

# eXCite 软件使用简单指南

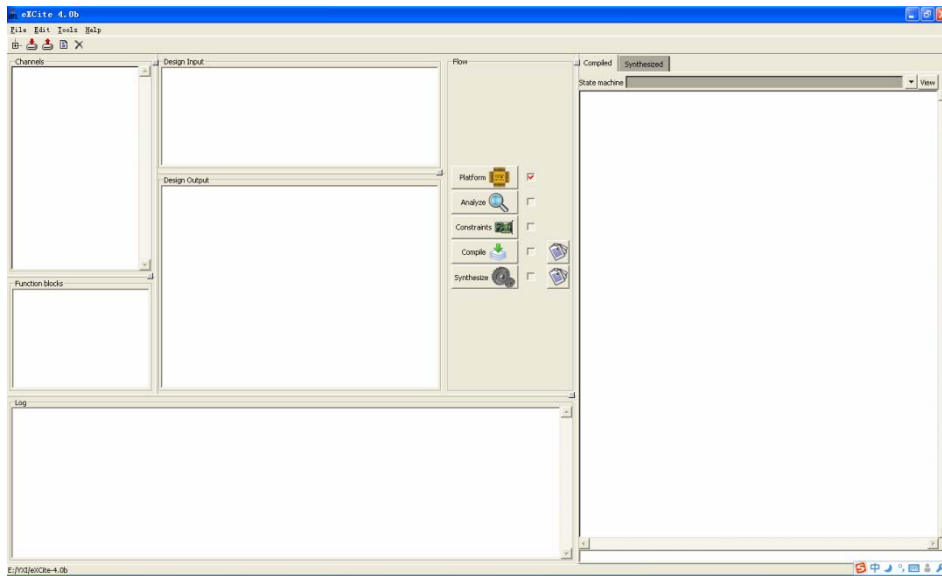
## 一 关于软件的 license

安装后第一次打开 eXCite 文件名的可执行文件

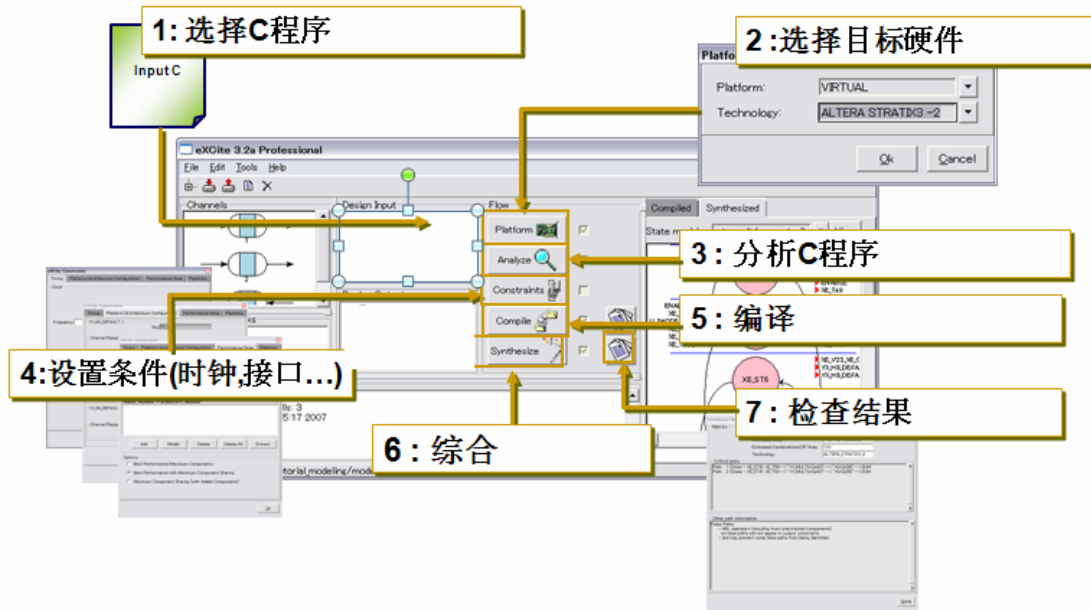
- ◆ 对于浮动的 license，将会出现需要 license 的对话框，选择 license Server，输入合法的 license 服务器地址，即可进入软件。
- ◆ 对于固定 license，将获得的 license 文件重命名为 license.dat，拷贝到 eXCite 的安装目录下即可。
- ◆ 如有疑问，请联系分销商或者我们。

