3.2.1 生成 IP 核

本设计需要使用 ROM/RAM 来保存地图相关信息以及要显示在 VGA 上的图片。

3.2.1.1 生成 coe 文件

COE 文件的格式是：

```
memory_initialization_radix=数据的进制；
memory_initialization_vector=数据，用逗号隔开，用分号结尾。
```

对于地图信息，本设计可以直接输入 COE 文件：

```
memory_initialization_radix=16;
memory_initialization_vector=
2,

3, 4, 5, 6, 4, 3, 8, 2, 5, 0, 0, 5,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
2, 0, 0, 0, 0, 0, 1, 1, 3, 0, 2,
2, 0, 1, 1, 4, 0, 0, 1, 0, 0, 2,
2, 0, 1, 1, 1, 0, 0, 1, 0, 0, 2,
2, 1, 1, 1, 1, 0, 0, 1, 0, 0, 2,
2, 0, 1, 1, 1, 1, 1, 1, 0, 0, 2,
2, 0, 1, 1, 1, 0, 0, 1, 0, 0, 2,
2, 0, 1, 1, 1, 1, 0, 0, 0, 0, 2,
2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

6, 5, 8, 7, 4, 4, 8, 5, 5, 0, 0, 5,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
2, 0, 1, 1, 1, 1, 1, 1, 1, 0, 2,
2, 0, 1, 0, 0, 0, 0, 0, 1, 0, 2,
2, 0, 1, 0, 4, 1, 1, 1, 1, 0, 2,
2, 1, 1, 0, 1, 1, 1, 1, 3, 0, 2,
2, 0, 0, 0, 1, 1, 1, 0, 1, 0, 2,
2, 0, 1, 1, 1, 1, 1, 0, 1, 0, 2,
2, 0, 1, 1, 1, 1, 1, 0, 1, 0, 2,
2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2;
```

上面是两关的信息，具体来讲：

```
首先是 1 个数字，表示关卡数

每关起始是 12 个数字，分别代表
```

```
起始豆子 起始箱子 豆子目标 箱子目标的行和列（从 0 计数，边框也算；等价于从 1 计数）
以及左右上下方向的入口位置

然后是 11*11 的地图（含一圈外边框）
0 - 墙
1 - 路
2 - 外边框，走到这里表示走出入口
3 - 箱子的目标
4 - 墙的目标

（每关计数 133 个数据）
```

对于图片，首先使用 Photoshop 将它们拼合：

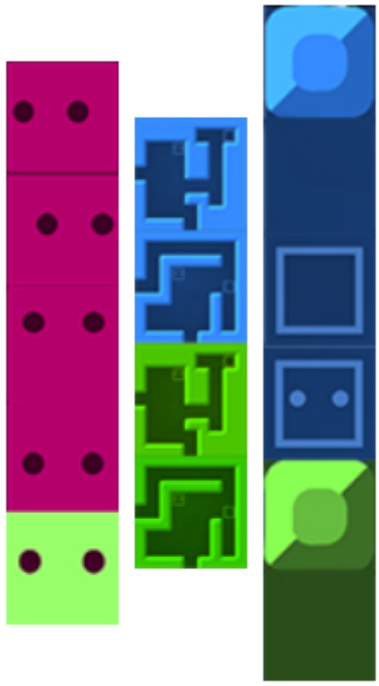

图 3.3 - 用到的素材图片（共三张）

我们编写了 MATLAB 代码来将 jpg 图片转化为 coe 文件：

```
clc;
clear all;
[filename, path] = uigetfile('*.jpg');
oriFilename = fullfile(path, filename);
RGB=imread(oriFilename);
R=RGB(:,:,1);
G=RGB(:,:,2);
B=RGB(:,:,3);
n = size(RGB, 1);
m = size(RGB, 2);
outdata=zeros(1,n * m);
for r = 1:n
    for c = 1:m
```

```
14          outdata((r-1)*m+c) = bitshift(bitand(R(r,c),240),4) +
    bitshift(bitand(G(r,c),240),0) + bitshift(bitand(B(r,c),240),-4);
15      end
16  end
17  fid=fopen([filename(1:end-4), '.coe'],'w+');
18  fprintf(fid,'memory_initialization_radix=16;\nmemory_initialization_ve
    ctor=\n');
19  for r = 1:n
20      for c = 1:m
21          if (r == n && c == m)
22              fprintf(fid, '%x%x%x;', bitand(B(r,c),240)/16,
    bitand(G(r,c),240)/16, bitand(R(r,c),240)/16);
23          else
24              fprintf(fid, '%x%x%x, ', bitand(B(r,c),240)/16,
    bitand(G(r,c),240)/16, bitand(R(r,c),240)/16);
25          end
26          if (c == m)
27              fprintf(fid, '\n');
28          end
29      end
30  end
31
32  fclose(fid);
```

