- 下方的信息框“Messages”

SignalCompiler 的设置都集中在项目设置选项部分。在“Device”下拉选择框中选择需要的器件系列，默认为 Stratix 系列器件，对此可以修改。在此选为 Cyclone 系列。

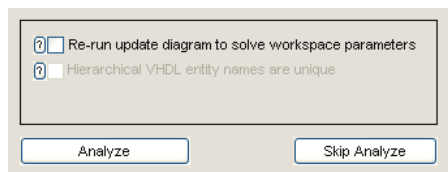


图 9-33 双击 SignalCompiler

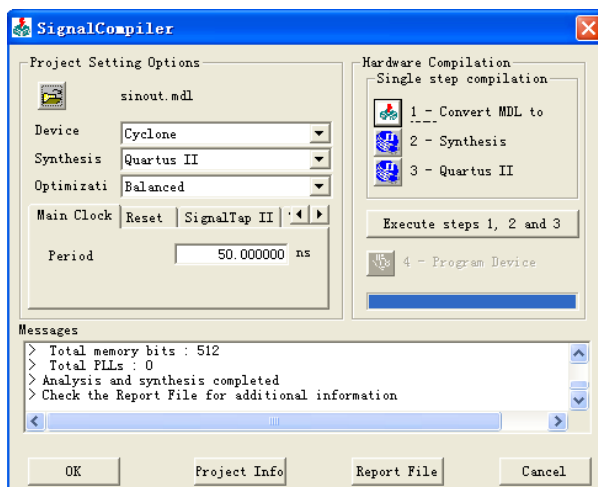


图 9-34 打开 SignalCompiler 窗口

注意，在“Device”中只能选择器件系列，不能制定具体的器件型号，这需由 QuartusII

自动决定使用该器件系列中的某一个具体型号的器件，或在手动流程中由用户指定。

在“Device”下拉选择框中，有一个“DSP Board”的选择，这是一个特殊的选项，是针对具体的 DSP 开发板的。

“Synthesis”（综合）下拉选择框可以选择综合器，共有三个选择：


- “LeonardoSpectrum”：Mentor 的 LeonardoSpectrum 综合器
- “Synpilfy”：Synpilcity 的 Synpilfy Pro 或 Synpilfy 综合器
- “Quartus II”：Altera 的 FPGA/CPLD 开发集成环境，内含综合功能

在这个例子中，可选择 Quartus II。

“Optimization”（优化）下拉选择框，指明在综合、适配过程中的优化条件，是对面积（Area）优化还是速度（Speed）优化的选择，即资源占用优先还是性能优先。

项目设置选项（“Project Setting Options”）部分的下侧是一些选项页，其中

- “Main Clock”中指定系统主时钟的周期；
- “Reset”是系统复位信号的设置
- “SignalTapII”是嵌入式逻辑分析仪 SignalTap 设置
- “TestBench”是仿真测试文件生成的选择
- “SOPC info”是 SOPC 相关设置

在这里，不妨指定 Main Clock 的周期为 50ns，点击  的右边箭头找到“TechBench”选项页，选择“Generate Stimuli for VHDL Testbench”生成含激励的仿真测试文件，用于 VHDL 仿真。

3、把模型文件 MDL 转换成 VHDL

当设置好“Device”和“Synthesis”后，右侧的硬件编译“Hardware Compilation”部分就会列出一个操作流程，见图 9-34。分别为：

1. “Convert MDL to VHDL”：转换 MDL 文件为 VHDL 文件；
2. “Synthesis”：对转换好的 VHDL 文件进行综合；
3. “QuartusII” Quartus 编译适配，生成编程文件

图 9-34 中的第 4 步是隐去的，这一步实现在 simulink 中，完成 FPGA 的硬件配置下载。这需要 DSP 开发板的支持，当“Device”选为“DSP Board”时，该一步才显示出来。

按步骤，先点击步骤 1 的图标，完成 simulink 文件 (*.mdl) 到 VHDL 文件的转换。转换完成后，在“Messages”信息提示框中，会显示“Generated top level “sinout.vhd” files”，即顶层文件 sinout.vhd 完成转换。若有错误，在“Messages”信息提示框中会显示简短的出错提示（出错的原因多数是授权文件的安装或文件本身有问题）。sinout 模型生成的 VHDL 文件“sinout.vhd”可以在文件夹 e:/myprj/sinwave 中找到。

4、综合（Synthesis）

点击步骤 2 的图标，完成综合过程。在这个例子中是调用 QuartusII 来完成综合过程的，在综合后生成 ATOM Netlist（原子网表），供适配器使用。“Messages”给出了相关信息：此设计项目使用了 23 个逻辑宏单元、11 个引脚、512 个 RAM 位（恰好等于 SinLUT 模块

中数据的容量)。

5、QuartusII 适配

点击步骤 3 的图标，调用 QuartusII 完成编译适配过程，生成编程文件：pof 文件和 sof 文件。编程文件可以直接用于 FPGA 的编程配置。不过，按现在这个流程，采用的是自动 (Auto) 器件，管脚 pin 是自动指定的。

另外，点击图 9-34 的“Report File”按钮，将获得详细的报告文件。sinout 模型对应的报告文件为 sinout_DspBuilder_Report.html。在 DSP Builder 报告文件中，详尽地描述了模型相关的设置情况、编译结果、生成文件列表说明、注意点等等，而且对使用在 sinout 模型中的模块分别进行了说明 (图 9-35)。

注意，对于手动设计流程，适配步骤已没有必要，通常只须完成前面两个步骤即可。

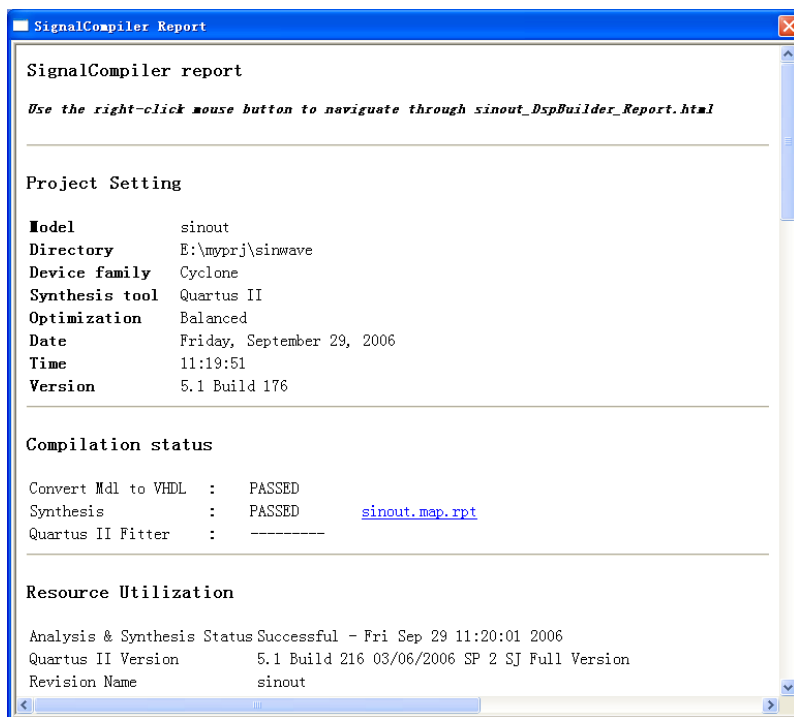


图 9-35 sinout 工程处理信息

9.2.4 使用 ModelSim 进行 RTL 级仿真

在 simulink 中进行的仿真是属于系统验证性质的，是对 mdl 文件进行的仿真，并没有对生成的 VHDL 代码进行过仿真。事实上，生成 VHDL 描述的是 RTL 级的，是针对具体的硬件结构的，而在 Matlab 的 simulink 中的模型仿真是算法级 (系统级) 的。两者之间有可能存在软件理解上差异。转换后的 VHDL 代码实现可能与 mdl 模型描述的情况不完全相符。这就需要针对生成的 RTL 级 VHDL 代码进行功能仿真。

当在 SignalCompiler 窗口选项页中设定 **Generate Stimuli for VHDL Testben** 后，DSP

Builder 在“MDL to VHDL”过程中会自动生成针对 HDL 仿真器 ModelSim 的测试文件。因此，在这里，需要使用 ModelSim 对生成的 VHDL 代码进行功能仿真。ModelSim 是一个基于单内核的 Verilog/VHDL 混合仿真器，是 Mentor Graphics 的子公司 Model Technology 的产品。

打开 ModelSim 环境，图 9-36 是 ModelSim 的主窗口。选择主窗口上的菜单“Tools”→“Execute Macro...”，在打开的文件选择对话框中，切换到 sinout 模型 sinout.mdl 所在的目录，接着选择 tb_sinout.tcl 来执行。其中，tb_sinout.tcl 是 DSP Builder 的 SignalCompiler 产生，若不存在，可再重复一次 5.2.3 小节的流程。如果一切配置正常的话，ModelSim 就开始执行 tcl 脚本：tb_sinout.tcl，开启仿真，随后就自动打开“wave”窗口，显示仿真结果（图 9-37）。

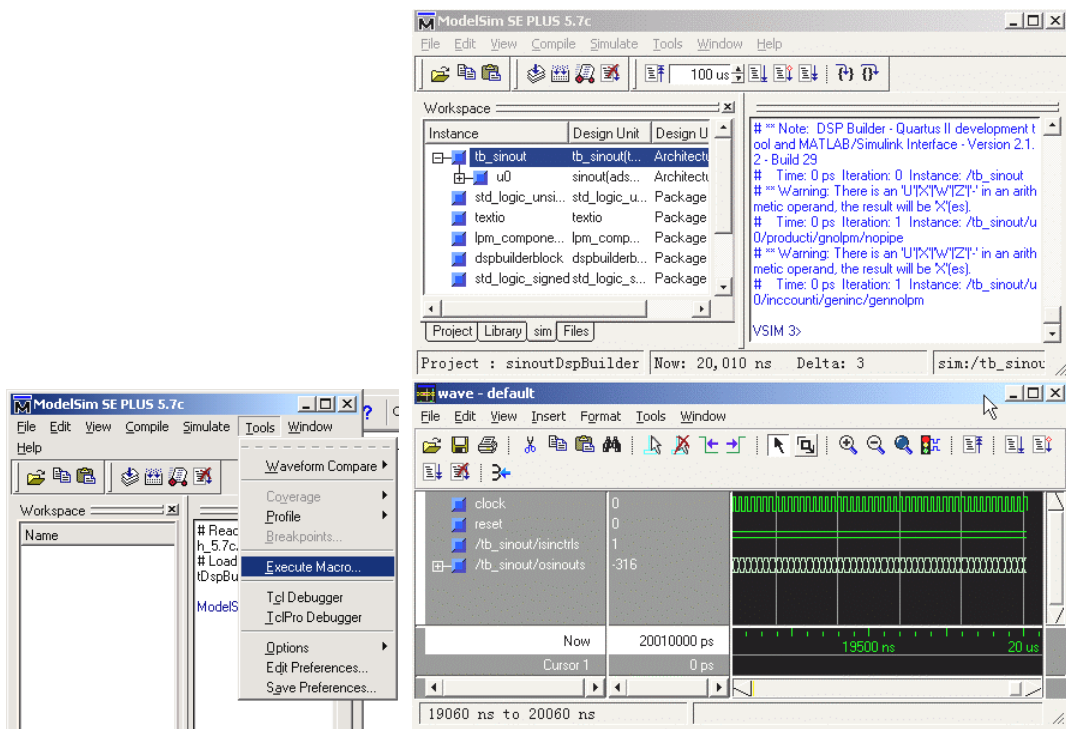



图 9-36 准备执行 tcl 文件

图 9-37 ModelSim 仿真结果

不过图中显示的仿真波形与 simulink 中的仿真结果没有可比性。点击“wave”窗口工具栏上的  图标，使之显示全部波形。在此可以把“wave”窗口中的“/tb_sinout/osinout”信号改成类似模拟信号的显示格式。先用鼠标选中“/tb_sinout/osinout”信号。在“wave”窗口的“View”菜单中选择“Signal Properties”，打开“Wave Signal Properties”对话框（图 9-38）。在“View”选项页中，修改“Radix”为 Unsigned（无符号数）；在“Format”选项页中（图 9-39），修改下列几项：“Format”选择 Analog；“Height”为 100；“Analog Display”中“Scale”为 0.4。

点击“OK”确定后，如图 9-40 所示。可以看到与 simulink 里的仿真结果基本一致。

注意：用于 ModelSim 仿真的 TestBench 文件中的输入信号激励由 SignalCompiler 根据 Simulink 的仿真结果产生。因此，若需要产生正确的仿真激励，必须先要在 simulink 中进行设置和仿真运行。

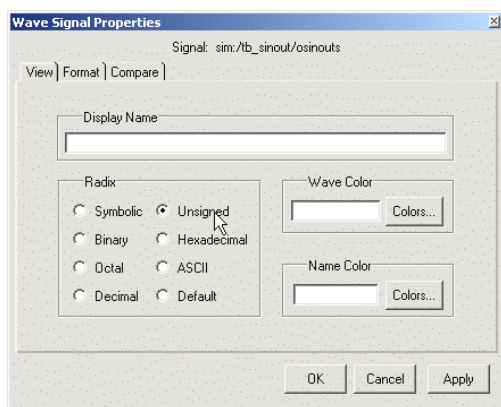


图 9-38 ModelSim 的信号设置

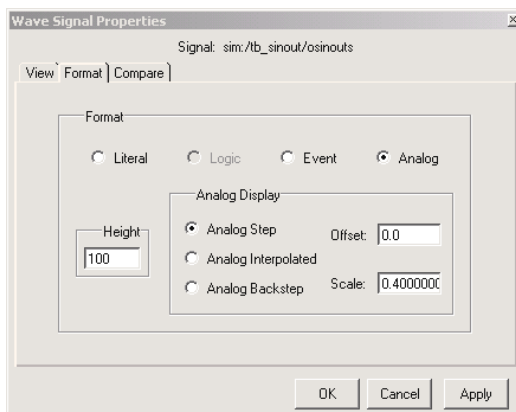


图 9-39 设为 Analog

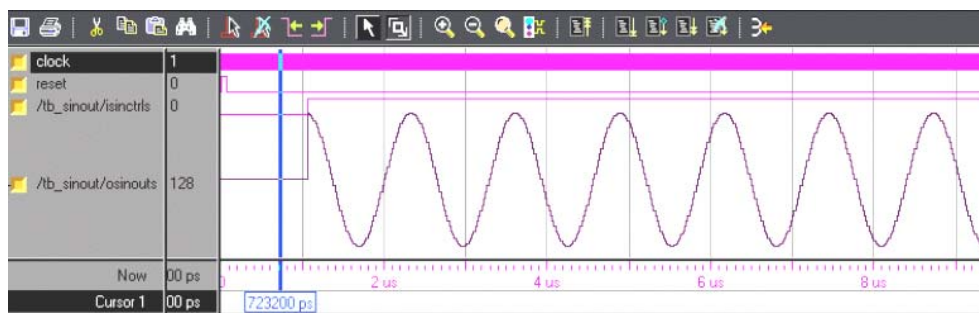


图 9-40 sinout 工程的 ModelSim 仿真波形（RTL 级仿真）

此外，如果要产生 ModelSim 能用的 tcl 仿真文件，原 VHDL 文件中不应包含 LPM 模块，因为 ModelSim 通常无法了解嵌在源程序中的 LPM 功能。因此在利用 ModelSim 仿真前，应在 simulink 编辑窗中消去 SinLUT 和 Product 模块的“LPM”选择，然后进行一次系统仿真，再用 SignalCompiler 将其转换成 VHDL 程序和 tcl 文件，进行 ModelSim 仿真。完成后恢复“LPM”选项，以便为 QuartusII 提供最后的时序仿真文件。

9.2.5 使用 QuartusII 实现时序仿真

ModelSim 完成的 RTL 级仿真只是功能仿真，其仿真结果并不能更精确地反映电路的全部硬件特性。进行门级的时序仿真仍然是十分重要的。好在 SignalCompiler 已将 MATLAB 上的仿真信息转变成了可用于 QuartusII 进行时序仿真的激励信息及相关仿真文件：sinout_quartus.tcl，因此可以很容易地完成此项仿真任务。步骤如下：

- 1、打开 QuartusII 环境，选择菜单“File”→“Open Project ...”，定位到 sinout 模型所

在目录，打开 DSP Builder 已自动建立好的 QuartusII 工程文件：sinout.qpf（见图 9-41）。注意，自动产生这个工程文件的前提是必须在 SignalCompiler 窗（图 9-34）中执行完第二个按钮：2-Synthesis。