## 二 软件的使用

软件界面如下：

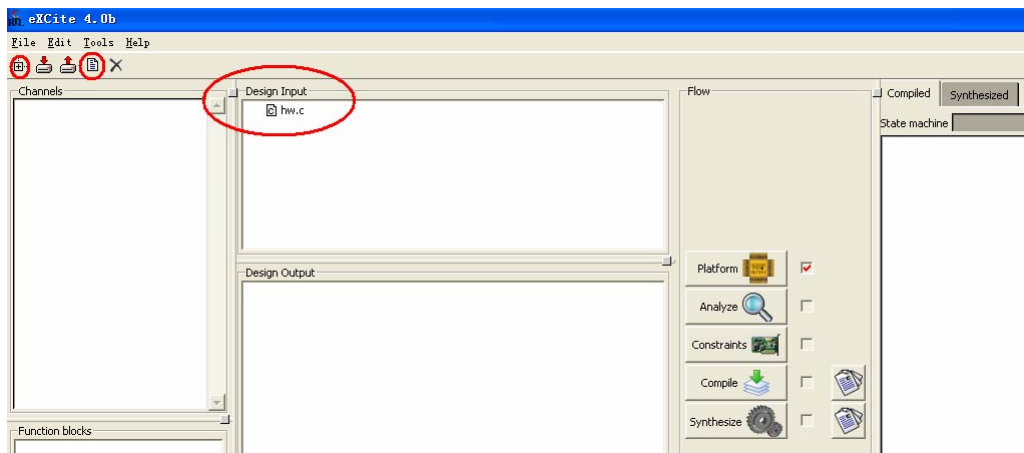


软件界面说明：

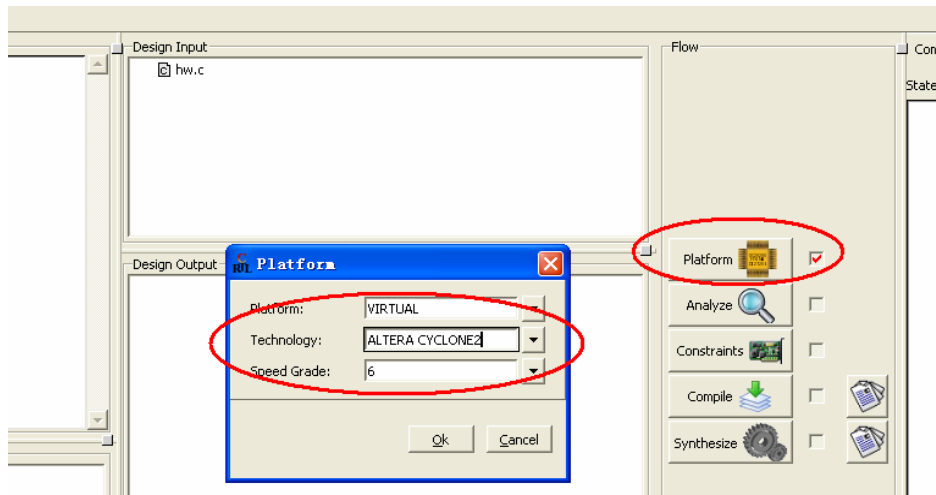


下面以 8 点 FFT 的实例进行使用说明：

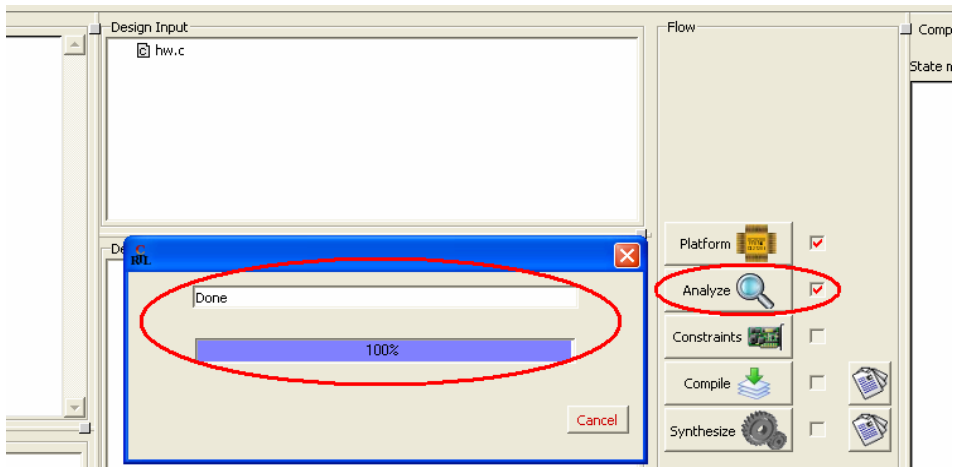
在 example 文件夹中有 hw.c 文件，将其添加进软件中准备转换：



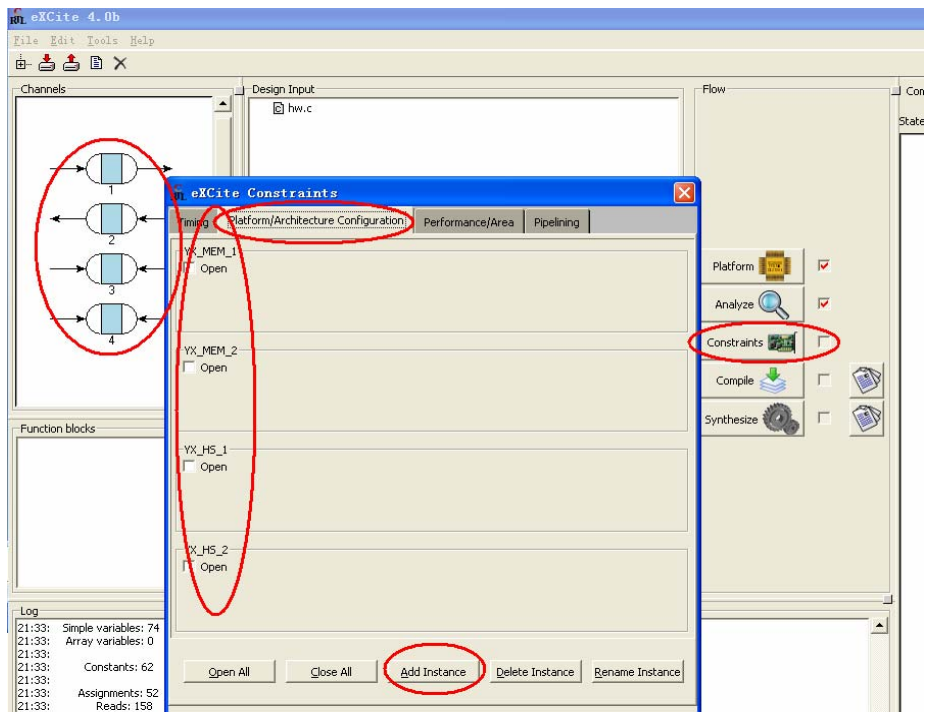
选择合适的硬件平台“Platform”，比如 ALTERA CYCLONE2 速度级别为 6：



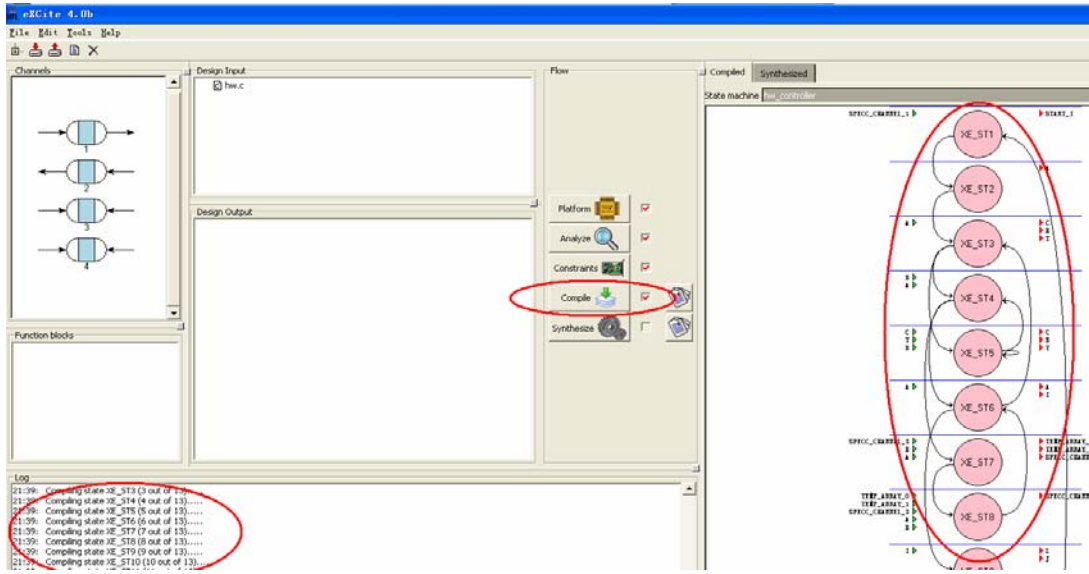
对 hw.c 文件进行代码分析：“Analyze”