这样，我们得到了所有需要加载的内容的 coe 文件。

3.2.1.2 生成 IP 核

New Source，选择 IP CORE，命名

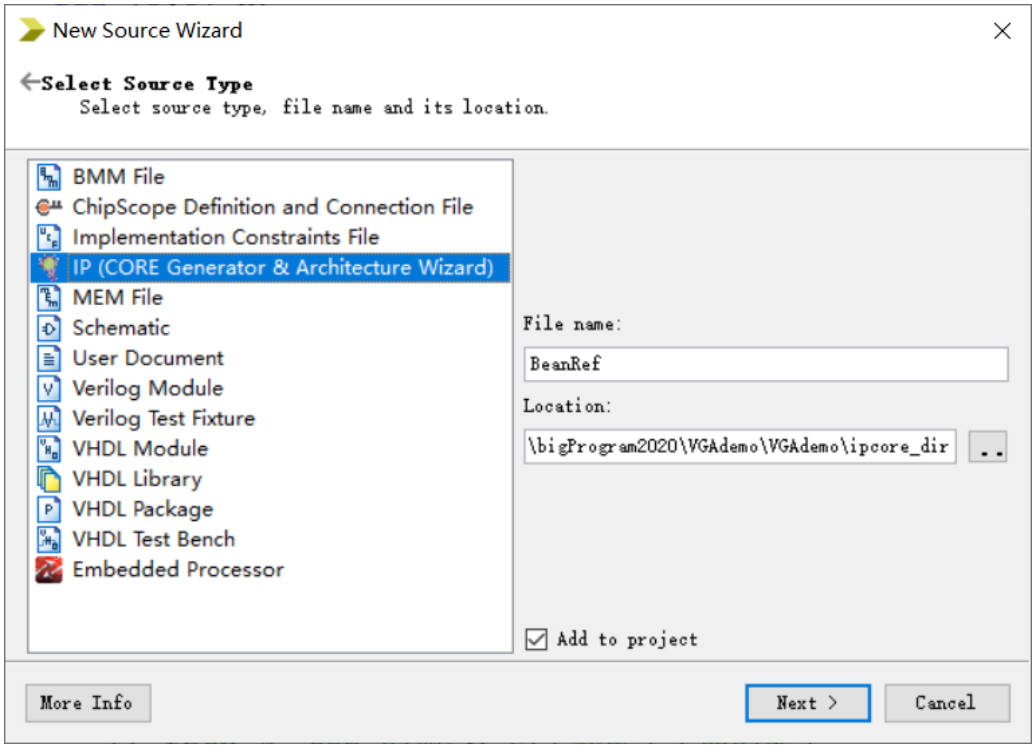

图 3.4 - 生成 IP 核步骤(1)