2、上文中提到，在 SignalCompiler 中的 QuartusII 编译，具体的器件由 QuartusII 自动决定，可实际使用中，器件往往不是 QuartusII 自动选定的那个型号，管脚也不是 QuartusII 自动分配的管脚。这些都需要在 QuartusII 中进行修改。所以这里须按照前面章节中叙述的方法选择器件型号。选择菜单“Assignments”→“Device ...”，在相应的对话框中选择合适的器件（Cyclone 系列器件），如 EP1C6Q240C8，然后启动编译，即执行菜单“Processing”→“Start Compilation”。

3、双击图 9-41 左侧的工程名“sinout”，打开转换好的 VHDL 文件，了解生成的程序，特别是端口实体的情况（图 9-42）。然后再打开已生成的仿真激励文件 sinout.vec（图 9-43）。

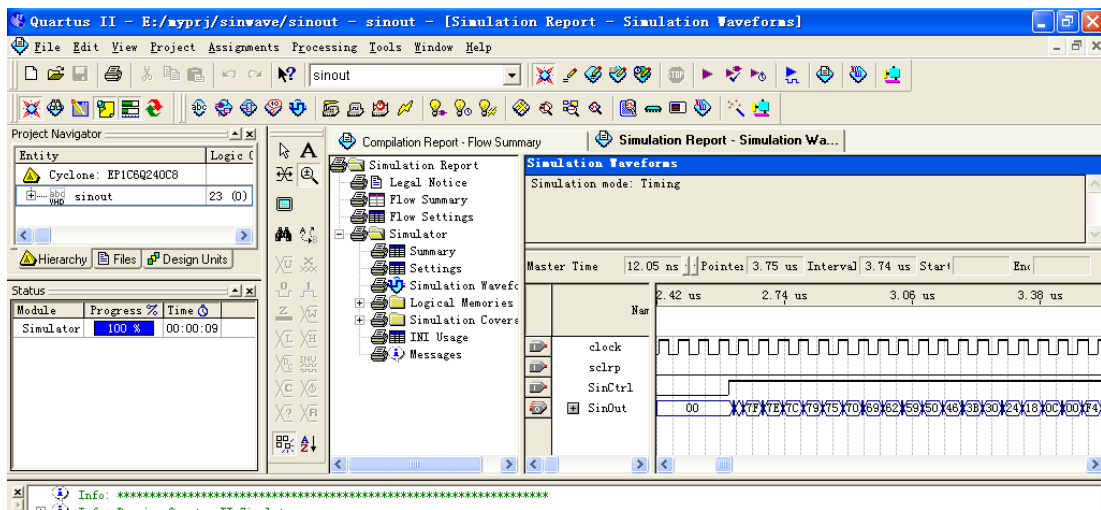


图 9-41 打开 QuartusII 工程进行编译和时序仿真

```

32 Entity sinout is
33     Port(
34         clock      : in std_logic;
35         sclrp      : in std_logic := '0';
36         SinCtrl    : in std_logic;
37         SinOut     : out std_logic_vector(7 downto 0)
38     );
39 end sinout;

```

图 9-42 QuartusII 工程 VHDL 程序实体

比较图 9-42，9-43，如果发现端口信号名不一致，应该修改图 9-43，使其一致。然后将图 9-43 另存为 vwf 文件：sinout.vwf。

4、设置仿真文件路径。选择菜单“Assignments”→“Settings”，弹出如图 9-44 窗口。在此窗左侧的 Category 栏选择 Simulator Settings 项，在右侧的 Simulation mode 栏选择 Timing，即选择时序仿真；在 Simulation input 栏选择仿真文件 sinout.vwf。关闭此窗。

5、启动仿真编译。即执行菜单“Processing”→“Start Simulation”。结束后能看到图

9-41 所示的界面。然后打开仿真波形报告窗，如图 9-45 所示，此窗已自动打开：即执行菜单“Processing”→“Simulation Report”。图 9-45 即为时序仿真波形。

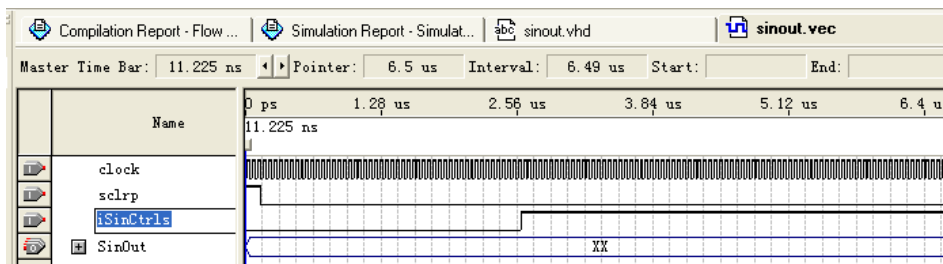


图 9-43 打开 QuartusII 工程的 vec 仿真激励文件

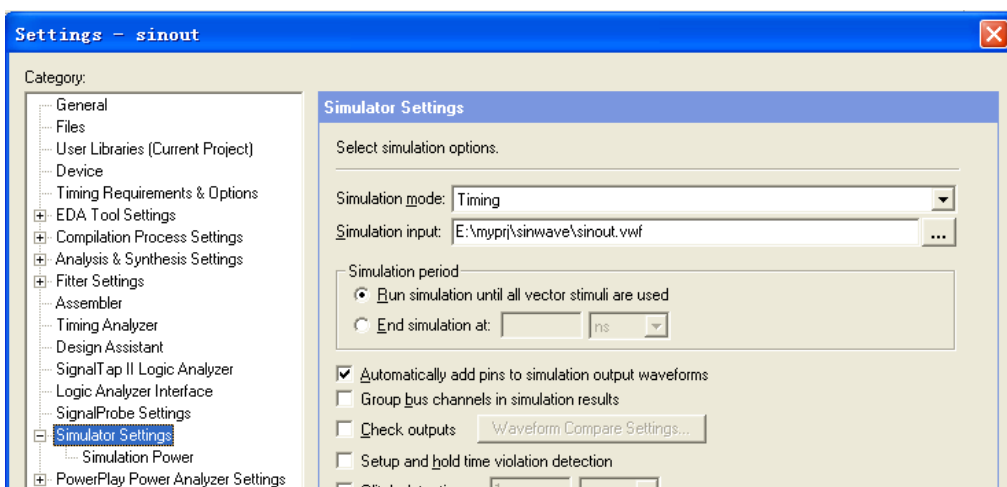


图 9-44 设置仿真文件路径

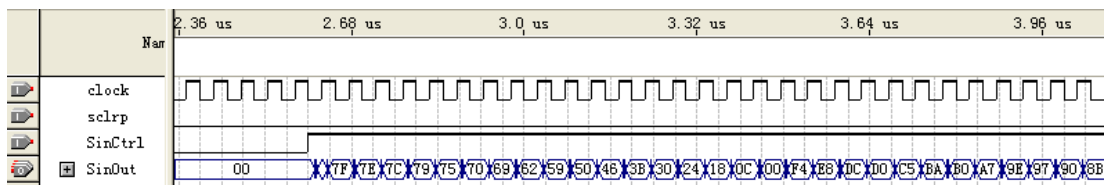


图 9-45 sinout 工程的 QuartusII 仿真波形（门级时序仿真）

9.2.6 硬件测试与硬件实现

在此按照第 4 章中介绍的方法锁定管脚 (Pin)：选择菜单“Assignments”→“Assign Pins ...”。首先打开 sinout 工程文件 sinout.vhd，了解端口情况 (图 9-42)，由于目标器件已确定为 EP1C6Q240C8，对 D/A 输出的 8 位 SinOut (7 downto 0) 可分别接 PIO31-PIO24，对应引脚号是：136、135、134、133、132、128、41、21；系统时钟 clock 接实验板的“clock0”，对应第 28 脚；清 0 信号 sclrp 接键 7，对应第 239 脚；输出控制 SinCtrl 接键 8，对应第 240

脚。接着进行编译，完成适配过程。最后是进行下载，连接好 FPGA 开发板。选择模式 5，clock0 可接 65536Hz；打开 +/-12V 电压，用示波器检测 D/A 的输出，把键控 SinCtrl 设置为有效（将键 8 设成高电平，键 7 设为低电平），就可以在示波器上看到产生的正弦波了。然后将实测结果与在计算机上进行的时序仿真结果进行比较。

最后可以利用嵌入式逻辑分析仪 SignalTapII 测定芯片内的实时波形，从而完成更深入的系统测试。

从 Simulink 到 SignalCompiler，再到 QuartusII 的设计验证十分方便，在第一个流程后，即最后实现硬件信号输出后，若再想改动 Simulink 中的 sinout.mdl 图，应该注意两点：1、内部电路结构和设置可以改，但端口信号名不要改，如输入的 SinCtrl，SinOut，因为此信号的引脚已被锁定，不便改变；2、改动 sinout.mdl 图后只宜作系统仿真和 VHDL 文件转换，不宜作综合，即最多只能对图 9-34 所示的界面执行第 1 个按钮，否则将可能把原来设定好的引脚全部冲掉。为了保存引脚信息，综合与适配必须进入 QuartusII 后进行（图 9-41）。

9.3 DSP Builder 层次化设计

上一节，已在 Matlab/Simulink 中设计了一个简单的正弦波发生器。但在许多实用的领域中，如通信领域中，实际需要实现的电路模型往往要复杂得多，如果把所有模块放在同一个 Simulink 图中，设计图会显得非常复杂、庞大，不利于阅读和排错。这时，就必须采用层次化设计方法来设计模型了。本节介绍 DSP Builder 的层次化设计的基本步骤。

在 Matlab 的 Simulink 建模时，可以使用 SubSystem 来完成子系统的封装和调用。DSP Builder 继承了 simulink 的子系统（SubSystem）来完成 DSP 模型的层次化设计。下面以一个示例来具体说明 DSP Builder 的层次化设计。步骤如下：

1、首先建立一个新的模型，命名为 subsint 模型，仍然依照图 9-2 连接起来，并以文件名 subsint 存盘。然后在 subsint 模型窗口中，按住鼠标左键，然后移动鼠标画框，选中图中除了 SignalCompiler、Step 模块以外的所有模块。（可以通过按住键盘上的 Shift 键，用鼠标左键点击来改变模块的选择情况。）

2、在选中的模块上点击鼠标右键，在弹出的右键菜单中（图 9-46），选择“Create SubSystem”，建立子系统。图 9-47 中所示的是建立子系统后 subsint 模型的 simulink 原理图，可以看到原来被选中的那些模块和连线都消失了，只剩下一个新建立的子系统模块：“Subsystem”。在新生成的 Subsystem 模块上共有两个端口：In1、Out1。

3、修改子模块名。事实上图 9-47 显示的是 subsint 模型的顶层（Top Level）原理图。这时用鼠标双击 Subsystem 子系统模块，就会弹出“subsint/Subsystem”窗口，显示 Subsystem 子系统模块封装的原理图（图 9-48）。可以看出，封装后的模块自动增加了两个 simulink 的端口，即：In1 和 Out1。

在打开的子系统模块中，可以任意地增删模块；放置仿真用的 simulink 库的模块；引入“Scope”等。不过某些 DSP Builder 库的模块只能放置在顶层原理图中，比如

SignalCompiler 模块。假如在子系统模块中放置了 SignalCompiler 模块,只可以进行 simulink 的仿真,但不能使用 SignalCompiler 来生成相应的 VHDL 代码。

同普通的 DSP Builder 模块一样,子系统模块也可以自行命名,操作方法同普通模块。见图 9-49,可以把 Subsystem 子系统模块的名字修改为 `singen`。

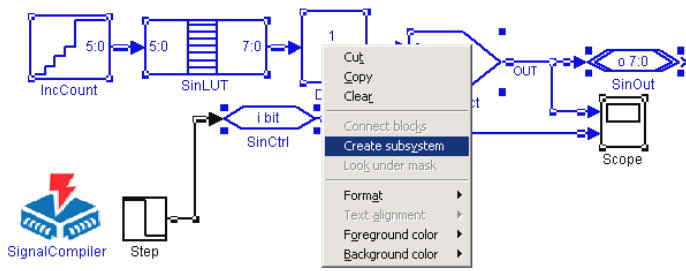
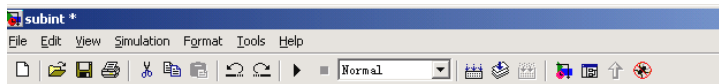


图 9-46 准备建立 subsystem

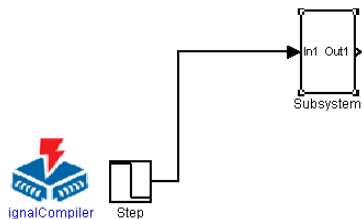


图 9-47 建立 subsystem 后

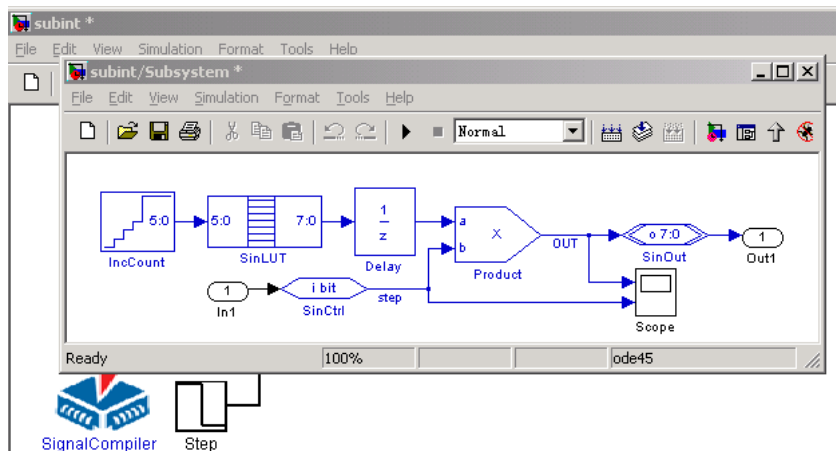


图 9-48 subsin/subsystem 子系统图

4、修改子系统内端口名。在子系统输入输出端口的名字也可以定制。在如图 9-48 所示的“subsin/singen”子系统窗口,修改输入端口“`In1`”为“`CtrlIn`”,输出端口“`Out1`”为“`OutSin`”(图 9-50)。当修改好端口后,修改的结果马上在 `subint` 模型的顶层原理图上的 `singen` 子系统模块上显示出来。不需要任何的更新操作。

图 9-51 就是更改 `singen` 子系统端口名称后, `subint` 模型顶层原理图的显示结果。

Simulink 的子系统的端口的增删操作较简单,可以直接在子系统的图上加入或者删除输入、输出端口。在调用该子系统模块的上层原理图上,立即就会更改相应的子系统模块的显示。

注意,如图 9-48, 9-58 的子系统,其端口必须接上来自 IO & Bus 库的输入输出端口,

如 Input、Output，否则无法转换成 VHDL。

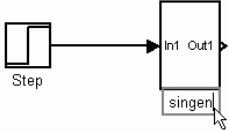


图 9-49 修改子系统名

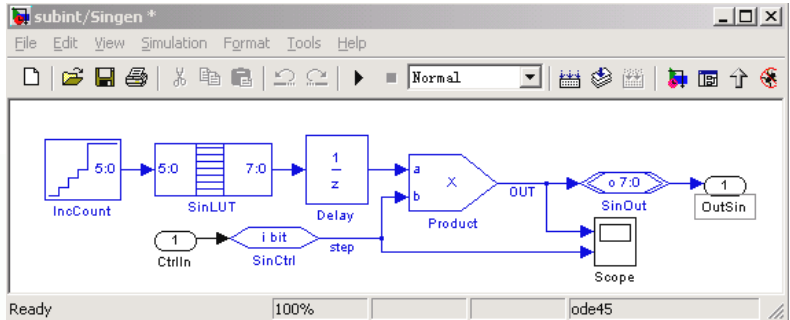


图 9-50 修改 SubSystem 的端口

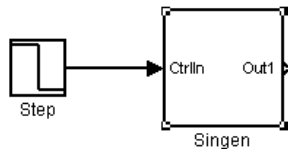


图 9-51 顶层图的变化

对于生成的子系统模块，可以当成一般的模块来使用，允许任意复制、删除子系统模块，或者再组合其他的模块来生成更高一层的子系统。

5、完成顶层设计。在图 9-51 的基础上，添加另外一些 DSP Builder 模块来构成一个实际的电路。图 9-52 是最后的 subsint 模型图。下面列出了新增模块需要修改的参数值：

1) Singen 子模块中的元件设置同上一节，且其中所有模块的总线类型皆设为 signed Integer。

2) Offset 模块：(Altbus)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“signed Integer”

参数“Bus Type”设为“Internal Node”