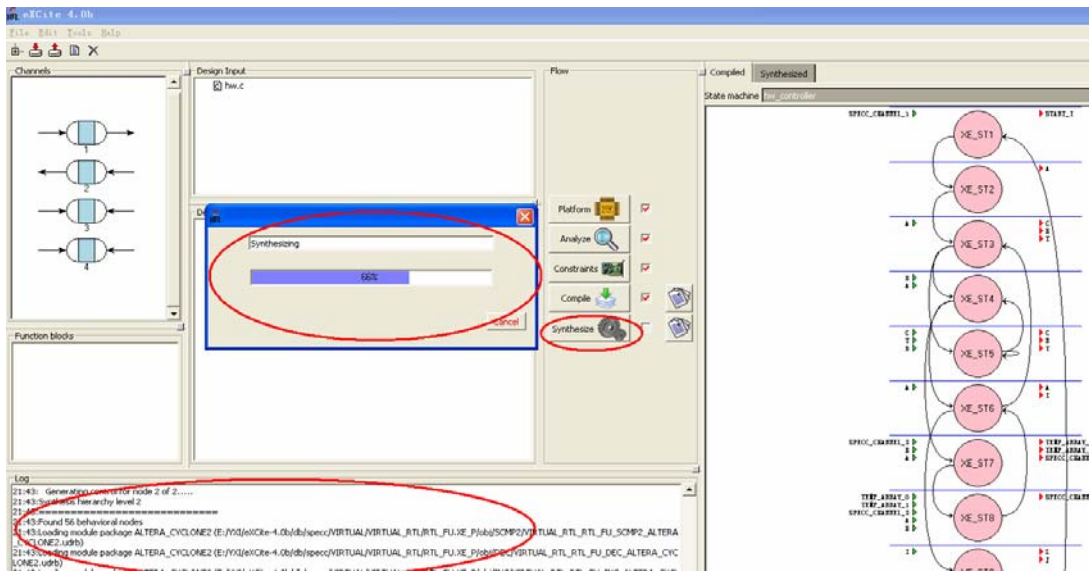
设置：添加所需的通道：“Constraints”

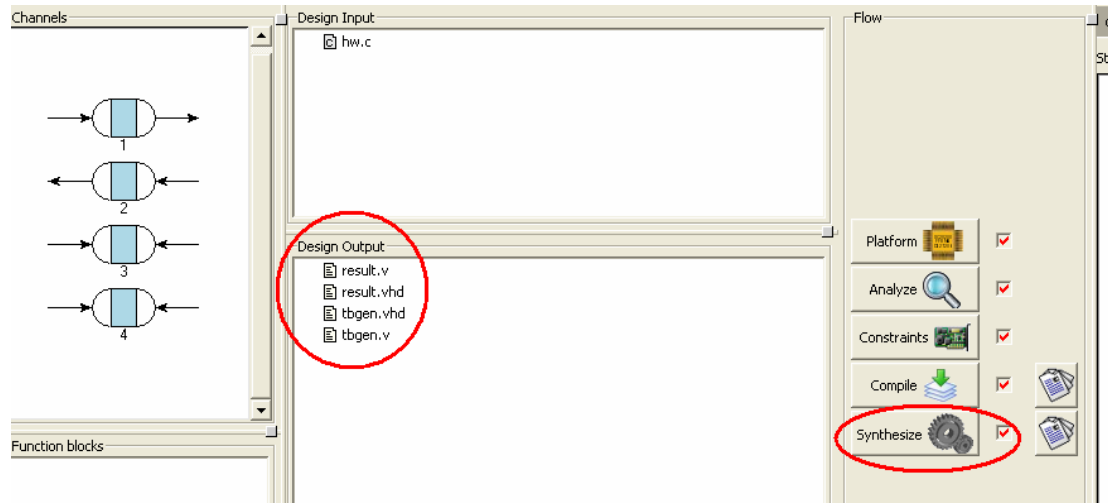


编译：“Compile” 编译后生成综合前的 FSM



综合：“Synthesize”，生成综合后化简状态图，测试文件 thgen.v，硬件 result.v





### 三 关于 sw 与 hw 的联调:

联调可以验证 sw 与 hw 之间数据传递执行结果是否正确,同时生成 databin.txt 文件。  
首先在 dos 环境下进入 example 文件夹目录:

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>E:
E:\>cd example
E:\example>
搜狗拼音 半:

```

生成 hw\_gen.c 文件——输入命令: csimgen -f hw.c -o hw\_gen.c

```

C:\WINDOWS\system32\cmd.exe
hw.c
done.
Analyzing and optimizing C code.
hw.c:15: scalar 'start_i' (1 X 8) is blocking message read channel 1
hw.c:18: scalar 'done_o' (1 X 8) is blocking message write channel 2
hw.c:22: array 'x' (16 X 32) is nonblocking memory read/write channel 3
hw.c:26: array 'W' (16 X 32) is nonblocking memory read/write channel 4
Frontend test: writing hw_gen.c for 100000 loop iterations
Parser post-optimization statistics:
  Operations: 0
  Simple variables: 15
  Array variables: 0
  Constants: 0
  Assignments: 0
  Reads: 0
Generated hw_gen.c.
Sun Mar 29 21:50:53 2009
E:\example>
搜狗拼音 半:

```

生成可执行文件 hw\_gen.exe ——输入命令: nmake hw\_gen.exe -f makefile.vc