选择 Block Memory Generator

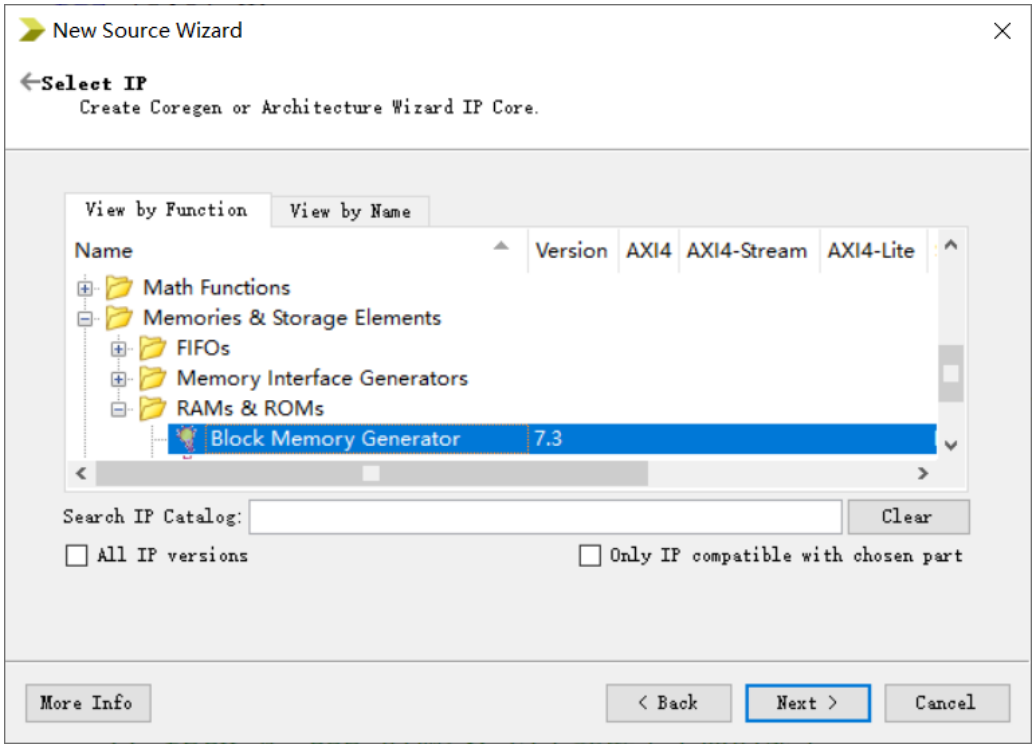

图 3.5 - 生成 IP 核步骤(2)

选择 Single Port RAM

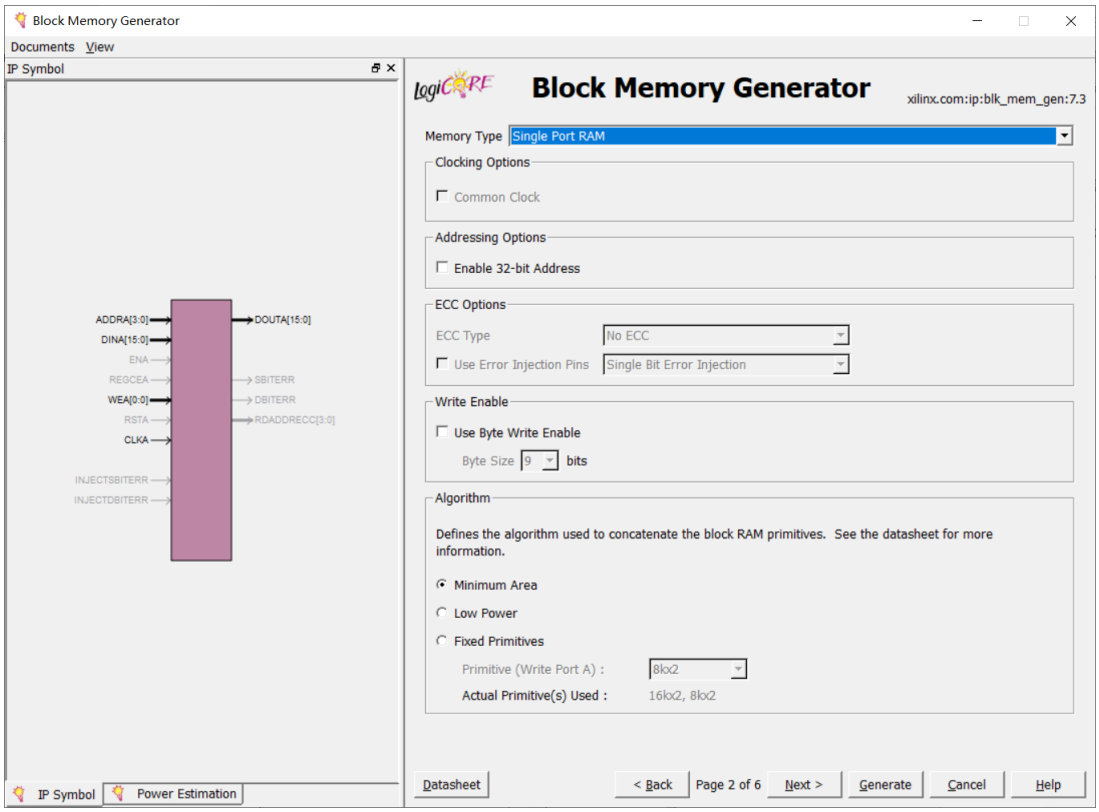

图 3.6 - 生成 IP 核步骤(3)