参数“number of bits”设为“8”；其余为 0。

3) Parallel Adder Subtractor 模块：(Parallel Adder Subtractor)

库：Altera DSP Builder 中 Arithmetic 库

“Add(+)Sub(-)”设为“+”；“Pipeline”设为 1

Clock Phase Selection 设 1。

4) xSin 模块：(Altbus)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“Unsigned Integer”

参数“Bus Type”设为“Output port”

参数“number of bits”设为“8”

5) CtrlIn 模块：(Altbus)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“Single Bit”

参数“Bus Type”设为“Input port”

6) Constant 模块：(Constant)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“signed Integer”

参数“Constant Value”设为 127；

参数“Sampling Period”设为1。

7) Mux模块

库: simulink中Signal Routing库

参数“Number of Inputs”设为“2”

参数“Display Option”选择“bar”。

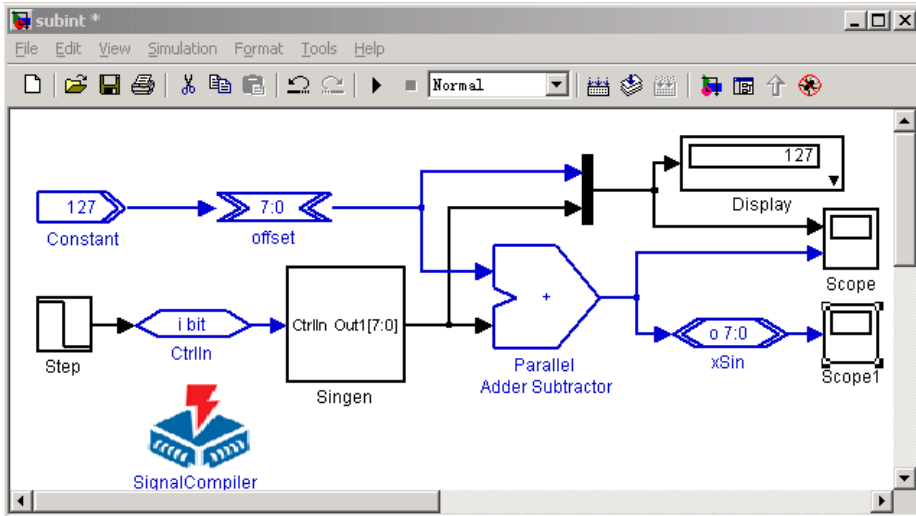


图 9-52 5-57 含 subsystem 的 subsint 模型

建立好 subint 模型后, 按照 sinout 模型的例子, 给定合适的 Step 模块的激励参数, 启动 subsint 模型的仿真。图 9-53、9-54 显示的就是 subsint 模型的仿真波形。在图上可以看出, subsint 模型在 sinout 模型的基础上增加了波形的电压偏移, 其功能与图 9-26 电路的功能是相同的。

注意, 图 9-53,9-54 是 subsint 模型顶层原理图上 Scope 波形显示。在这个例子中, 子系统 singen 中还含有一个 Scope 模块, 同样也可以对相应节点的波形进行观察。

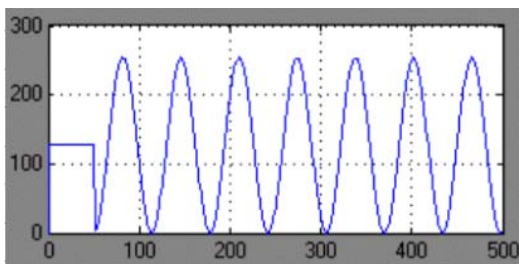


图 9-53 Scope1 波形图

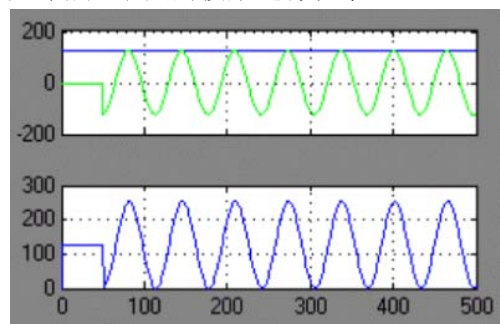


图 9-54 Scope 波形图

不过, 虽然可以进行 simulink 仿真, 但是还不能进行 SignalCompiler 分析。按照前面介绍的步骤作的 subsint 模型中的 singen 子系统模块, 必须进行设置才可以让 SignalCompiler 识别为 DSP Builder 的子系统。以下给出设置步骤:

- 1、在打开的 subsint 模型窗口中, 先选中子系统模型 singen, 然后选择菜单“Edit”→

“Mask subsystem...”项（图 9-55，或右键点击该模块）。打开“Mask Editor:singen”设置对话框。注意“Mask subsystem...”项在设置后即变为“Edit Mask”项。

2、在“Mask Editor:singen”对话框中选择“Documentation”选项页（图 9-56），设置“Mask type”为“SubSystem AlteraBlockSet”（子系统 Altera 模块集）。

注意，输入时没有引号，且大小写和空格要严格按照此字符串形式。

设置完“Mask type”后，SignalCompiler 就可以正常地生成 VHDL 代码。不过，与 sinout 模型生成的 VHDL 代码有所不同，sinout 模型在转换时，只生成一个 VHDL 文件（TestBench 文件除外）。而 subsint 模型在转换后生成了两个 VHDL 文件：subsint.vhd 和 singen.vhd

其中 subsint.vhd 是顶层的 VHDL 文件，singen.vhd 是 singen 子系统的 VHDL 文件。SignalCompiler 对于子系统模块的处理是：每个子系统模块都产生一个 VHDL 文件，在上一级的 VHDL 文件中加以例化调用。

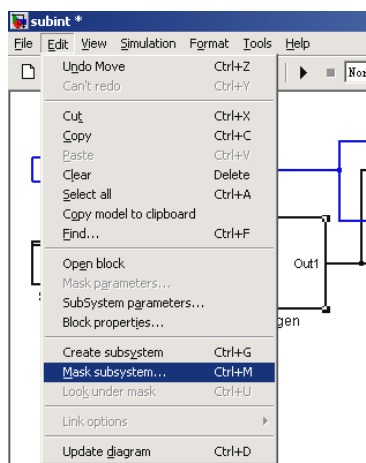


图 9-55 SubSystem 设置

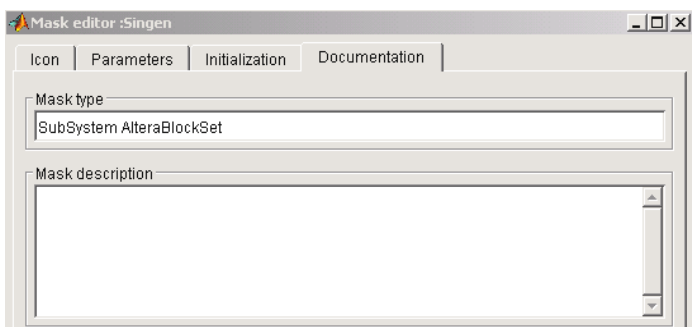


图 9-56 编辑 singen 的“Mask type”

通过对每一层次的子系统调用，可以实现复杂的电路模型结构。为了便于设计者管理，Simulink 提供了“Model Browser”来显示模型的结构。在模型窗口选择“View”→“Model Browser Options”→“Model Browser”项，在“Model Browser”前打上“√”。

9.4 基于 DSP Builder 的 DDS 设计

DDS 的基本原理和设计方法已在第 7 章中讨论过了，本章首先介绍基于 Matlab 和 DSP Builder 平台的 DDS 设计方法，然后给出几个基于 DDS 的实用系统的设计方法。

9.4.1 DDS 模块设计

图 9-57 是 DDS 图的顶层设计，其子系统 SubDDS 的结构图如图 9-58 所示。设计流程可以这样：先在 Simulink 中新建一个模型，调用 DSP Builder 模块构成如图 9-58 的相应电

路，再用层次设计方法作成模块 DDS 模型子系统 SubDDS（端口必须接上来自 IO & Bus 库的输入输出端口）。然后设计如图 9-57 所示 DDS 顶层电路模型。

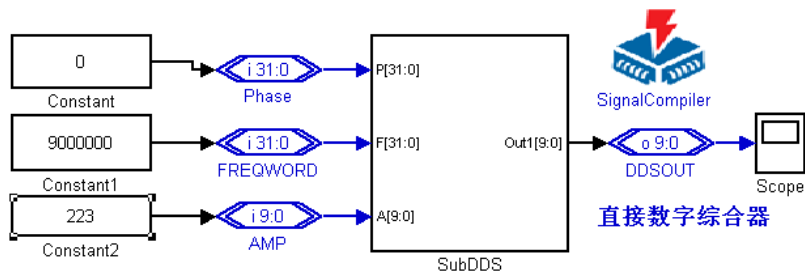


图 9-57 DDS 系统

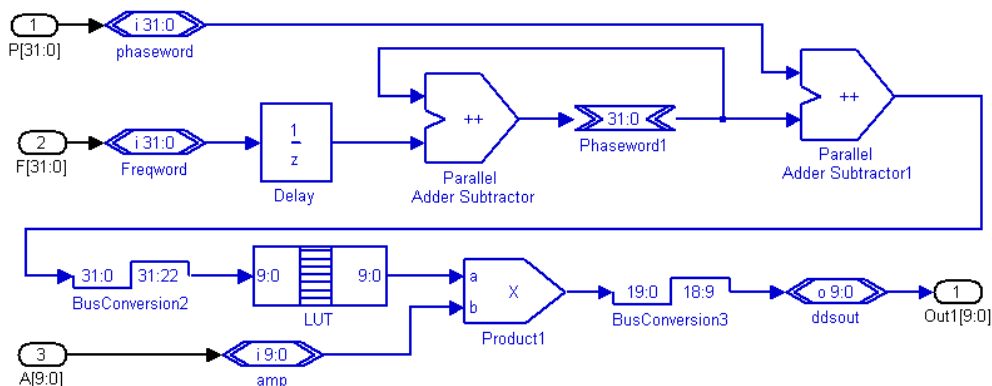


图 9-58 DDS 子系统 SubDDS

图中，DDS 子系统 SubDDS 共有三个输入，分别为 Freqword（32 位频率字输入）、Phaseword（16 位相位字输入）、Amp（10 位幅度控制字输入）；一个输出，即 10 位 DDSout 输出。注意，通常由于 10 位高速 D/A 是无符号器件，在图 9-57 的输出口应该增加一些转换电路，将输出的数据类型转换成无符号的类型。对于输入的控制信号也要作一些位数和类型的变换。此外，如果发现在仿真参数设置上出现问题，可以先用来自 Altera DSP Builder 库中 IO & Bus 中的“Constant”元件代替图 9-57 中的 Constant2。

设置 simulink 的仿真停止时间 Stop Time 为 5，仿真步进 Fixed Step Size 为 1e-3。相位、频率和幅度控制字输入按图 9-57 所示，则输出波形即如图 9-59 所示；若相位、频率和幅度控制字输入分别取 500000000、4000000 和 89，则输出波形如图 9-60 所示。

子系统 SubDDS 输入输出模块的参数设置为：

Freqword 模块：(Altbus)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“signed Integer”

参数“Node Type”设为“Input port”

参数“number of bits”设为“32”

Phaseword 模块：(Altbus)

库：Altera DSP Builder 中 IO & Bus 库

参数“Bus Type”设为“signed Integer”
参数“Node Type”设为“Input port”
参数“number of bits”设为“32”

Amp模块: (Altbuss)

库: Altera DSP Builder中IO & Bus库
参数“Bus Type”设为“signed Integer”
参数“Node Type”设为“Input port”
参数“number of bits”设为“10”

DDSout模块: (Altbuss)

库: Altera DSP Builder中IO & Bus库
参数“Bus Type”设为“signed Integer”
参数“Node Type”设为“Output port”
参数“number of bits”设为“10”

由 Delay、Parallel Adder Subtractor 和 Phaseword1 模块构成相位累加器, 参数如下:

Parallel Adder Subtractor模块: (Parallel Adder Subtractor)

库: Altera DSP Builder中Arithmetic库
参数“Number of Inputs”设为“2”
“Add(+)Sub(-)”设为“++”
选择“Pipeline”
参数“Clock Phase Selection”

Delay模块: (Delay)

库: Altera DSP Builder中Storage库
参数“Depth”设为“1”
参数“Clock Phase Selection”设为“1”

Phaseword1模块: (Altbuss)

库: Altera DSP Builder中IO & Bus库
参数“Bus Type”设为“signed Integer”
参数“Node Type”设为“Internal Node”
参数“number of bits”设为“32”

注意加法器模块使用了“Pipeline (流水线)”, 已经内含了寄存器, 因而加法器出来后就不需要有延时模块存在。

相位调整部分由 Parallel Adder Subbactor1 模块和 BusConversion2 构成, 参数如下:

Parallel Adder Subtractor1模块: (Parallel Adder Subtractor)

库: Altera DSP Builder中Arithmetic库
参数“Number of Inputs”设为“2”
“Add(+)Sub(-)”设为“++”
选择“Pipeline”
参数“Clock Phase Selection”

BusConversion2模块: (BusConversion)

库: Altera DSP Builder中IO & Bus库
参数“Input Bus Type”设为“signed Integer”
参数“Input [number of bits].[]”设为32
参数“Output Bus Type”设为“Signed Integer”

参数“Output [number of bits][.]”设为“10”
 参数“Input Bit Connected to Output MSB”设为“31”
 参数“Input Bit Connected to Output LSB”设为“22”
 使用“Round”

剩下的模块构成幅度控制部分，模块参数如下：

Product模块：(Product)

库：Altera DSP Builder中Arithmetic库

参数“Pipeline”设为“2”

参数“Clock Phase Selection”设为“1”

不选择“Use LPM”

BusConversion3模块：(BusConversion)

库：Altera DSP Builder中IO & Bus库

参数“Input Bus Type”设为“signed Integer”

参数“Input [number of bits][.]”设为“20”

参数“Output Bus Type”设为“Signed Integer”

参数“Output [number of bits][.]”设为“10”

参数“Input Bit Connected to Output MSB”设为“18”

参数“Input Bit Connected to Output LSB”设为“9”

使用“Round”

使用“Saturate”

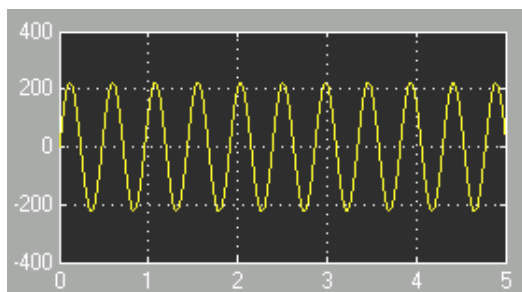


图 9-59 DDS 系统输出波形

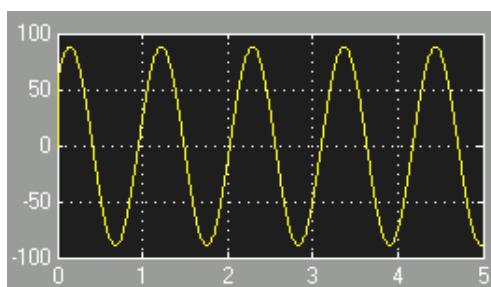


图 9-60 DDS 系统输出波形

9.4.2 FSK 调制器设计

由于 FSK 调制器实际上是 DDS 的简化应用模型。二进制数字频率调制 (2FSK) 是利用二进制数字基带信号控制载波进行频谱变换的过程。在发送端，产生不同频率的载波振荡来传输数字信息“1”或“0”，在接收端，把不同频率的载波振荡还原成相应的数字基带信号。相邻两个振荡波形的相位可能是连续的，也可能是不连续的，因此有相位连续的 FSK 及相位不连续的 FSK 之分。

FSK 调制的方法有两种：

1、直接调频法。用数字基带矩形脉冲控制一个振荡器的某些参数，直接改变振荡频率，输出不同频率的信号。

2、频率键控法。用数字矩形脉冲控制电子开关在两个振荡器之间进行转换，从而输出不同频率的信号。

在此设计一个 FSK 模型，在调制方法上选择直接调制法。采用 DDS 方法来生成频率可控的正弦信号，利用数字基带信号控制 DDS 的频率字输入，实现 FSK 调制。

图 9-61 所示的是一个简化的 DDS 结构，由 8bit 累加器作为相位累加器，由二选 1 多路选择器来选择累加器的相位，相位是由数字基带信号控制的。

采用改变相位增量来控制频率的方法，可以产生相位连续的调制波形。在实际应用中，数据输出 Out8 需要经过 DAC 进行数模转换，然后经低通滤波器后，产生最终的模拟输出信号。FSK 调制的仿真结果如图 9-62 所示，高电平控制的时候，正弦波的频率较高，而低电平的时，正好相反。最后通过 SignalCompiler 及 QuartusII 在 FPGA 上完成硬件实现后，可以在示波器上看到类似的波形输出。