```
C:\WINDOWS\system32\cmd.exe

Simple variables: 15
Array variables: 0

Constants: 0

Assignments: 0
Reads: 0

Generated hw_gen.c.
Sun Mar 29 21:53:03 2009
cl -Ie:\XXI\exCite-4.0h\bcl /Od /Z7 /c hw_gen.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

hw_gen.c
link /nodefaultlib /subsystem:console /entry:mainCRTStartup hw_gen.obj /
out:hw_gen.exe e:\XXI\exCite-4.0h\bcl\yx_bcl.lib msvrt.lib oldnames.lib ws2_32.
lib kernel32.lib
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

E:\example>
搜狗拼音 半:
```

生成 chan\_gen.h 文件——输入命令：ciogen -f chan.h -o chan\_gen.h

```
C:\WINDOWS\system32\cmd.exe

Analyzing and optimizing C code.
chan.h:5: scalar 'mio' (1 X 8) is blocking message write channel 1
chan.h:8: scalar 'mio' (1 X 8) is blocking message read channel 2
chan.h:12: array 'mio' (16 X 32) is nonblocking memory read/write channel 3
chan.h:12: array 'mio' (16 X 32) is nonblocking memory read/write channel 3
chan.h:16: array 'mio' (16 X 32) is nonblocking memory read/write channel 4
chan.h:16: array 'mio' (16 X 32) is nonblocking memory read/write channel 4
chan.h:5: scalar 'start_i' (1 X 8) is blocking message write channel 1
chan.h:8: scalar 'done_o' (1 X 8) is blocking message read channel 2
chan.h:12: array 'x' (16 X 32) is nonblocking memory read/write channel 3
chan.h:16: array 'W' (16 X 32) is nonblocking memory read/write channel 4
Frontend test: writing chan_gen.h for 1000 loop iterations
Parser post-optimization statistics:
Operations: 0

Simple variables: 17
Array variables: 0

Constants: 0

Assignments: 0
Reads: 0

Sun Mar 29 21:54:44 2009

E:\example>
搜狗拼音 半:
```

生成可执行文件 sw.exe ——输入命令：nmake sw.exe -f makefile.vc

```
C:\WINDOWS\system32\cmd.exe
Frontend test: writing chan_gen.h for 1000 loop iterations
Parser post-optimization statistics:
  Operations: 0

  Simple variables: 17
  Array variables: 0

  Constants: 0

  Assignments: 0
  Reads: 0

Sun Mar 29 21:55:59 2009
cl -Ie:\VXI\KXCite-4.0b\bc1 /Od /Z7 /c sw.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

sw.c
link /nodefaultlib /subsystem:console /entry:mainCRTStartup sw.obj /out:
sw.exe e:\VXI\KXCite-4.0b\bc1\yx_bc1.lib msucr.lib oldnames.lib ws2_32.lib kern
e132.lib
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

E:\example>
搜狗拼音 半:
```

再打开一个 dos 窗口，两个窗口分别输入 sw，hw\_gen，然后运行：

```
C:\WINDOWS\system32\cmd.exe - hw_gen
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>e:

E:\>cd example

E:\example>hw_gen
bc1_meschan.c:885: recv on 1 failed.
recv: 远程主机强迫关闭了一个现有的连接。

搜狗拼音 半:
```

hw.exe 运行后通道关闭

```
C:\WINDOWS\system32\cmd.exe

E:\example>sw
Begin testing...
The result are as follows
31000+11000i
-536+3535i
-2000-4000i
-6293-4950i
-7000-1000i
6536-3535i
2000-6000i
-7707+4950i
End testing.

E:\example>
搜狗拼音 半:
```

sw.exe 运行结果，计算出 8 点 FFT 的值

同时生成文本文件 data\_bin.txt，其中表示的是运行的输入和输出数据：

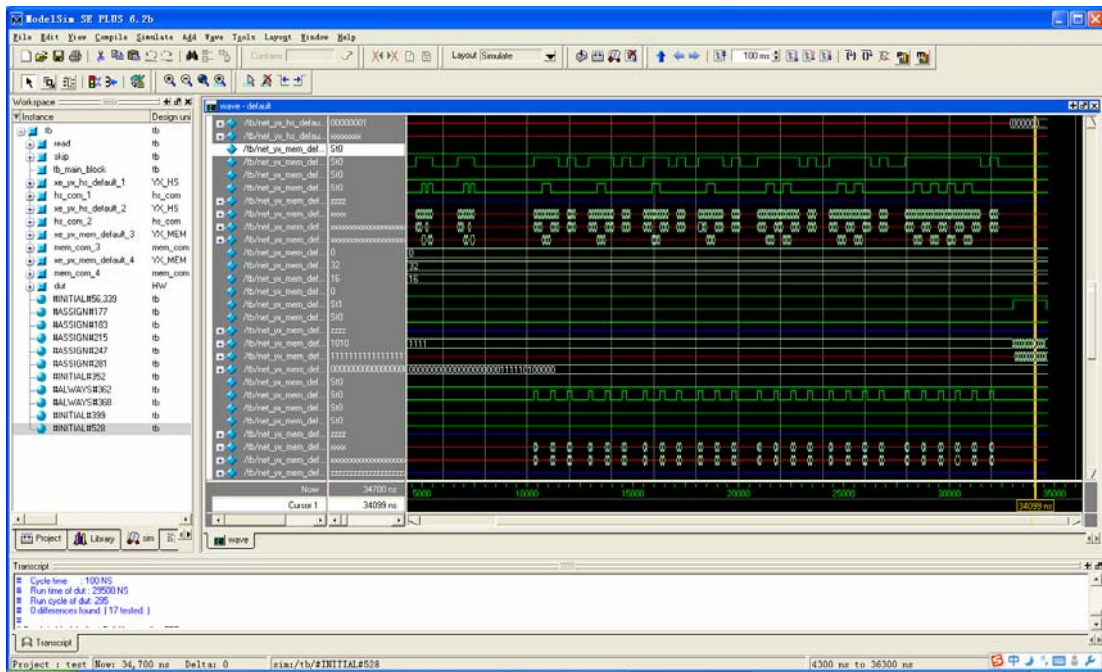
```

data_bin.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
# BCL version 3.1
# Sun Mar 29 21:58:31 2009
3 1 0
0000000000000000000000001111010000
3 1 1
00000000000000000000000000000000
3 1 2
0000000000000000000000001111010000
3 1 3
0000000000000000000000001111010000
3 1 4
0000000000000000000000010111011000
3 1 5
00000000000000000001001110001000
Ln 1, Col 1

```

#### 四. Modelsim 仿真

将生成的 data\_bin.txt, 测试文件 thgen.v, 硬件 result.v 放在同一文件夹下, 同时使用路径 \YXI\exCite-4.0b\simssyn\_packages\verilog\_src\SOLITON 中的所有文件, 以及路径 \YXI\exCite-4.0b\simssyn\_packages\verilog\_src\VIRTUAL\_RTL 中的文件: RTL\_FU.v 建立工程, 编译 thgen.v, result.v, RTL\_FU.v 三个文件, 仿真时选择顶层模块“tb”即可。



仿真结果:

```

Transcript
# Cycle time : 100 NS
# Run time of dut : 29500 NS
# Run cycle of dut : 295
# 0 differences found. ( 17 tested. )
#

```

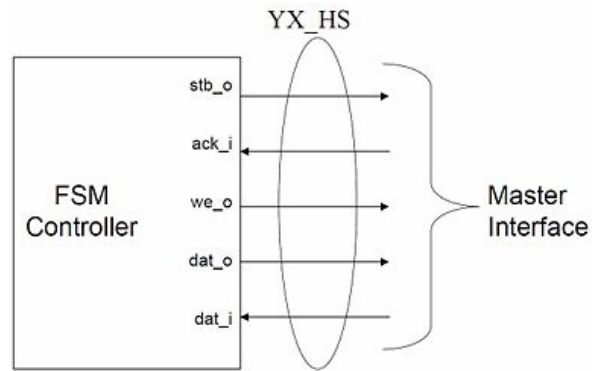
(在 10MHz (100ns) 的时钟下仿真)

硬件仿真结果与 data\_bin.txt 对照, 0 differences found. ( 17 tested. ), 周期数为 295。

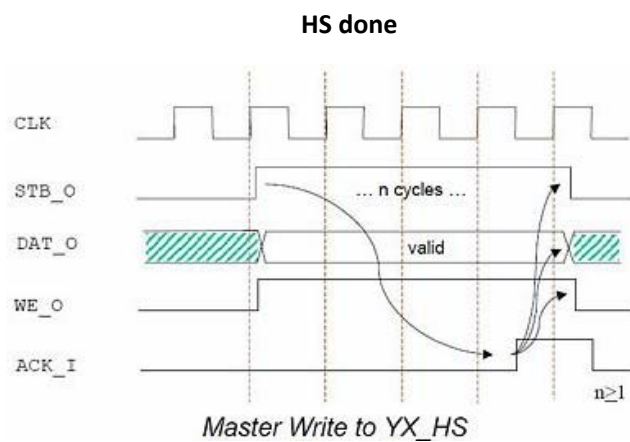
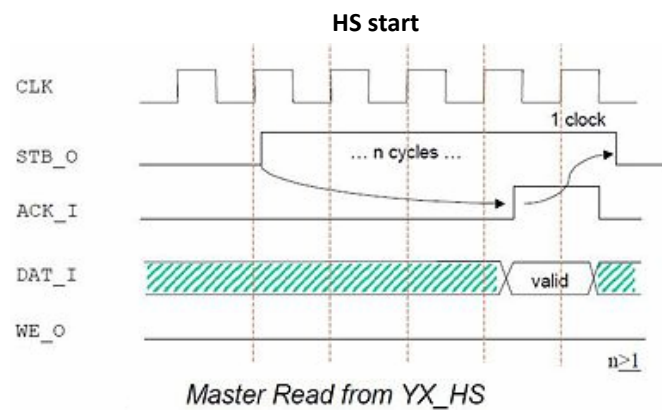
## 五. Quartus 仿真下载

eXCite 软件目前只能做到 modelsim 仿真这一步，下载验证需要编写 testbench，来调用 result.v (HW 的硬件程序文件) 中的函数 module HW，描述 HS，MEM 接口协议。

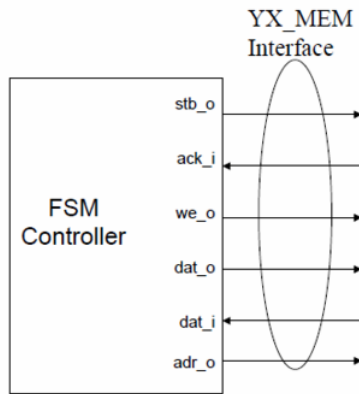
### 1. HS 接口的说明：



HS 通信方式如下图所示：（这是其中的一种方式）

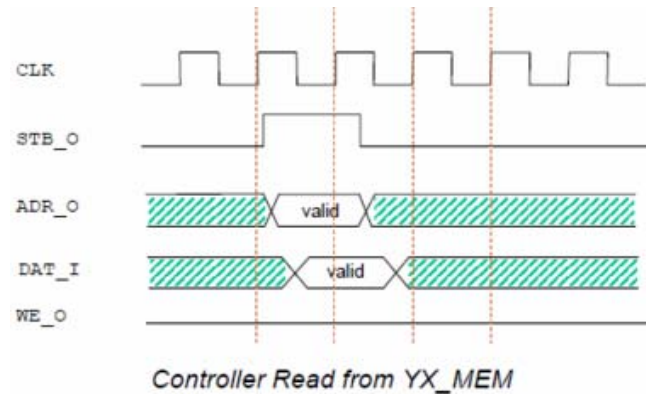


### 2. MEM 接口说明：

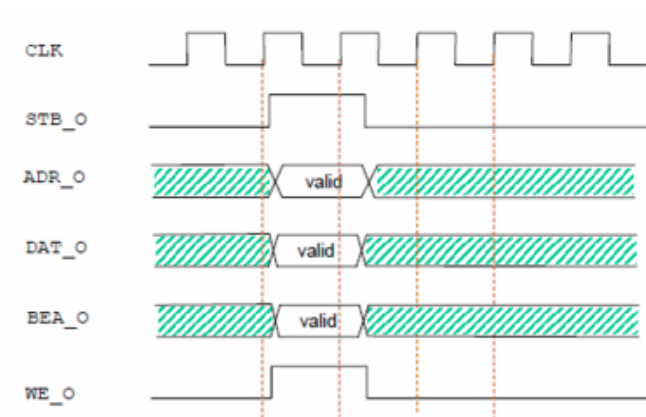


HS 通信方式如下图所示：（这是其中的一种方式）

### YX\_MEM Read Cycle

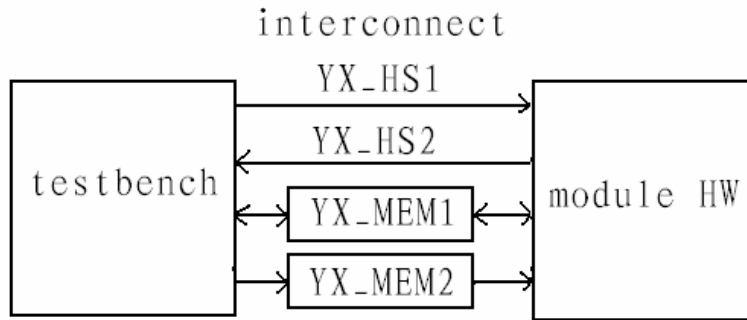


### YX\_MEM Write Cycle



本例采用——HS 双通道+ MEM 双通道：

如下所示：YX\_HS1 和 YX\_HS2 都是握手协议的通道，主要传递握手信号，不传数据。YX\_MEM1 和 YX\_MEM2 通道都是存储器的通道，YX\_MEM1 存放将 testbench 传来要处理的 8 点 X[8] 的值，YX\_MEM2 存放将 testbench 传来要处理的 W 旋转因子的值，两者同时进入 module HW 进行运算。运算完后的 8 点值仍然通过 YX\_MEM1 输出给 testbench，YX\_MEM2 通道是单向的，只向 module HW 传入数据，没有输出。

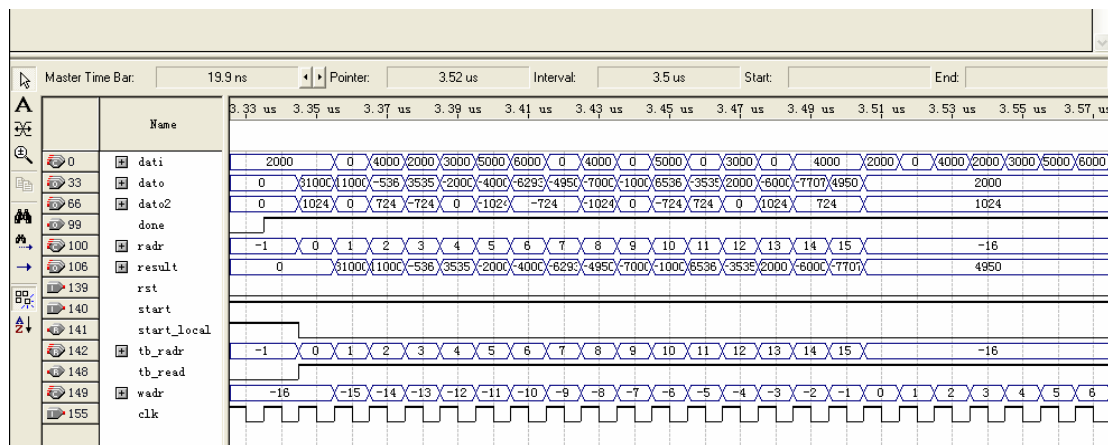


**HW 模块的接口名称:**