Write / Read Width：即每个数据的宽度。设置为 coe 中数据的最大位数（二进制）。

Write Depth：即数据的最大数目。设置为不小于 coe 中数据数目的数。

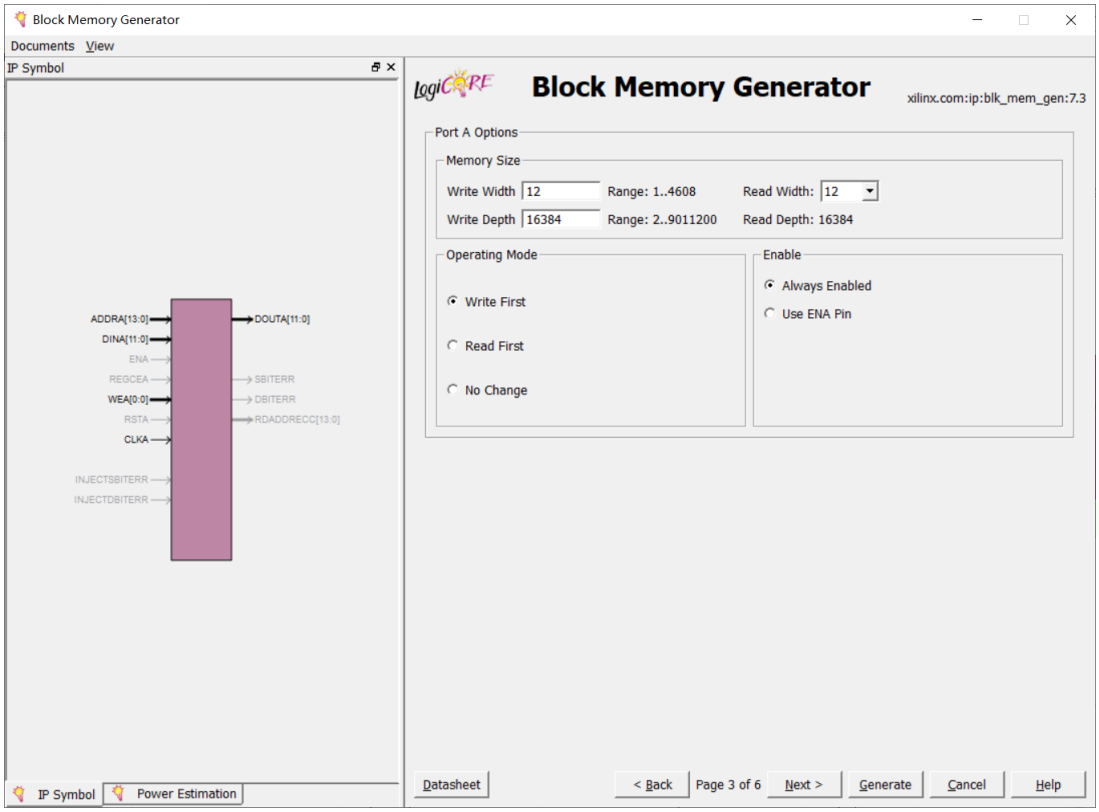

图 3.7 - 生成 IP 核步骤(4)