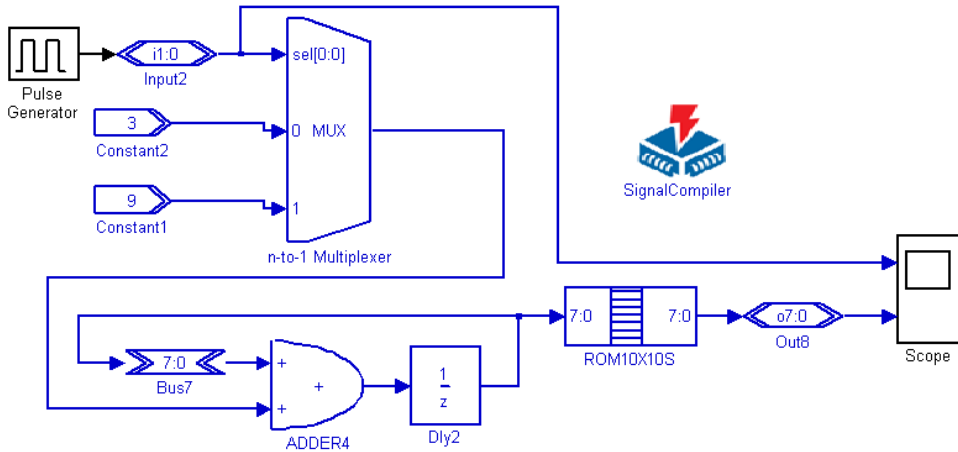


图 9-61 FSK 调制模型

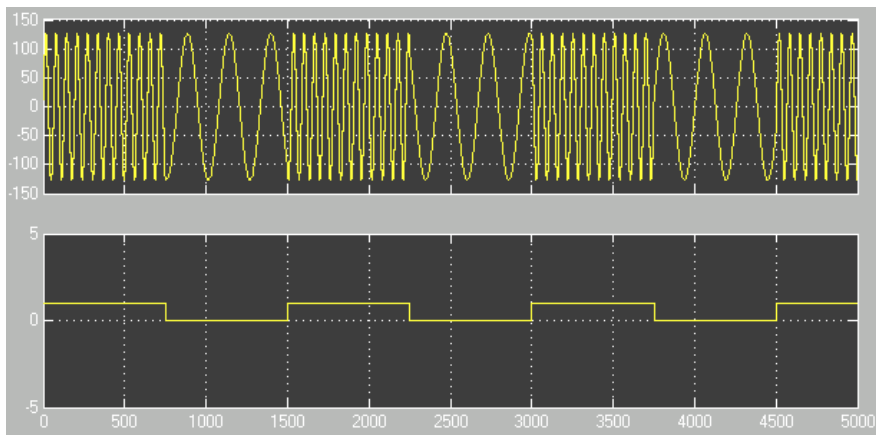


图 9-62 FSK 调制的 Simulink 仿真结果

9.4.3 正交信号发生器设计

对于通信上的应用，往往需要得到一对正交的正弦信号，以便进行正交调制和正交解

调。在用模拟的压控振荡器 VCO 时，输出一组完全正交的信号较为困难，而对于 DDS 而言，只要在基本 DDS 结构中增加一块 ROM 查找表，在两块 ROM 中分别放置一对正交信号即可（如一个放置 Sin 表、另一个放置 Cos 表）。

幅度调制在很多场合是需要的，通过改进基本 DDS 结构，在正弦 ROM 表后、D/A 前放置一个幅度控制模块，一般采用乘法器来实现。此信号发生器的子系统如图 9-63 所示。

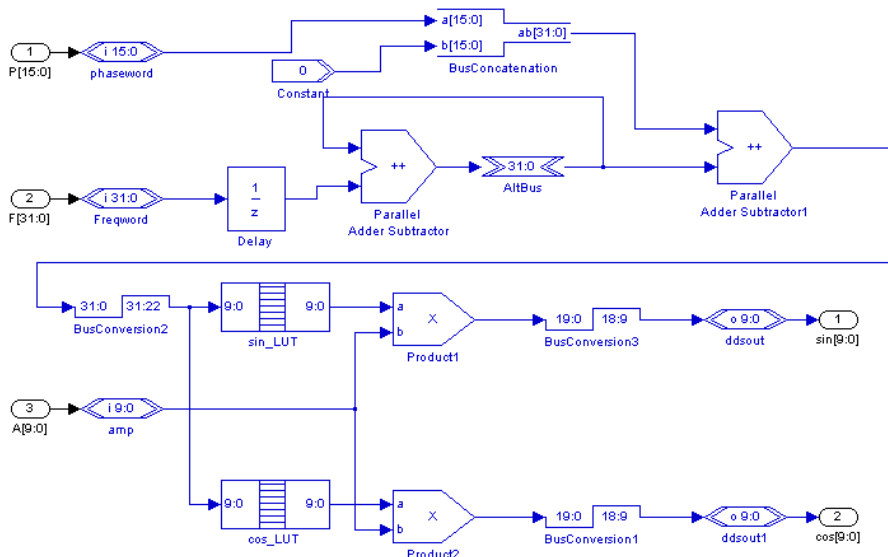


图 9-63 正交信号发生器 MDL 模型

9.4.4 数控移相信号发生器设计

在 2003 年的全国大学生电子设计竞赛 C 题中，有一项内容要求设计一个数字移相信号发生器，要求输出两路正弦信号，一路为参考信号，另一路是可数控的移相信号，并且这两路可同步进行幅度和频率数控。即对于这两路输出的正弦信号在相位、频率和幅度 3 个参数上都能完成等步长数控步进，而且还能对指定的参数值进行设定。

不难发现，如果使用 DDS 模型的设计方案，此类赛题是十分容易完成的，且能高精度高效率地完成。高精度指设计的项目能达到很高的技术指标；高效率指可以在很短的时间内就能成功的完成设计任务。

图 9-64 就是一个基于 DDS 的数字移相信号发生器完整的设计。他们的 Simulink 仿真波形如图 9-65 所示。最上的波形是正弦波输出基准信号；中间是随相移控制的信号；地部是 ROM 的地址信号。

注意，在实际的硬件实现前也必须对输入输出信号的有符号数据类型作一些转换。此外，如果要提高输出信号各项参数的指标，就必须增加各参数对应的功能模块和电路总线宽度。这时如果用单片机来控制各项参数的数控数据输入，就必须在输入端口上加上适当的锁存电路。例如，如果相位控制数有 32 位，则必须在其端口增加 4 个 8 位锁存器和一个 2-4 译码器，单片机要分 4 次才能对相位控制数赋值完毕。

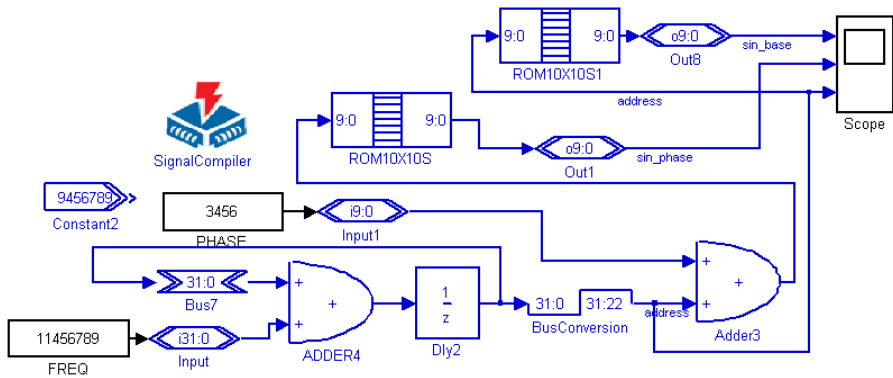


图 9-64 数字移相信号发生器 MDL 模型

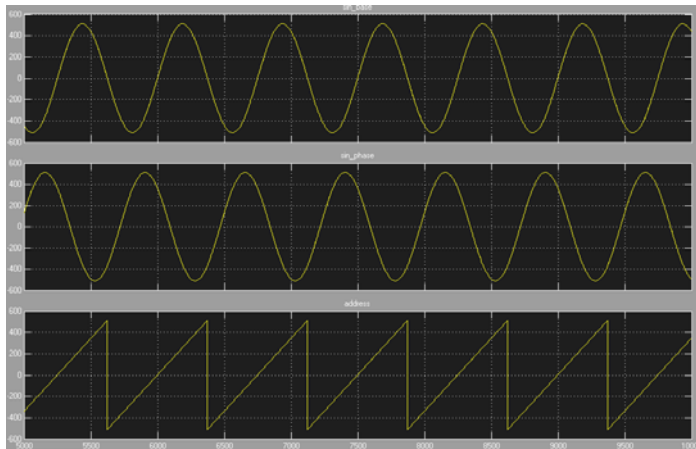


图 9-65 数字移相信号发生器输出波形

9.4.5 幅度调制信号发生器设计

幅度调制信号发生器是 2005 年大学生电子设计竞赛题中的一个设计项目。图 9-66 是用 MATLAB 和 DSP Builder 设计的幅度调制信号发生器 AM 电路。仿真编译后可以在 FPGA 上实现。图 9-67 是仿真波形，最上的是载波，中间的是调制波，下图是 AM 被调制后的波。

式 9-4 是调制信号表式，其中 F 是调制后的输出信号， F_{dr} 是载波信号， F_{am} 调制信号， m 是调制度 $0 < m < 1$ 。

$$F = F_{dr} \cdot (1 + F_{am} \cdot m) \tag{9-4}$$

注意， F_{dr} 和 F_{am} 都是有符号函数。

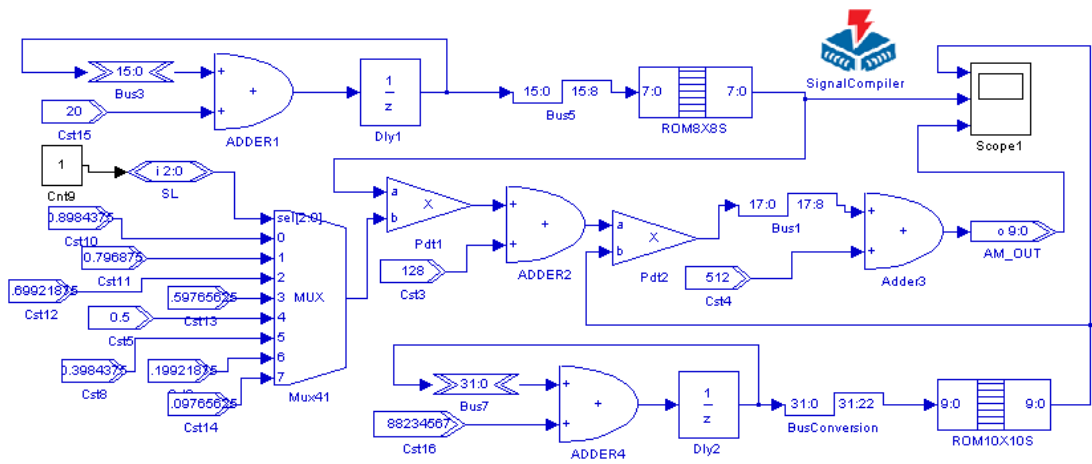


图 9-66 AM 发生器模型

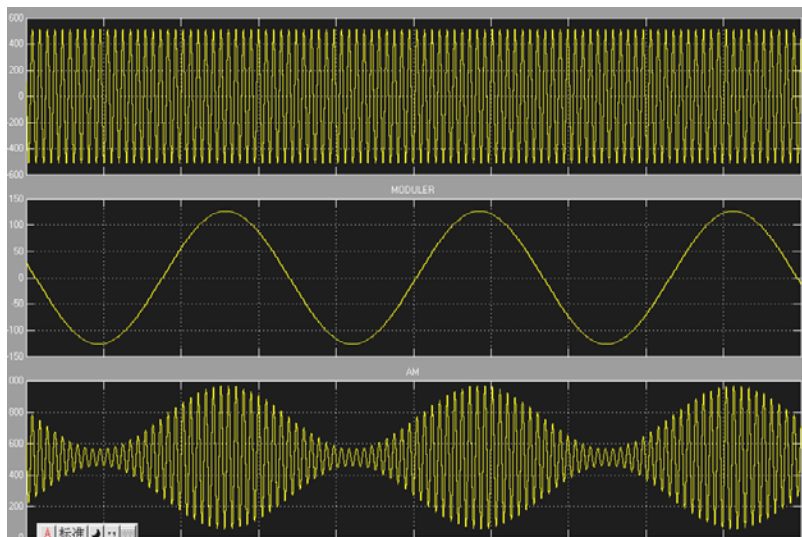


图 9-67 AM 模型仿真波形

根据式 9-4 可以作出图 9-66 电路模型。其中元件 ADDER1、Dly1、Bus5、Bus3、ROM8X8S 构成一个 DDS 模块，产生调制信号 F_{am} ，进入乘法器 Pdt1 的 a 端；进入 b 端的是 Mux41 的输出，这是一个 8 选 1 的多路选择器，对预设好的 8 个调制度数据进行选择，选通信号由 SL 输入。

ADDER2 将乘积（乘积项取高 8 位整数）与 128 相加。由于是 8 位乘积，故 128 类似于 9-4 式中的 1。和进入第 2 个乘法器 Pdt2 的 a 端。

元件 ADDER4、Dly2、BusConversion、Bus7、ROM10X10S 构成另一个 DDS 模块，产生载波信号 F_{dr} ，进入乘法器 Pdt2 的 b 端。

将乘积的高 10 位与 512 相加，进入 10 位 DAC 输出。

图 9-66 中，cst15 输入的数据控制调制信号频率；cst9 的数据控制调制度（图 9-67 的调制度是 0.79）；cst16 输入的数据控制载波信号频率。

9.5 数字编码与译码器设计

在数字通信领域，为了提高数字信号的传输质量，必须对传输的数字信号在发送端进行编码操作，在接收端进行译码操作。利用 DSP Builder 还可以构建编码或译码电路模型。另外，DSP Builder 借助于专用的编码解码 IP Core，可以实现复杂的编码、译码操作。

本节将介绍一些常用的编解码模型的设计，涉及到 IP 核的设计放在第 10 章中讨论。

9.5.1 伪随机序列

对于数字信号传输系统，传送的数字基带信号（一般是一个数字序列），由于载有的信息在时间上往往是不平均的（比如数字化的语音信号），对应的数字序列编码的特性，不利于数字信号的传输。对此，可以通过对数字基带信号预先进行“随机化”（加扰）处理，使得信号频谱在通带内平均化，改善数字信号的传输；然后在接收端进行解扰操作，恢复到原来的信号。伪随机序列广泛应用于这类加扰、解扰操作中。下面以一类伪随机序列，即 m 序列为例，用 DSP Builder 构建一个伪随机序列发生器。

m 序列，即最长线性反馈移位寄存器序列，是一种比较常见的伪随机序列发生器，可由线性反馈寄存器（Linear Feedback Shift Registers, LFSR）来产生。如图 9-68。

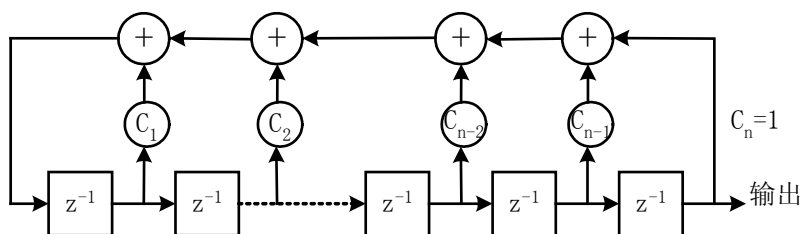


图 9-68 线性反馈移位寄存器的构成

其特征多项式可表示为：

$$F(x) = \sum_{i=0}^n C_i x^i \quad 9-5$$

在图中涉及的乘法和加法都是指模二运算中的乘法和加法，即逻辑与和逻辑异或。

要产生最长的线性反馈移位寄存器序列的 n 级移位寄存器，其特征多项式必须是 n 次本原多项式。比如 $n = 5$ ，可以生成 m 序列的 5 级 LFSR 的特征多项式，即：

$$x^5 + x^2 + 1 \quad 9-6$$

上式可生成的 m 序列的周期为 $2^5 - 1$ 。

下面以 m 序列发生器模型 $x^5 + x^2 + 1$ 为例，利用 DSP Builder 构建一个伪随机序列发生器。图 9-69 显示了上式的 DSP Builder 模型表述，这里采用相连的延时单元组作为移位寄存器，用异或（XOR）完成模二加运算，输出为 $mout$ 。