```
module HW (clk, rst, yx_hs_1_ack_i, yx_hs_1_dat_i, yx_hs_2_ack_i,
  yx_mem_1_dat_i, yx_mem_2_dat_i, yx_hs_1_stb_o, yx_hs_1_we_o, yx_hs_2_stb_o,
  yx_hs_2_we_o, yx_hs_2_dat_o, yx_mem_1_stb_o, yx_mem_1_we_o, yx_mem_1_adr_o,
  yx_mem_1_dat_o, yx_mem_2_stb_o, yx_mem_2_we_o, yx_mem_2_adr_o);
```

其中 HS 接口: yx\_hs\_1\_ack\_i, yx\_hs\_1\_stb\_o, yx\_hs\_1\_we\_o, yx\_hs\_1\_dat\_i, ( HS start )  
 yx\_hs\_2\_ack\_i, yx\_hs\_2\_stb\_o, yx\_hs\_2\_we\_o, yx\_hs\_2\_dat\_o, ( HS done )  
 MEM 接口: yx\_mem\_1\_dat\_i, yx\_mem\_1\_stb\_o, yx\_mem\_1\_we\_o, yx\_mem\_1\_adr\_o,  
 yx\_mem\_1\_dat\_o, (MEM1)  
 yx\_mem\_2\_dat\_i, yx\_mem\_2\_stb\_o, yx\_mem\_2\_we\_o, yx\_mem\_2\_adr\_o (MEM2)  
 其他接口: clk,rst,

**quartus 波形如下图所示: 时钟周期 10ns**



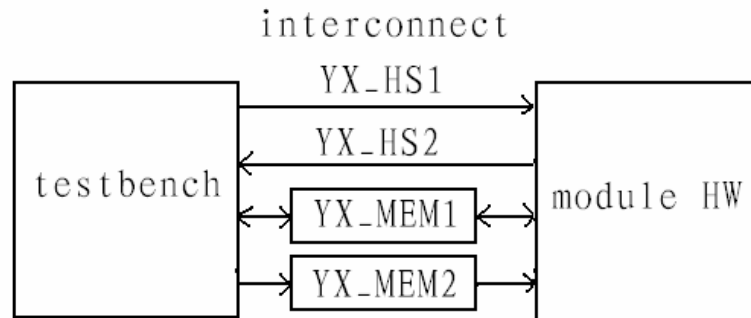
输出结果正确, 时钟数大约是 330 多个, 与 modelsim 基本一致。

## 附录一 hw 与 sw 函数的拆分（以本例说明）

### 1. 拆分方法

对一个普通的 C 代码进行拆分时，要选择合适的函数用来加速，可以是一个函数，可以是几个函数，但是要找到他们顶层的输入量和输出量，作为通道的接口。

以本例来说，对于 8 点的 FFT，变址部分和蝶形单元运算部分比较复杂，需要进行复数的加减乘运算，于是将这两部分放在 hw 中。它的通道输入输出分别是： $x[8][2]$ 、 $W[8][2]$ 与  $x[8][2]$ ，其中两个输入量，一个输出量（与其中一个输入量用同一通道），如下图所示：



这样通道的代码描述，如红框中所示，两个 HS 接口信号（用于连接通信，握手协议），两个 MEM 接口信号（用于传递数据），同时要引用“yx\_bcl.h”头文件。