选择 Load Init File，选择自己的 coe 文件，然后直接 Generate

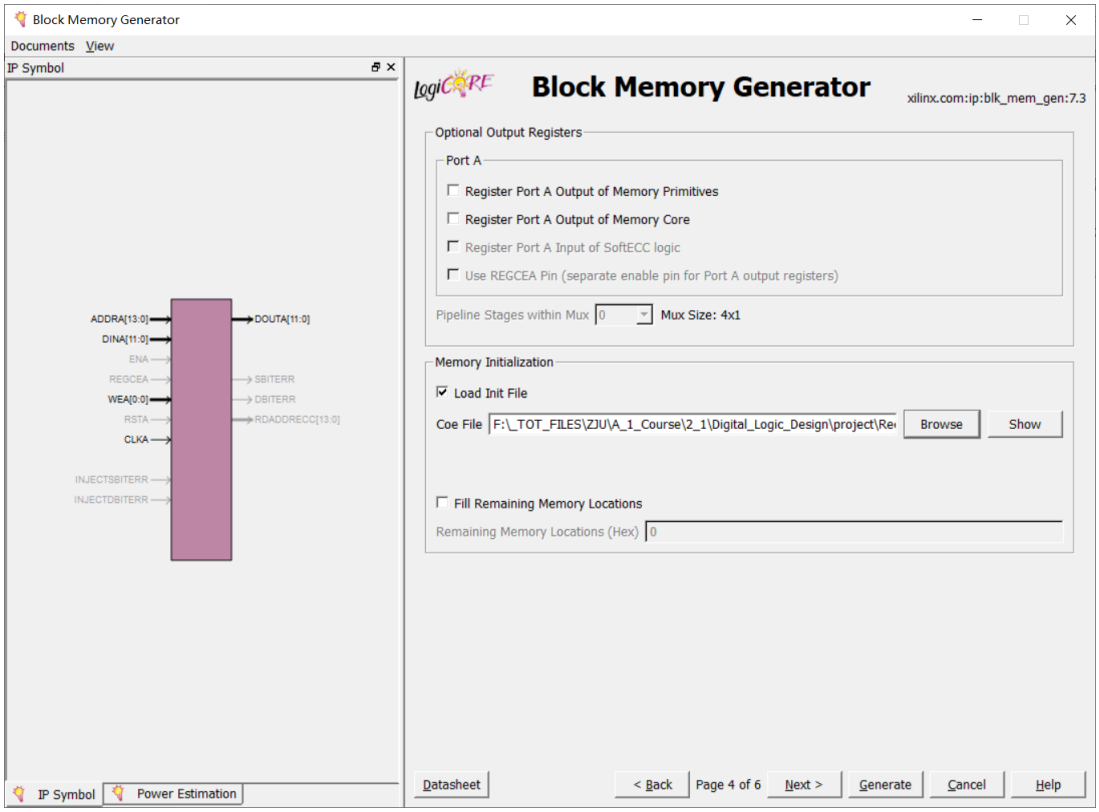

图 3.8 - 生成 IP 核步骤(5)

3.2.1.3 调用 IP 核

IP 核（我们这里使用的是 RAM 且只用来读取）实际上就是一个数组。我们主要使用其输入端口 `clka`，`addra` 和输出端口 `douta` 。

我们在多处调用了 IP 核，例如这几处：

```
    // 读取 Bean 和 Map 的颜色
    wire [11:0] beanOut, mapOut;
    wire [13:0] refAddr;
    assign refAddr = picInd * blkSz * blkSz + x_off + y_off * blkSz;
    BeanRef beanRef(.clka(clk), .addra(refAddr), .douta(beanOut));
    MapRef mapRef(.clka(clk), .addra(refAddr), .douta(mapOut));

    // 豆子上下左右的格子类型 0~4
    wire [3:0] left, right, up, down;
    wire [9:0] addr;
    assign addr = 1 + LevelOffset + WholeLevel * level + x + y * 11;
    LevelRef leftRef(.clka(clk), .addra(addr - 1), .douta(left));
    LevelRef rightRef(.clka(clk), .addra(addr + 1), .douta(right));
    LevelRef upRef(.clka(clk), .addra(addr - 11), .douta(up));
    LevelRef downRef(.clka(clk), .addra(addr + 11), .douta(down));
```

可以看到，我们只使用了这三个端口；只要给 `addra` 赋值， `douta` 就会输出对应位置的数据。

我们发现，IP 核就像是一个模块，而且同一个 IP 核可以建立多个实例。

需要注意的是，我们需要根据 IP 核各端口的实际位数（由之前设置的 width 和 depth 决定）来设置 wire 的位数。

3.2.2 VGA 显示

我们使用了 VGA Demo 中的 vgac 模块，在 top.v 中创建了如下实例和相关内容：

```
1       // 处理 VGA 输出：
2       // input clrn: ~clear, if it's 0 then clear, 也就是需要 SW[0] 为 1
3       // input vga_data: bbbb_gggg_rrrr
4       // output row_addr & col_addr: 需要给出 vga_data 的像素坐标
5       // 其它是 top 的输入输出，不用管
6       reg [11:0] vga_data;
7       wire [9:0] col_addr;
8       wire [8:0] row_addr;
9       vgac v0 (
10          .vga_clk(clkdiv[1]), .clrn(SW_OK[0]), .d_in(vga_data),
    .row_addr(row_addr),
11          .col_addr(col_addr), .r(r), .g(g), .b(b), .hs(hs), .vs(vs)
12      );
```

VGA 显示的逻辑是，在每个时钟的上升沿，vgac 其更新一个 `row_addr` 和 `col_addr`，表示现在需要显示的 VGA 屏幕上的像素的坐标。我们需要给 VGA 输入 `d_in`，即我们程序中的 `vga_data`，表示该像素点的颜色。然后 vgac 据此更新 `r`，`g`，`b`，`hs`，`vs` 等的值传给 top 模块再由 top 接给对应引脚，实现输出。

本设计最关键的就是 VGA 显示，而从上面的分析我们可以看出，我们要做的事就是**对每一时刻的 `row_addr` 和 `col_addr`，给出一个对应的 `vga_data`，即这个像素要显示的颜色。**