但应注意，图 9-69 所示的电路一般无法正常工作，这是由于在 DSP Builder 默认的延时单元在开始工作时，存储内容为 0，而对于 m 序列来说，起始序列为全 0，那么根据多项式，输出序列也将为全 0。全 0 序列不是正常的 m 序列。因此只要起始时，寄存器中有一个为 1， m 序列就可以正常输出。为此，对图 9-69 的模型进行修改，修改后的模型见图 9-70。对图 9-70 的模型进行仿真，可得到一个伪随机序列信号输出，如图 9-71 所示。

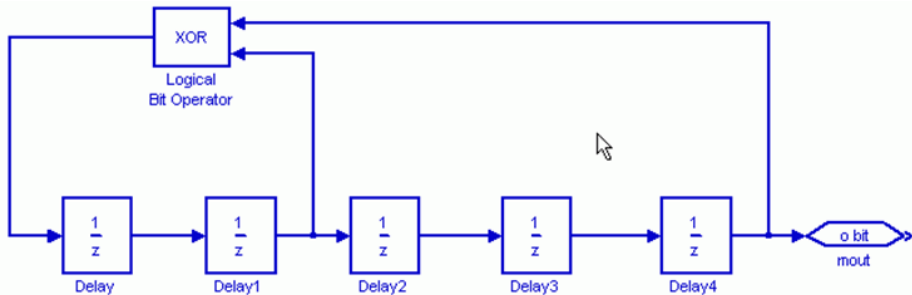


图 9-69 m 序列发生器模型

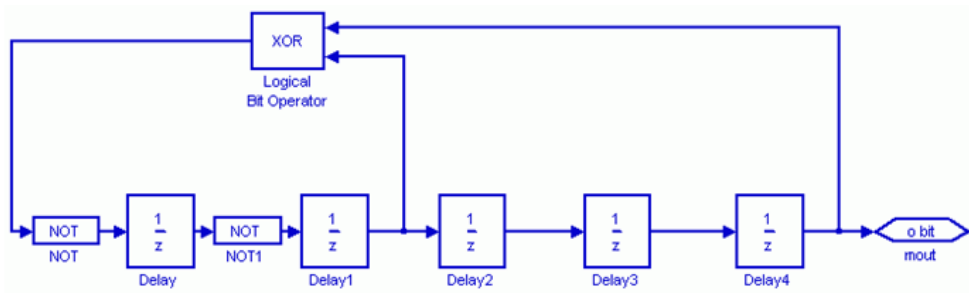


图 9-70 修改后的 m 序列发生器模型

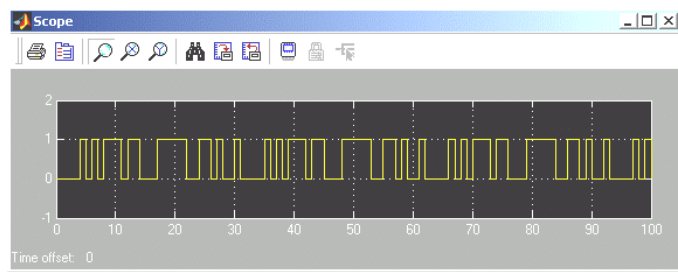


图 9-71 m 序列发生器 Simulink 仿真结果

9.5.2 帧同步检出

在数字通信系统中，同步是非常关键的。由于信号的远距离传输，不可避免存在信号延时、干扰、非线性失真、收发两端的时钟偏差等等。为保证数字传输信号的有效性，必须进行同步。根据同步的作用，可以分为：载波同步、位同步、帧同步、网同步。

本节以帧同步设计为例，介绍 DSP Builder 的设计方案。

数字通信中，信号流的最小单元是码元，若干码元构成一个帧，若干个帧再构成一个复帧，…。在接收端，必须分辨出每个帧的起始和接收，否则，无法正确恢复信息。这种同步，被称为帧同步（又称群同步）。

帧同步可以用多种方法来实现，在此仅列举一种：连贯插入法。即在每一帧的开头连续插入一个特殊码组，比如巴克码。收端检测到该特殊码组的存在，就意味着帧开始了。

巴克码是一个有限长的数字序列。一个 n 位巴克码序列 $\{x_i\}$ ，其中 $1 \leq i \leq n$ ， x_i 取值为 +1 或者 -1，其局部自相关函数满足：

$$R(j) = \sum_{i=1}^{n-j} x_i x_{i+j} = \begin{cases} n, & j = 0 \\ 0, \pm 1, & 0 < j < n \\ 0, & j \geq n \end{cases} \quad 9-7$$

即当 $j = 0$ 时，巴克码的局部自相关函数达到峰值，其它 j 值时， $R(j)$ 在 ± 1 附近波动，可以用作帧同步的特殊码组。符合上述自相关特性的码组是存在的，比如 $\{+1, +1, +1, -1, -1, +1, -1\}$ 就是 7 位巴克码序列。

当 $j = 0$ 时， $R(j) = \sum_{i=1}^7 x_i^2 = 7$ ，达到峰值；

当 $j = 1$ 时， $R(j) = 1$ ；当 $j = 3, 5, 7$ 时， $R(j) = 0$ ；当 $j = 2, 4, 6$ 时， $R(j) = -1$ 。

根据以上讨论的原理，可以给出巴克码的检出模型。关键是，若需要在数字信号流中检出巴克码组，只要检测序列的自相关函数即可。设计方案如下：

在 Simulink 环境中，建立一个 DSP Builder 模型，检出 7 位巴克码，序列为 $\{+1, +1, +1, -1, -1, +1, -1\}$ ，MDL 模型如图 9-72 所示。

由 Shift Taps 模块完成输入序列存储，由 bxp1m、bxn1m 子系统模块完成 $x_i x_{i+j}$ 运算。

7 输入加法器模块完成求和运算。

注意，若要求帧同步输出脉冲没有延时，不能选择参数“Pipeline（流水线）”。

由 Comparator 比较器模块和 Constant 常数模块构成判决电路，Constant 模块的值设为 6，即序列局部自相关函数输出大于 6，就认为检出巴克码了。

对于输入的数字序列值是 0 或者 1，而对于巴克码是 +1 和 -1。我们在这里规定输入数

字信号序列中的 0 对应于巴克码的-1。

据此可以建立两个子系统模块 bxp1m 和 bxn1m ，分别完成 $+1 * x_{i+j}$ 、 $-1 * x_{i+j}$ 。

子系统模块图见图 9-73、图 9-74。图中仅用了一个选择器和几个常数模块就实现了要求的功能。

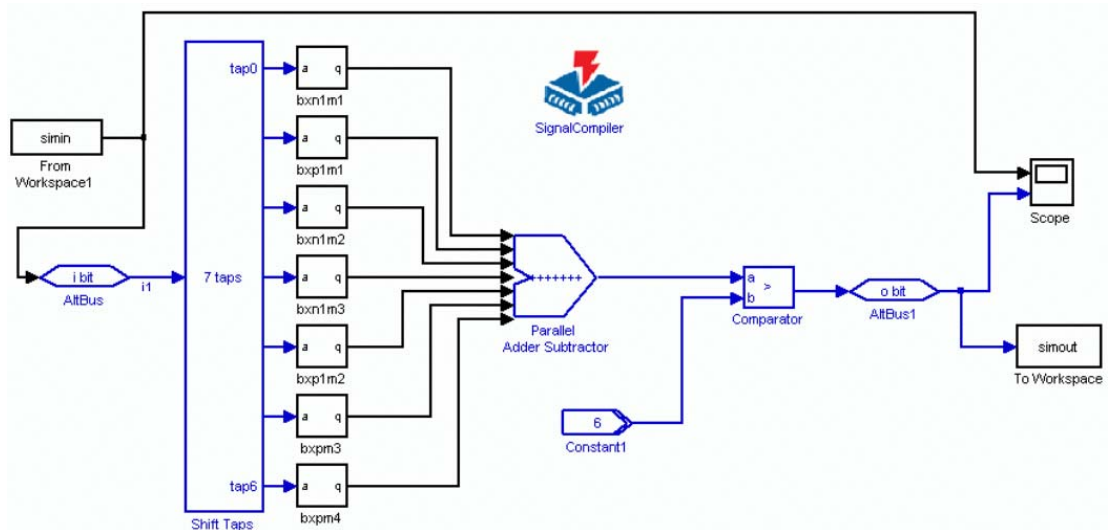


图 9-72 帧同步检出模型

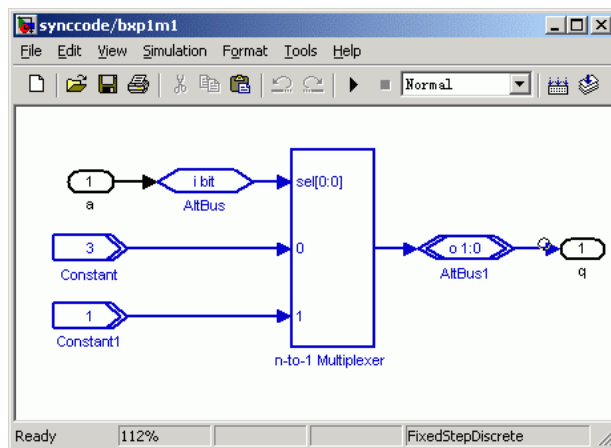


图 9-73 bxp1m 子系统

在 Simulink 中仿真时，通过 From Workspace1 模块从 Matlab 的工作区获得输入序列：

1 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1

仿真结果检到了用于帧同步的巴克码（图 9-75）。

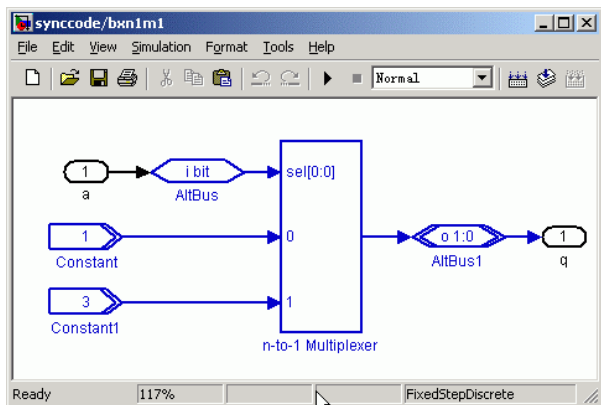


图 9-74 bxnlm 子系统

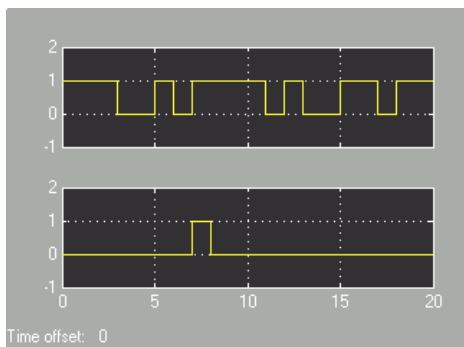


图 9-75 帧同步的巴克码检测仿真结果

9.6 硬件环 HIL 仿真设计

以上提到的在 Simulink 平台上完成的诸多仿真都属于算法级仿真，因为不涉及任何硬件（如 FPGA），因此可认为是软件仿真。这种仿真有几个缺点：1、由于是通过计算完成的，所以速度慢，耗时多，对于较大的设计模型尤为如此；2、仿真结果没有完全反映模块的硬件特性；3、许多仿真内容不易控制。

但是如果完全放在底层设计工具 QuartusII 上仿真，尽管获得了全硬件的仿真结果，但对于许多希望获得的仿真结果比较难以实现，特别是一些特定功能的激励信号难以获得，因为已没有了 Matlab 提供的大量功能强大的仿真工具。

因此最理想的方法是直接在 Simulink 平台上将设计模型下载进 FPGA，利用 Simulink 提供的各类仿真工具进行仿真。

9.6.1 HIL 仿真流程

Altera 的 DSP Builder 提供的 HIL（Hardware in the Loop）模块能很好地完成这种类型的仿真。图 9-76 说明了 HIL 的功能。HIL 可以在 Simulink 模型与 FPGA 开发板间通过 JTAG 通信口建立一种联系，从而实现基于 Matlab/DSP Builder 平台的硬件仿真。

以下说明基于 HIL 模块的仿真流程：

1. 首先完成一个 Simulink 模型设计

图 9-77 所示的电路是一个扫频滤波信号发生器模型，这里以此电路为例。此原例的安装路径是：Altera\DSPBuilder\DesignExamples\Tutorials\HIL\FreqSweep。

本例路径是：E:\myprj\FreqSweep_HIL\freqsweep

图 9-77 中的 Sine Wave Cordic 模块是正弦扫频信号发生器，Low PassFilter 模块是一个 FIR 低通滤波器。图 9-78 是此电路的仿真波形（软件仿真）。

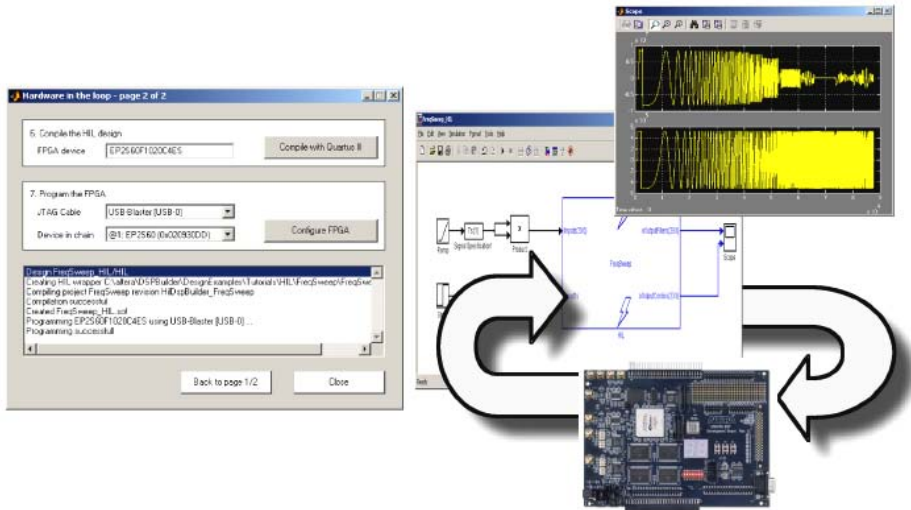


图 9-76 插入 HIL 的 Simulink 模型硬件仿真说明图

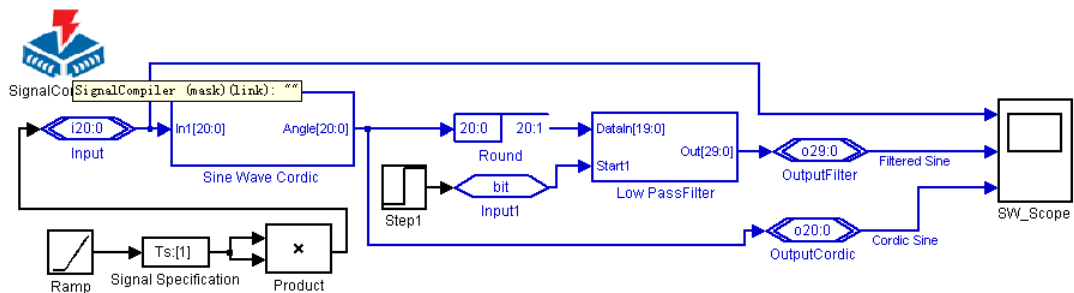


图 9-77 扫频滤波信号发生器 Simulink 模型图，文件名 freqsweep.mdl

2. 通过 DSP Builder 转化成 QuartusII 的工程

如图 9-79 所示，利用 SignalCompiler 对扫频滤波信号发生器进行转换、综合和适配。分别按动此对话框的按钮 1、2、3，使 freqsweep.mdl 转化为一个 QuartusII 工程。关闭窗口图 9-79。

3. 用 HIL 模块取代设计模型的所有电路

如图 9-80 所示，删去图 9-77 中所有 Altera DSP Builder 库元件构成的电路，再引入 HIL 模块。如图 9-81 所示，HIL 模块在 Altera DSP Builder 的 AltLab 库中，用鼠标拖入图 9-80 的设计窗即可。最好将原设计换一名称另存。

4. HIL 模块参数设置

双击图 9-80 的 HIL 模块，将弹出如图 9-82 所示的 HIL 对话框。首先在“1.Select the QuartusII project”栏加入以上生成的工程：

E:\myprj\FreqSweep_HIL\freqsweep\freqsweep.ppf;

在“2. Select the clock pin”栏选择 clock（默认）；根据图 9-77 电路输入输出端口的数据类型，确定“3. Identity the signed ports”栏中各端口的数据类型。