```
#include <stdio.h>
#include "yx_bcl.h"

int
main()
{
    int i,j,k,l,a,b,c,d,e,t;

    //yx array_eliminate
    int aa[4],dd[2],bb[2],cc[2],temp[2];

    unsigned char start;

    //yx channel_transform 1 message blocking read
    unsigned char start_i;

    //yx channel_transform 2 message blocking write
    unsigned char done_0;

    //yx channel_transform 3 memory nonblocking
    //yx array_eliminate
    int x[8][2];

    //yx channel_transform 4 memory nonblocking
    //yx array_eliminate
    int W[8][2];

    while(1)
    {
        // Receive Start Flag
        start = start_i;

        for (a=0;a<8;a++)
```

将需要加速的函数放到 `While (1) { 需要加速的函数 }`，注意，函数运行中所需的中间变量，需要在 `While (1)` 之外去定义数据类型；赋值时需要在 `While (1)` 内进行，这是软件代码编写的规则，如下图所示：

```
grad_mag
#include <stdio.h>
#include "yx_bcl.h"

int
main()
{
    int i,j,k,l,a,b,c,d,e,t;

    //yx array eliminate
    int aa[4],dd[2],bb[2],cc[2],temp[2];

    unsigned char start;

    //yx channel_transform 1 message blocking read
    unsigned char start_i;

    //yx channel_transform 2 message blocking write
    unsigned char done_o;

    //yx channel_transform 3 memory nonblocking
    //yx array_eliminate
    int x[8][2];

    //yx channel_transform 4 memory nonblocking
    //yx array_eliminate
    int W[8][2];

    while(1)
    {
        // Receive Start Flag
        start = start_i;

        for(a=0;a<8;a++)
        {
            c=a;
            b=0;
            t=3;

```

在 hw 的结尾需要写明 HS 完成的接口信号 done，写在 while（1）之内。如下图所示：

```

        dd[1]=x[j+k][1]+bb[1];

        cc[0]=x[j+k][0]-bb[0];
        cc[1]=x[j+k][1]-bb[1];

        x[j+k][0]=dd[0];
        x[j+k][1]=dd[1];

        x[j+k+1][0]=cc[0];
        x[j+k+1][1]=cc[1];
    }
}

// Send Done Flag
done_o = 1;
}
```

对于 SW 部分，需要引用“chan\_gen.h”头文件，其中，chan\_gen.h 是 chan.h 文件在联调中生成的，即：**生成 chan\_gen.h 文件——输入命令：ciogen -f chan.h -o chan\_gen.h**

chan.h 文件是由我们编写的，主要内容是通道信号定义，与 hw 中的保持一致，同时需要引用“yx\_bcl.h”头文件，代码如下图所示：

```
C++ - [chan.h]
rt Project Build Tools Window Help
grad_mag
#include "yx_bcl.h"

//yx generate_io
//yx channel_transform 1 message blocking write
unsigned char start_i;
//yx generate_io
//yx channel_transform 2 message blocking read
unsigned char done_o;

//yx generate_io
//yx channel_transform 3 memory nonblocking
int x[8][2];

//yx generate_io
//yx channel_transform 4 memory nonblocking
int W[8][2];
```

sw 中需要写明数据的读入读出，如下图所示：

```
PI=atan(1)*4;
yx_io_init(); 通道初始化
printf("Begin testing...\n");

x[0][0]=2000;
x[0][1]=0;
x[1][0]=4000;
x[1][1]=2000;
x[2][0]=3000;
x[2][1]=5000;
x[3][0]=6000;
x[3][1]=0;
x[4][0]=4000;
x[4][1]=0;
x[5][0]=5000;
x[5][1]=0;
x[6][0]=3000;
x[6][1]=0;
x[7][0]=4000;
x[7][1]=4000;

yx_write_x(&x); 写入数据
for(b=0;b<size_x;b++)
{
    W[b][0]=cos(2*PI/size_x*b)*1024;
    W[b][1]=-1*sin(2*PI/size_x*b)*1024;
}

yx_write_W(&W); 写入数据
yx_write_start_i(&start); 写入开始信号
yx_read_done_o(&done); 读出完成信号
yx_read_x(&x); 读出数据

//输出函数
printf("The result are as follows\n");
```

## 2. 注意事项

注意在 hw 中不要出现 while (1) 内外重复定义数据类型和赋值的情况，同时注意尽量避免过多进行变量 i 的使用，因为系统中很多地方也用到了变量 i，比如数据 io 接口部分等。

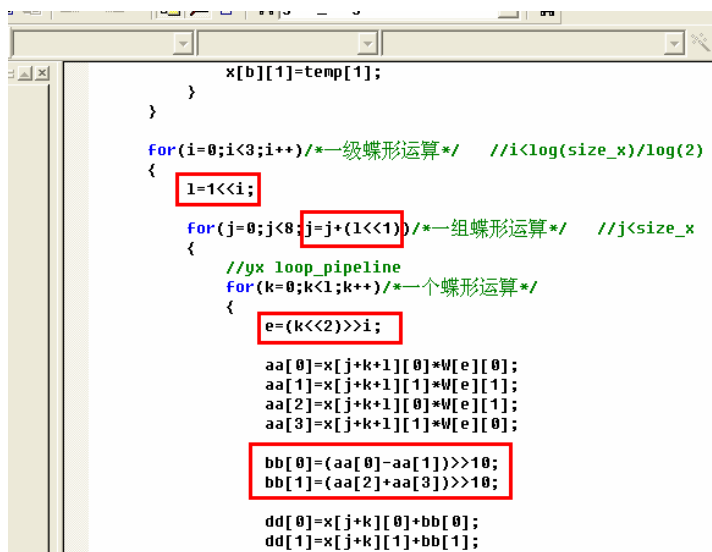
另外，目前的 hw 中还不能支持 math.h 库函数，所以一些特定函数的运算需要在 sw 中实现，或者变换合适的运算形式在 hw 中进行。

在 hw 运算中，尽量不要采用 float，double 等浮点数，可以改为 int 等定点数，浮点数在硬件综合时可能会导致运算结果不对。

## 附录二 hw 函数的优化——用于硬件加速：有效减小状态数和时钟数（以本例说明）

初始完成的 hw 加速效果通常不是很好，需要进行进一步的优化，从硬件的角度去编写和优化程序，举例如下：

1. **逻辑（移位）运算对于乘除法器优化：**在硬件中，调用乘除法器是很费资源的，但如果是基 2 的乘除法运算等，可以采用移位的形式进行，像这样的逻辑运算能够节省资源。如下图所示：