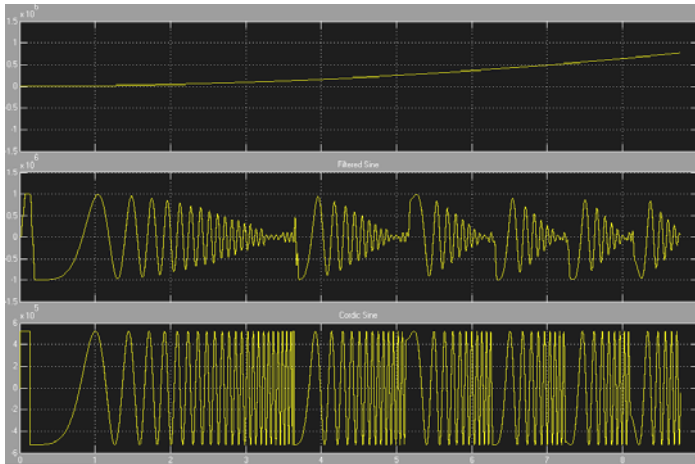


图 9-78 扫频滤波信号发生器算法（软件）仿真波形

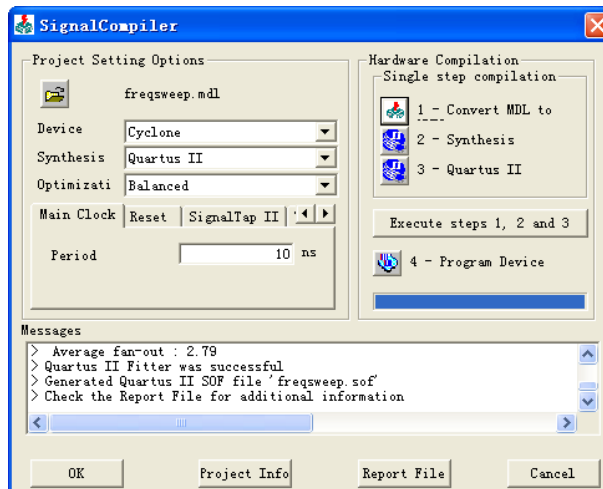


图 9-79 SignalCompiler 对扫频滤波信号发生器进行转换、综合和适配

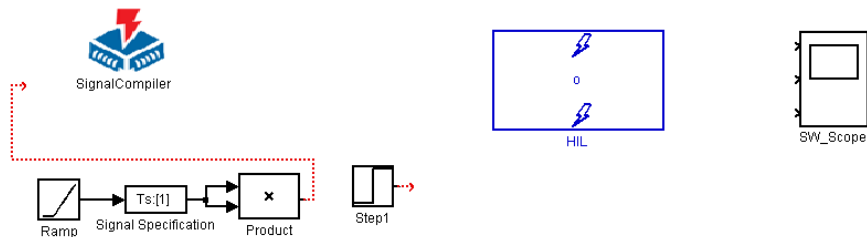


图 9-80 消去原设计，加入 HIL 模块

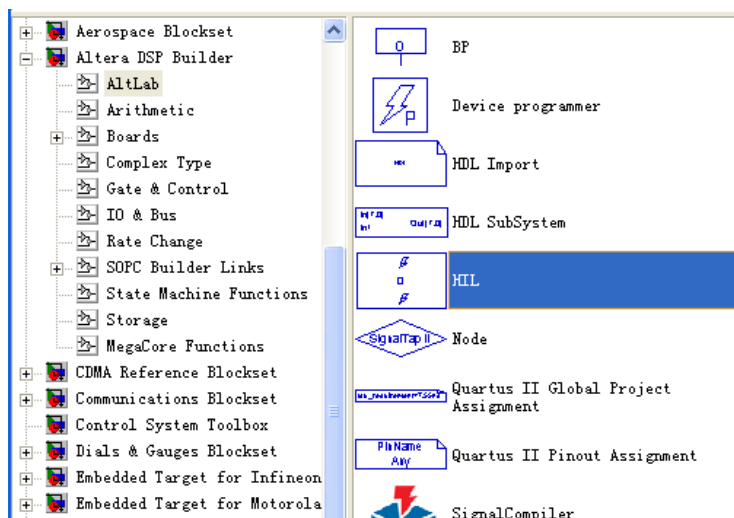


图 9-81 向 Simulink 图中拖入 HIL 模块

在右栏的 Burst Mode 处打勾；在 Burst length 处键入 1024（默认）；在 Frame Mode 处打勾（以下为默认）。按 Next page 按钮，进入下一页（图 9-83）。注意，此前应该用编程线（如 USB-Blaster 等）将 PC 机与 FPGA 开发板连好，并打开电源。

在图 9-83 所示的对话框中，在 FPGA device 栏键入 FPGA 开发板上的 FPGA 的型号，如 EP1C6Q240C8。注意，如果 PC 机与 FPGA 的 JTAG 口通信正常，则在 Program the FPGA 栏的 JTAG Cable 栏将自动显示编程器的类型；而 Device in chain 栏将自动显示已从 FPGA 板上测得的 FPGA 型号。

按“Compile with QuartusII”按钮，进行编译，即将原来图 9-77 已生成的工程与 HIL 模块中输入输出信号缓冲模块一起编译成一个下载文件，以便能进行基于 HIL 的硬件仿真。

如果编译成功，在图 9-83 所示的窗下方的信息栏将出现文字：Created freqsweep_HIL.sof。这就是针对此工程生成的下载文件名是 freqsweep_HIL.sof。

接下去就是下载。

点击图 9-83 的 Configure FPGA 按钮，通过 JTAG 口将此文件下载进 FPGA 中。如果下载无误，在图 9-83 所示的窗下方的信息栏将出现文字：Programming successfull，然后关闭此窗。

5. 进行 HIL 硬件仿真

如图 9-84，将 HIL 模块与外围电路连接好。点击 Simulink 仿真按钮，启动仿真。图 9-85 即为加入了 HIL 模块的扫频滤波电路模型的硬件仿真波形。

显然此波形与图 9-78 的软件仿真波形十分相似。注意，加入 HIL 模块的电路模型只能用于硬件仿真，不能利用 SignalCompiler 变成 VHDL。

另外，也可以利用图 9-81 所示的窗口中的 Device Programmer 模块进行编程。图 9-86 就是使用此模块编程的示例。下载流程同上。

注意，有 HIL 模块的硬件仿真过程中，编程通信线必须始终与 FPGA 连接无误。

对于 GW48EDA 系统，可以任意选择 ByterBlasterMV、ByterBlasterII 或 USB-Blaster 编程线。

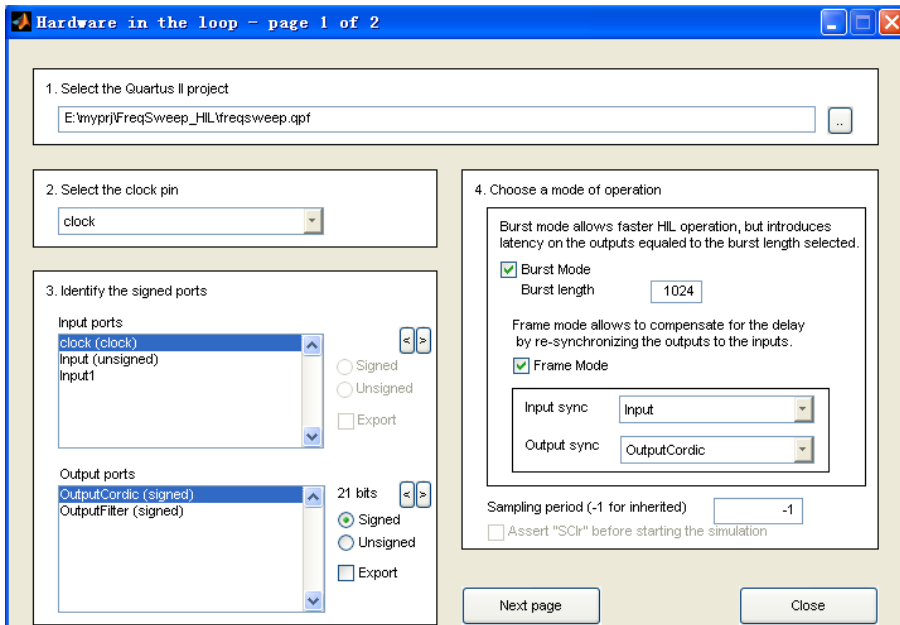


图 9-82 HIL 模块工程加载与参数设置窗

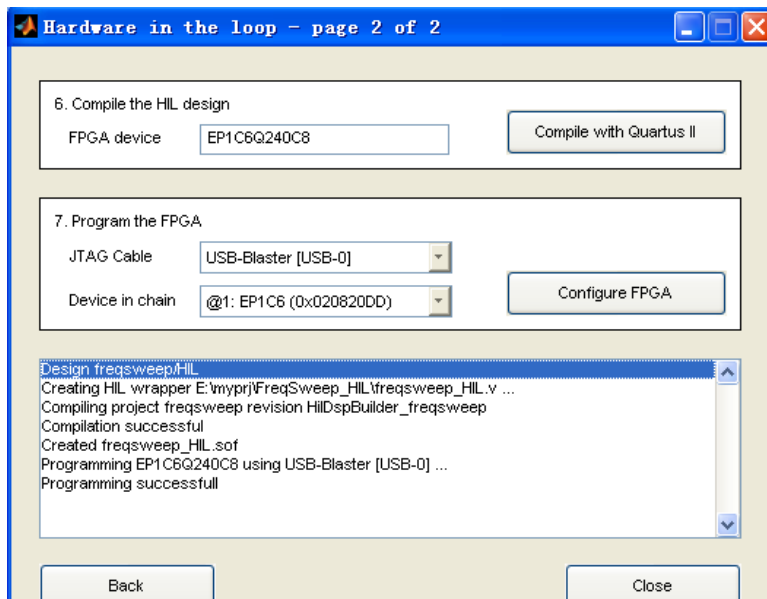


图 9-83 HIL 模块编译与编程窗

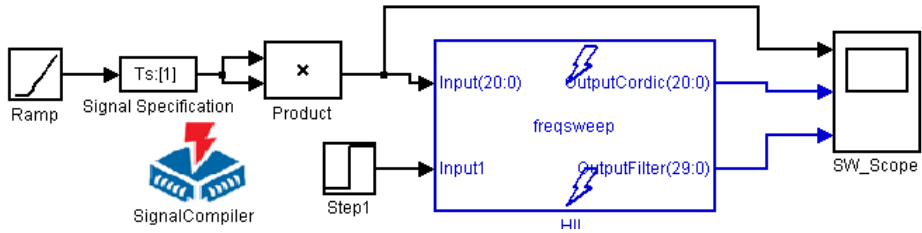


图 9-84 加入了 HIL 模块的扫频滤波电路模型

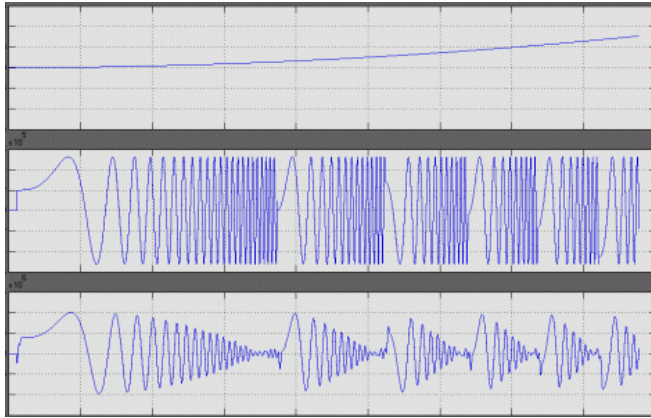


图 9-85 加入了 HIL 模块的扫频滤波电路模型的硬件仿真波形

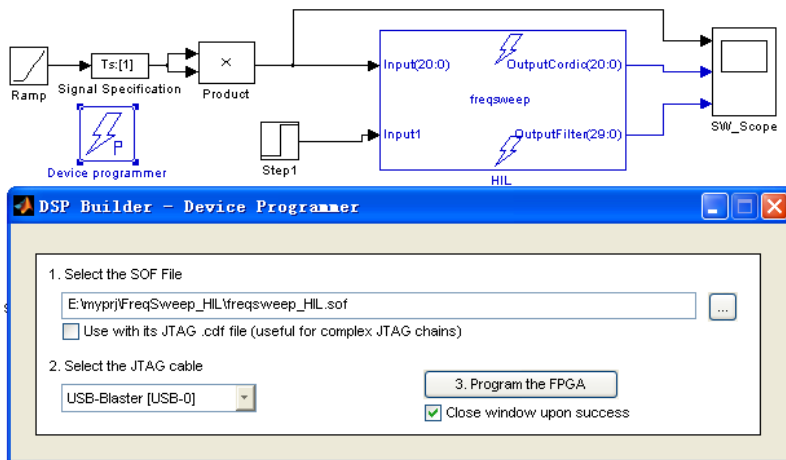


图 9-86 利用专用编程模块向 FPGA 下载

9.6.2 FSK 的 HIL 仿真

图 9-87 是图 9-61 的 HIL 仿真电路。其中的方波发生器参数设定窗如图 9-88 所示。FSK 模型仿真参数设置窗如图 9-89 所示。

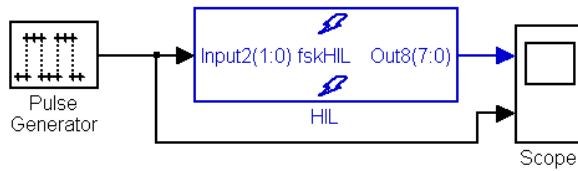


图 9-87 加入了 HIL 模块的 FSK 模型

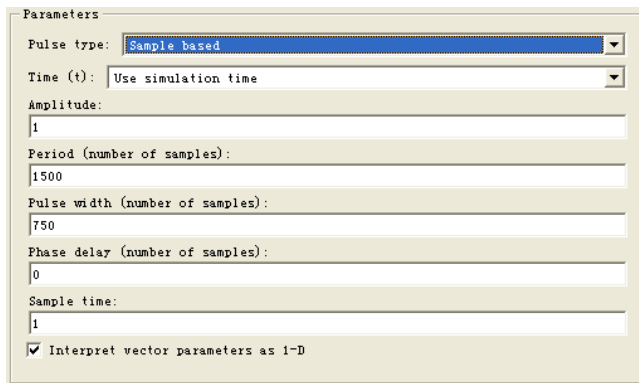


图 9-88 加入了 HIL 模块的 FSK 模型中方波信号参数设置窗

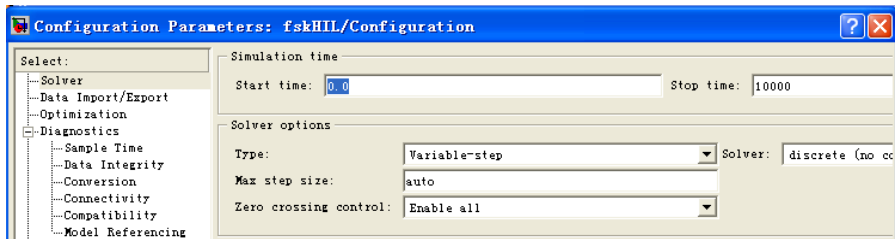


图 9-89 加入了 HIL 模块的 FSK 模型中仿真参数设置窗



图 9-90 加入了 HIL 模块的 FSK 模型仿真波形图

下载后，图 9-87 的仿真波形如 9-90 所示。由此图可见，输入波形与输出波形有一定相位差，这个相位差说明信号在 FPGA 中有了延时。而图 9-62 中是看不到这种延时的。

9.7 DSP Builder 的状态机设计

在 Simulink 图形编辑窗中也允许利用状态机模块设计 DSP 系统中必须的控制电路。状态机模块是由状态机表格模块构成的，它允许设计者使用状态转换表格来构建一个 1 位热码编码方式的 Moore 状态机。此表格模块经 SignalCompiler 转换后可以变成 VHDL 的状态机文本描述。

9.7.1 FIFO 控制状态机设计示例

作为示例，在此使用双口 RAM 模块 (Dual-Port RAM) 和状态机模块 (State Machine Table) 完成一个 FIFO 存储器的设计。这两个模块可以分别从 Simulink Library Browser 的 Altera DSP Builder 库中的 State Machine Functions 子目录和 Storage, 及 Arithmetic 子目录中获得。构成的电路及其仿真配置图如图 9-91 所示。

其中的双口 RAM 的地址线宽是 8 位，端口 rdad 和 wrad 分别是 8 位数据读地址和写地址，d 和 q 分别是 8 位输入和输出数据线；总线模块为有符号类型；wren 是写入允许信号。

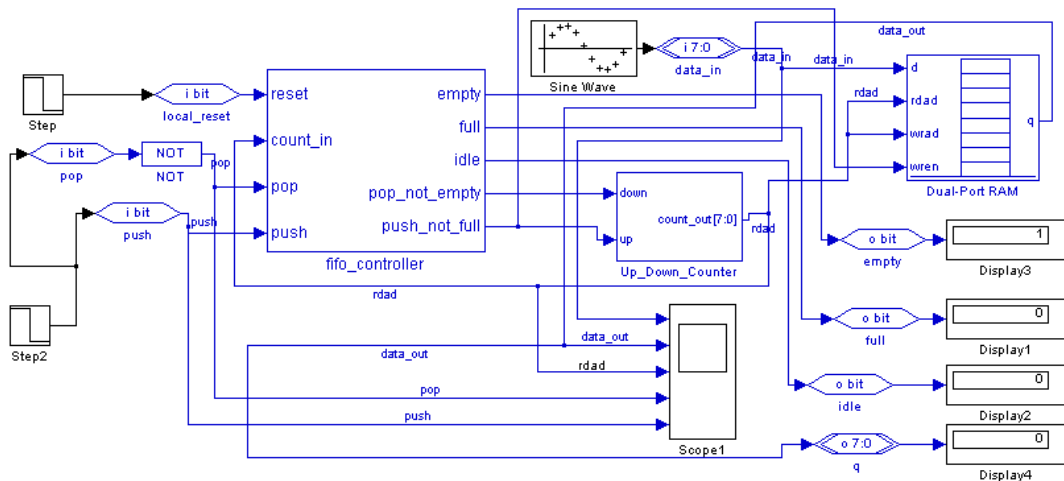


图 9-91 由状态机模块和双口 RAM 构成的 FIFO 存储器电路 fifo_control.mdl

图 9-91 中的加减计数器模块 Up_Down_Counter 内部电路如图 9-92 所示。其中的加减计数器都是 8 位无符号整数总线类型计数器，并行加减法器的操作符是“+”。

状态转换模块或表格模块编辑界面如图 9-93 所示。来自库中的默认的状态机表格符号如图 9-94 所示。默认的状态机有 5 个输入端和 5 个状态。每一状态以一个输出信号端来代表。当状态机运行时，如果相应的状态等于当前状态，则此输出为电平 1，所有其它输出为 0。在 Simulink 中，输入和输出是有符号整数表示的，而在 VHDL 中，输入和输出是由标准逻辑位矢量表示的，注意其中的对应关系。

以下将叙述在 DSP Builder 中状态机的设计流程。在此以 FIFO 控制状态机设计 fifo_control.mdl 为例。此项设计中包含了一个用于 FIFO 进行逻辑控制的简单的状态机。

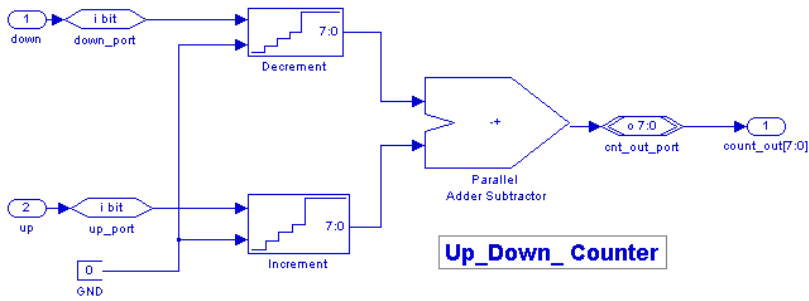


图 9-92 加减计数器模块 Up_Down_Counter 内部电路

此状态机向一双口 RAM 的输入端以及地址计数器的输入端馈入控制信号，其操作步骤和流程如下：

1. 当 FIFO 的数据压入信号（push）一出现，且此时地址计数器的计数值小于 250，地址计数器就递增 1，于是一个字节的数据就被压进（写入）到 RAM 中去了；
2. 当 FIFO 的数据弹出信号（pop）一出现，且地址计数值大于 0，地址计数器就递减 1，于是一个字节的数据就被弹出（被读出 RAM）；
3. 当地址计数器的值等于 0 时，FIFO 数据空的标志信号（empty）出现；
4. 当地址计数器的值等于 250 时，则 FIFO 数据已满的标志信号（full）出现。

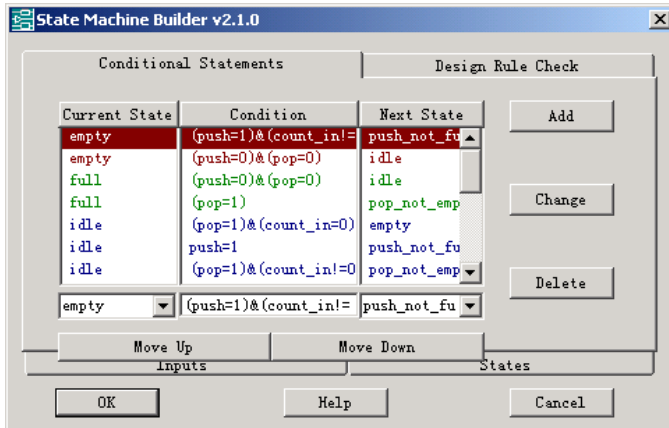


图 9-93 状态机转换表图

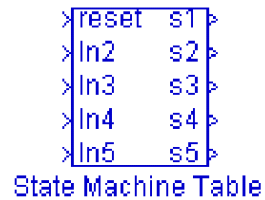


图 9-94 库中默认状态机表模块

表 9-1 是 FIFO 控制器的状态转换表。

表 9-1

当前状态	条件	次态
empty	(push = 1) & (count_in != 250)	push_not_full
empty	(push = 0) & (pop = 0)	idle
full	(push = 0) & (pop = 0)	idle
full	(pop = 1)	pop_not_empty
idle	(pop = 1) & (count_in = 0)	empty

idle	push =1	push_not_full
idle	(pop =1) & (count_in! = 0)	pop_not_empty
idle	(push =1) & (count_in=250)	full
pop_not_empty	(push =0) & (pop=0)	idle
pop_not_empty	(pop =1) & (count_in = 0)	empty
pop_not_empty	(push =1) & (count_in!=250)	push_not_full
pop_not_empty	(pop =1) & (count_in! = 0)	pop_not_empty
pop_not_empty	(push =1) & (count_in=250)	full
push_not_full	(push =0) & (pop=0)	idle
push_not_full	(push =1) & (count_in=0)	empty
push_not_full	(push =1) & (count_in!=250)	push_not_full
push_not_full	(push =1) & (count_in=250)	full
push_not_full	(pop =1) & (count_in! = 0)	pop_not_empty

9.7.2 状态机设计流程

下面以上述 FIFO 控制器状态机为例，说明利用状态机表格来设计状态机的步骤：

1. 加入状态机表格模块

在 Simulink 设计中加入状态机表格模块，并为此模块取一个名。图 9-95 即为此模块。此例中，模块的名是 fifo-controller。图 9-95 是改名 fifo_controller 后的状态机模块。注意，必须将状态机表格模块的默认名改成其它选定的名，然后才能定义状态机的特性！

2. 设定状态机的特性

双击该表格模块，以便能设定状态机的特性。当出现“State Machine Builder”对话框后，选择“Inputs”页。“Inputs”页将显示出用于定义状态机的输入名称，并展示出一个允许设计者向其中加入，改变和消去输入名的界面。如图 9-96 所示是在定义了 FIFO 控制状态机后的“Inputs”页。定义的输入信号是 reset（默认）、count_in、pop、push。

```

>reset s1
>ln2 s2
>ln3 s3
>ln4 s4
>ln5 s5
fifo_controller

```

图 9-95 状态机模块图

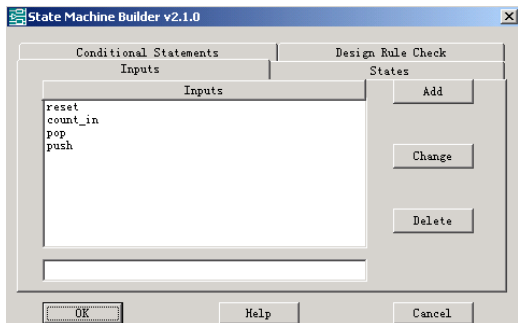


图 9-96 状态机表的“Inputs”页

3. 设定状态名

单击“State”页。此页将显示用于定义状态机的状态名，并给出允许设计者加入，修改和删除状态名的界面。“State”页上也可选择状态机的复位状态。复位状态即为当复位输入信号到来时，状态机即向此状态转换。定义的状态有5个：empty、full、idle、pop_not_empty、push_not_full。注意，状态机的状态至少应有两个状态，图9-97是“State Machine Builder”的“State”页，已作了对FIFO控制状态机的状态特性编辑。

4、设定状态机工作方式

在设定好了输入信号与状态名后，点击“Conditional Statements”页，准备描述状态机的工作方式（如图9-93所示）。“Conditional Statements”页显示的是状态转换表，表中含有状态机的条件描述。其中条件描述由3部份组成：

- ① 当前状态；
- ② 导致转换方式的条件；
- ③ 状态机转换的次态。

当前态和次态状态值必须是在“State”页中定义的状态名。表9-2中是用于定义条件描述的条件操作符的优先级别。注意，在“Conditional Statements”页中至少应定义一个条件描述。此外，如果希望状态机能根据某个条件无条件地从一种状态向另一种状态转换，那么这个条件就可用1来表述。

表9-2

比较操作符	说明	优先级	示例
- unary)	负	1	-1
(...)	括号	1	(1)
=	数值相等	2	in1=5
!=	不等于	2	in1!=5
>	大于	2	in1>in2
>=	大于等于	2	in1>=in2
<	小于	2	in1<in2
<=	小于等于	2	in1<=in2
&	与	2	(in1=in2)&(in3>=4)
ξ	或	2	(in1=in2) ξ (in1>=in2)

图9-93所示的即为“Conditional Statements”页。在上页已定义了FIFO控制器的条件描述。表中的条件描述一旦确定，状态机的工作方式就确定了。一般，当一状态机处于某一特定状态时，就会对次状态的条件进行计算判断，以便确定下次转换到什么状态上去。对于条件叙述表中所列的条件，状态机是顺序计算判断的。例如如表9-3所示。

表9-3

当前状	条件	次态
Idle	(pop =1) & (count_in = 0)	empty
Idle	push =1	push_not_full

Idle	(pop =1) & (count_in!= 0)	pop_not_empty
Idle	(push =1) & (count_in=250)	full

表 9-3 描述了当状态机处于“idle”状态时的各种态状转换条件。由于条件 (pop =1) & (count_in!= 0) 在表中的位置高于条件 (push =1) & (count_in=250)，所以，前者就有更高优先权。当它们同时满足时，将首先依前者的条件进行转换。

条件 (pop =1)&(count_in!=0) 仅次于最高优先权的条件描述，而条件 (push =1) & (count_in=250) 所处的位置具有最低的优先权。

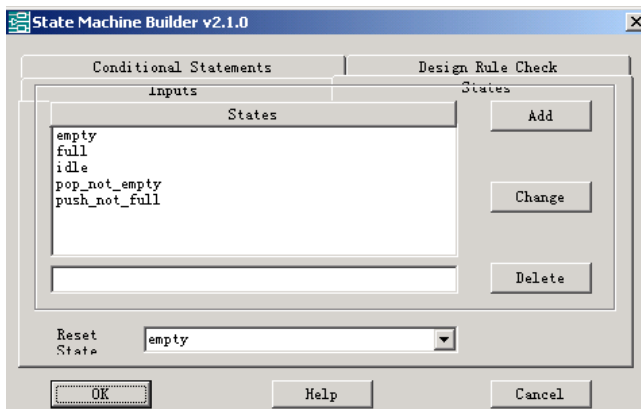


图 9-97 状态机表的“States”页

例 9-1 就是表 9-3 编译转换而来的 VHDL 描述。其中的 _sig 是在 VHDL 文件中另加的。

【例9-1】

```

IF ((pop_sig=1) AND (conut_in_sig=0)) THEN
    next_state <= empty_st;
ELSIF (push_sig=1) THEN
    next_state <= push_not_full_st ;
ELSIF (pop_sig=1) AND (conut_in_sig /=0)) THEN
    next_state <= pop_not_empty_st ;
ELSIF (push_sig=1) AND (conut_in_sig =250)) THEN
    next_state <= full_st ;
ELSE
    next_state <= idle_st ;
END IF ;

```

也可以通过使用“Move Up”和“Move Down”按钮来改变条件描述句的上下位置。例如，如果将表 9-3 改变后，如表 9-4 所示。

表 9-4

当前状	条件	次态
Idle	(pop =1) & (count_in = 0)	empty

Idle	(push =1) & (count_in=250)	full
Idle	(pop =1) & (count_in! = 0)	pop_not_empty
Idle	push=1	push_not_full

那么 State Machine Builder 将此表翻译成如例 9-2 所示的 VHDL 程序。

【例9-2】

```

IF ((pop_sig=1) AND (conut_in_sig=0)) THEN
    next_state <= empty_st ;
ELSIF (pop_sig=1) AND (conut_in_sig / =0)) THEN
    next_state <= pop_not_empty_st ;
ELSIF (push_sig=1) AND (conut_in_sig =250)) THEN
    next_state <= full_st ;
ELSIF (push_sig=1) THEN
    next_state <= push_not_full_st ;
ELSE
    next_state <= idle_st ;
END IF ;

```

5、点击“Design Rule Check”页面，以便确定前面各步骤中所定义的状态机没有违反任何设计规则，然后点击“Analyze”，对状态机中所设的条件进行计算判别，以确定是否存在一般错误或逻辑错误。如果有错，将在“Analyze Results”栏中以红色给出错误信息。

图 9-98 所示即为点击“Analyze”后的“Design Rule Check”页面。如果发现在信息栏的分析结果中有报错，应该找出错误所在，重新启动检测分析的操作，直至排除所有错误。最后就是系统功能仿真，通过后，就可以将其转换成 VHDL 代码描述了。

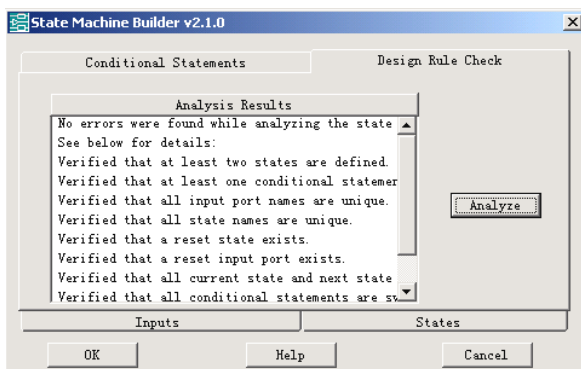


图 9-98 State Mahine Builder Design Rule Check 页面

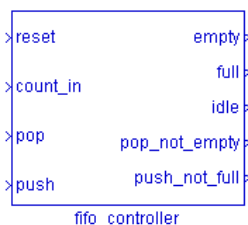


图 9-99 设定了状态机后的表格模块

6、点击“OK”，将所有更改后的内容存盘。这时就关闭了 State Machine Builder，并将回到 Simulink 设计文件。这时的设计文件将自动更新在前面各步骤中定义的输入输出名称。图 9-99 是 FIFO 设计实例更新后的 State Machine Table 模块。

7、最后，将此 State Machine Table 模块连接到总体设计文件的部分。

启动仿真后就能得到对图 9-91 电路测试的如图 9-100 所示的波形。对波形的说明如下：

图 9-100 所示的波形中，从上到下的波形分别为：data_in, data_out, rdad, pop, push。
data_in 是状态机的输入数据，有符号的正弦信号波形数据；data_out 是 RAM 的控制输出，前半段，在地址信号 rdad 和 push 信号的作用下在向双口 RAM 压如波形数据的同时，RAM 通过 data_out 也向外输出数据；从 rdad 波形可见，当地址增加到最大后便递减下降，这是便将 push 进的数据 pop 出来，由 data_out 波形的对称性可以说明这一点。