```
x[b][1]=temp[1];
}
}
for(i=0;i<3;i++)/*一级蝶形运算*/ //i<log(size_x)/log(2)
{
    l=1<<i;
    for(j=0;j<8;j=j+(1<<l))/*一组蝶形运算*/ //j<size_x
    {
        //yx loop_pipeline
        for(k=0;k<1;k++)/*一个蝶形运算*/
        {
            e=(k<<2)>>i;
            aa[0]=x[j+k+1][0]*W[e][0];
            aa[1]=x[j+k+1][1]*W[e][1];
            aa[2]=x[j+k+1][0]*W[e][1];
            aa[3]=x[j+k+1][1]*W[e][0];
            bb[0]=(aa[0]-aa[1])>>10;
            bb[1]=(aa[2]+aa[3])>>10;
            dd[0]=x[j+k][0]+bb[0];
            dd[1]=x[j+k][1]+bb[1];
        }
    }
}
```

2. **For 循环的流水线优化：**采用流水线的方式也能有效提高加速效果，对于 for 循环等，插入流水线只能在最内层的循环架构中，同时要保证插入流水线的循环再数据上前后依赖性小，顺序执行性好，如果不适合加流水线，在软件分析和编译的时候会报错。使用如下图所示：

```

        x[b][1]=temp[1];
    }
}
for(i=0;i<3;i++)/*一级蝶形运算*/ //i<log(size_x)/log(2)
{
    l=1<<i;
    for(j=0;j<8;j=j+(1<<1))/*一组蝶形运算*/ //j<size_x
    {
        //yx loop pipeline
        for(k=0;k<1;k++)/*一个蝶形运算*/
        {
            e=(k<<2)>>i;

```

3. **数组定义时的优化：**对于数组的优化，是将数组中的数据从较慢的 SRAM，DRAM 中，改存在 REG 中，速度提高也很明显，但相对的，对占用大一些的面积，可以折衷考虑。

```

int
main()
{
    int i,j,k,l,a,b,c,d,e,t;
    //yx array eliminate
    int aa[4],dd[2],bb[2],cc[2],temp[2];

    unsigned char start;

    //yx channel_transform 1 message blocking read
    unsigned char start_i;

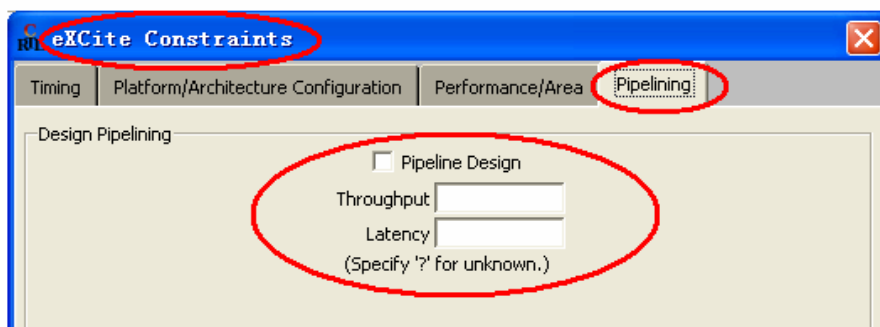
    //yx channel_transform 2 message blocking write
    unsigned char done_o;

    //yx channel transform 3 memory nonblocking
    //yx array eliminate
    int x[8][2];

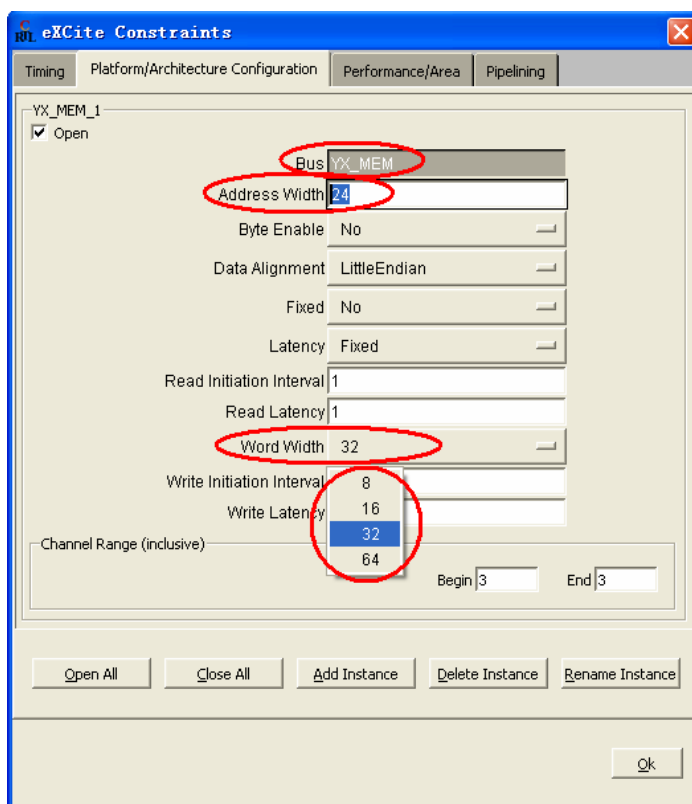
    //yx channel transform 4 memory nonblocking
    //yx array eliminate
    int w[8][2];

```

4. **打开循环结构：**在 hw 中，结构上比较复杂，但运算次数很少的 for 循环，如果进行赋值等工作，可以将循环打开直接赋值，直接描述功能，也会有优化的效果。
5. **整体的 hw 加入流水线：**hw 中的结构如果是顺序执行的，也可以做流水线处理，如下图所示，选择合适的延迟等指标即可。



6. **通道字宽，地址宽度的设置：**使用 MEM 接口时，可以适当增加数据的字宽，比如 32 位，对于地址宽度可以根据实际需要来确定，如下图所示：



其实，类似的优化方法还有很多，综合使用能够有效地减少状态数和周期数，另外关于 eXCite Constraints 设置，有很多内容可以根据实际的需要进行，来达到最好的优化效果。

其他相关问题可以参见eXCite软件帮助文档中的“eXCite Advanced”中的内容。