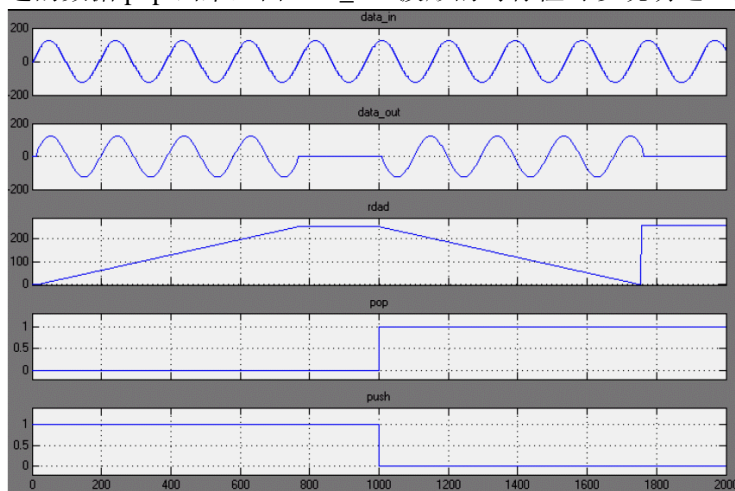


图 9-100 仿真波形

此外，当分别给定仿真参数后，图 9-91 的电路在数据显示器上给出如下数据，显然满足状态机的条件设置。

```

stop time = 400 ;empty=0; full=0; idle=0    data_out=48
stop time =900 ;empty=0; full=1; idle=0    data_out=
stop time = 2000 ;empty=1; full=0; idle=0  data_out=48
stop time = 400 ;empty=0; full=0; idle=0    data_out=48

```

习 题

- 9-1 说明 Matlab、DSP Builder 和 QuartusII 间的关系，给出 DSP Builder 设计流程。
- 9-2 把图 9-2 设计模型通过 SignalCompiler 转化为 VHDL 文件，并用 ModelSim 进行功能仿真。
- 9-3 DSP Builder 子系统模块与 Simulink 的 SubSystem 是什么关系，对于可以用 SignalCompiler 编译的 DSP Builder 子系统在 SubSystem 的基础上还需要什么设置？
- 9-4 在手动流程中能完成哪几个层次的仿真，各有什么作用？
- 9-5 简述 DDS 的实现原理。
- 9-6 分别说明 DDS 的输入信号与输出正弦信号的关系，分析 DDS 带来的误差问题。

实验与设计

实验 9-1. 利用 Matlab/DSP Builder 设计基本电路模块实验

(1) 实验目的: 熟悉利用 Matlab 和 DSP Builder 进行基本电路模型设计方法, 包括手动流程和自动流程, 学习不同的硬件设计仿真技术。

(2) 实验任务 1: 首先按照以上给出的详细步骤, 利用 Matlab, DSP Builder 等工具分别完成:

1. 电路图 9-21 的正弦信号发生器 (有符号数据类型);
2. 电路图 9-21 的正弦信号发生器 (无符号数据类型);
3. 电路图 9-26 的正弦信号发生器;
4. 电路图 9-52 的信号发生器;
5. 电路图 9-91 的状态机控制器的设计、仿真测试、硬件实现和硬件实时测试。包括 Matlab 建模、系统仿真、SignalCompiler 转换、功能仿真、QuartusII 适配、时序仿真、编程下载、SignalTapII 测试等。对于 SinLUT 模块, 分别完成选中 LPM 和不选 LPM 的设计, 比较它们的性能和资源利用情况。

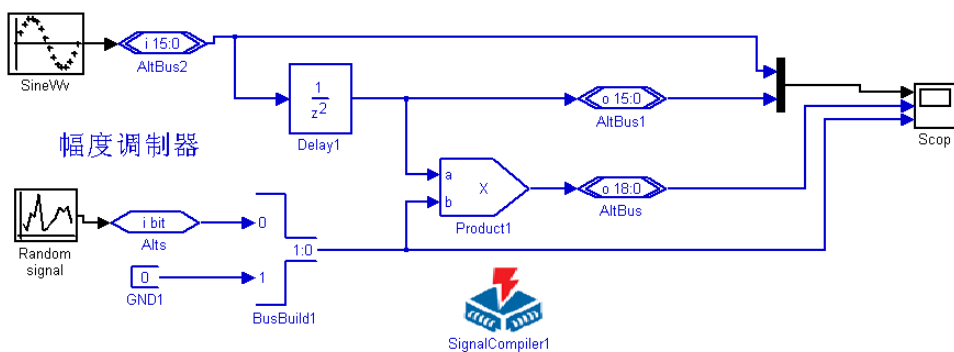


图 9-101 正弦调制信号模型

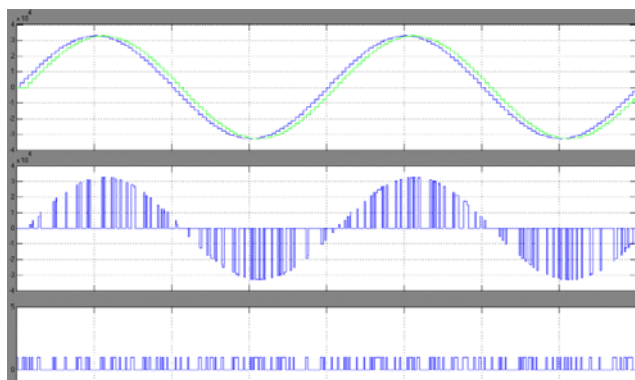


图 9-102 正弦调制信号仿真波形

- (3) 实验任务 2: 设计一个幅度调制器模型, 电路模型如图 9-101 所示, 两个输入信号源分别是正弦

信号发生器和随机信号发生器。图 9-102 是输出波形。

(4) **实验报告**: 根据以上的实验内容写出实验报告, 包括设计原理、模型设计、各层次仿真分析结果、硬件测试和详细实验过程记录。

实验 9-2 基于 DSP Builder 的 DDS 应用模型设计

(1) **实验目的**: 掌握利用 Matlab/Simulink、DSP Builder 和 QuartusII 设计 DDS、FSK、PSK、移相信号发生器等模块或系统的设计和硬件实现技术。

(2) **实验任务 1**: 参照 9.4 节给出的模型和设计方法, 给出 DDS 模型的完整设计, 包括 Matlab 建模、仿真、FPGA 硬件实现和硬件测试, 推荐使用 GW48 系统的 Cyclone 系列 FPGA 和 10 位超高速 D/A, 因而可以使用 SignalTapII 进行实时测试。

具体电路也可参考图 9-103 的电路, 该电路模块全部采用无符号整数数据类型。其中相移输入字 Pword 和频率输入字 Fword 都是 8 位, D/A 为 10 位。TRAGout 是未作移相的地址信号, 是 10 位锯齿信号输出, 因此可作为正弦信号移相输出的参考信号。SINLUT 模块中的函数设置如图 9-103 下方所示。

如果选择电路模式 1, 可用键 2, 键 1 输入 8 位频率字 Fword, 键 4, 键 3 输入 8 位相位字 Pword, 时钟接 Clock0; 键 8 控制全局清 0 端; TRAGout 和 ddsout 分别通过两个 10 位 D/A 输出波形。

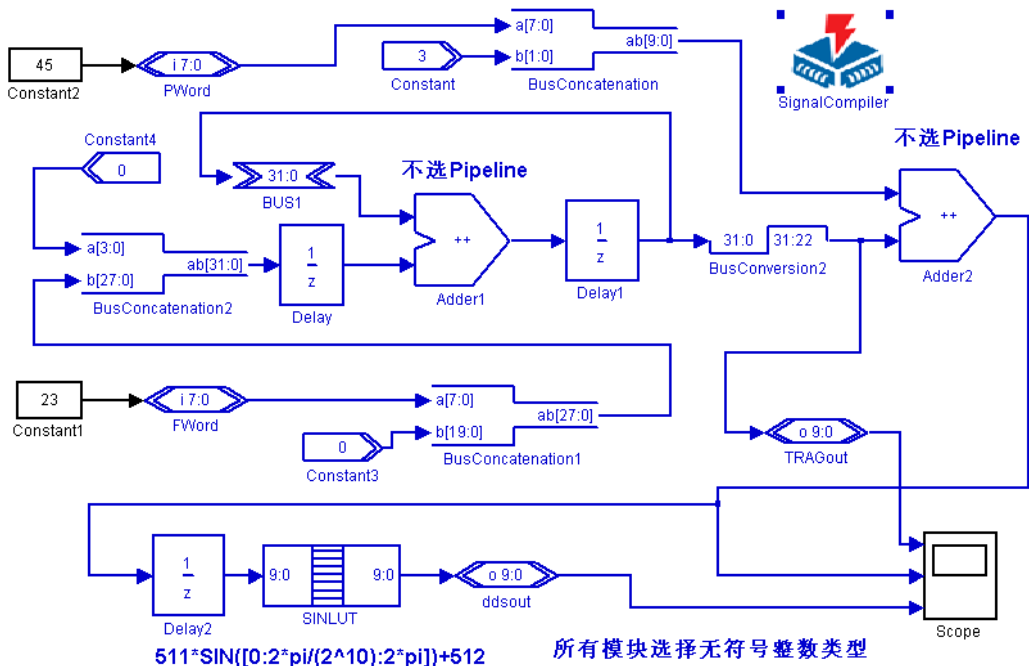


图 9-103 端口数据类型和位数变换后的 DDS 模型

(3) **实验任务 2**: 参照相关章节给出的模型, 完成数控正交信号发生器的设计, 最后在 GW48 上的双高速 D/A 系统上验证, 用双踪示波器验证输出信号的正交关系。

(4) **实验任务 3**: 参照相关章节的设计模型, 设计一个载波抑制双边带调幅电路。

实验提示: 在幅度控制信号输入端加入待调制信号, 注意待调制信号没有直流分量。

(5) 实验任务 4: 利用 DDS 模型, 设计一个普通调幅(AM)电路。已知 FPGA 开发板上时钟为 50MHz, 载波为 1.5MHz, 待调制信号为 50Hz~8kHz, 调制度为 30%。(提示, 在实验任务 4 的基础上, 待调制信号加上直流偏置)。

(6) 实验任务 5: 参照相关章节的模型和设计原理, 给出 2FSK 调制器的完整设计。

(7) 实验任务 6: 利用 DDS 模型来实现 2ASK 功能。

(8) 实验任务 7: 利用 DDS 模型设计一个 2PSK 调制器。

(9) 实验任务 8: 将图 9-103 稍加修改即可得到图 9-104 所示的数字移相信号发生器, 此模型可以直接在 GW48EDA 系统上实现。即在原 TRAGout 输出口处增加了 10 位地址线 10 位数据线的正弦信号数据 ROM, SINLUT1, 通过它, 输出作为基准信号的正弦波。把两路正弦信号接入示波器, 可以看到随相位字 Pword 输入的改变而改变的李萨如图形。

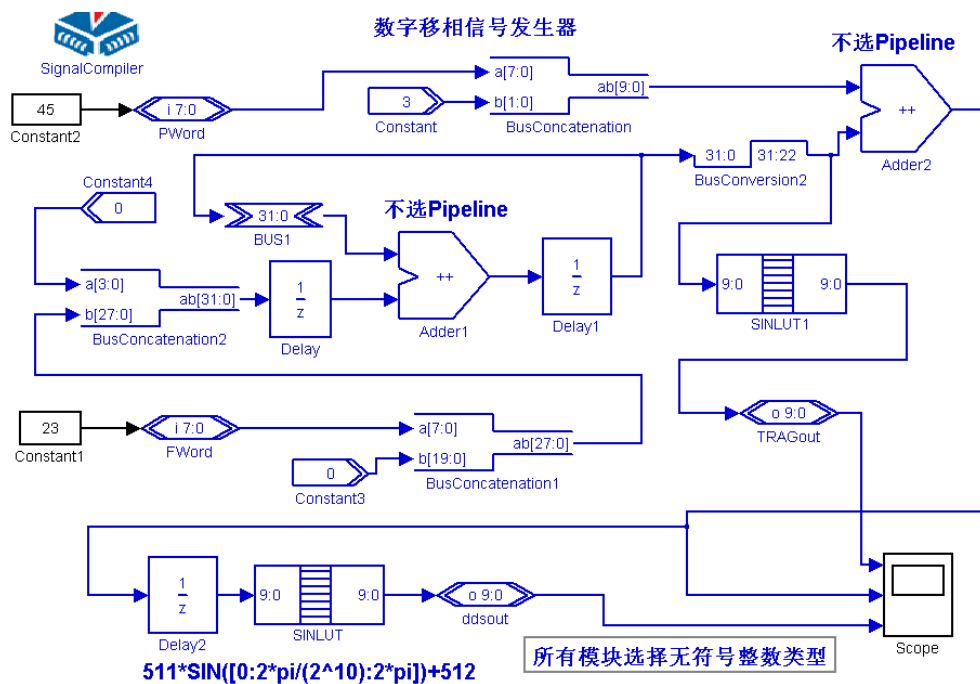


图 9-104 数字移相信号发生器

为了在实验台上容易演示, 相位字和频率字都选择了 8 位, 其余位置 0。在 GW48 系统上可选择模式 1, 这样使键 2/键 1 控制输入 8 位频率字, 键 4/键 3 控制输入相位字, 键 8 控制复位, 时钟选择 clock0, 接 50MHz 或 12MHz。信号输出由两个 10 位的超高速 D/A 担任。如果希望获得更好的控制和更高的输出精度, 必须通过单片机来给出完整的频率字和相位字。

(10) 实验任务 9: 为了提高频带的利用率, QAM (正交载波调制) 在数字通信中应用广泛, 试用 DDS 设计一个 16QAM 调制器, 电路模型可以参考图 9-105。

(11) 实验任务 10: 利用 Altera 公司提供的 DDS IP Core 称为 NCO Compiler, 试用 NCO Compiler 设计 2FSK 调制电路。

(12) 思考题 1: 根据图 9-103, 给出该 DDS 模型的最小频率步进值 (按 32 位频率字计算) 和相位步

进值（按 10 位相位字计算）；说明 Delay2 模块的作用。

(13) 思考题 2：根据图 9-103，说明如果要保证 SINLUT 输出的每周期波形的点数不少于 16 点，则 delay 模块（Fword）的输入值不应该大于（或小于）多少。

(14) 实验报告：根据以上要求和实验内容，记录并分析所有实验结果，完成实验报告

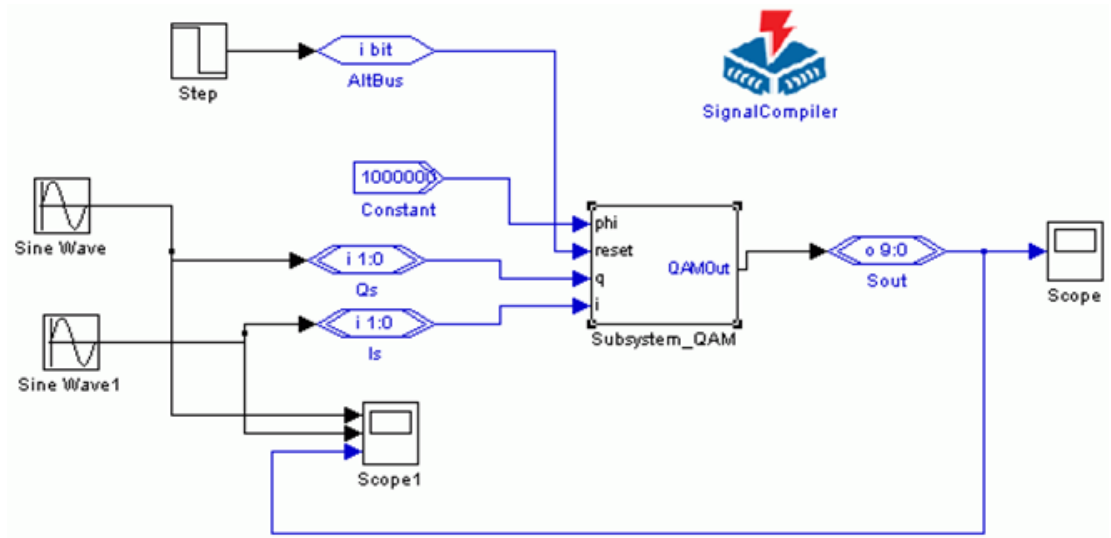


图 9-105 QAM 模型

实验 9-3 编译码器设计实验

- (1) 实验目的：掌握利用 Matlab/Simulink、DSP Builder 和 QuartusII 设计和实现编译码器的技术。
- (2) 实验任务 1：参照相关章节，设计一个 m 序列发生器。
- (3) 实验任务 2：参照相关章节，设计一个巴克码检出模型。
- (4) 实验报告：根据以上要求和实验内容，记录并分析所有实验结果，完成实验报告。

实验 9-4. HIL 硬件环仿真实验

- (1) 实验目的：熟悉利用 HIL 模块进行硬件仿真的技术。
- (2) 实验任务：在 GW48EDA 系统上，利用 HIL 硬件环模块，分别对图 9-59，图 9-61，图 9-64 和图 9-66 的 DDS 模型，FSK 模型，数字移相信号发生器和 AM 发生器模型进行硬件仿真。