概要

本篇说明文档介绍了基于 RL78/G13 单片机的 CMOS 摄像头图像采集处理程序的编写。主要介绍了如何在代码生成工具 CubeSuit+下配置本例程所需要的 UART、简易 IIC、外部中断、Port 等模块并生成相应的驱动代码，同时对生成的代码文件添加的摄像头驱动与数据存储函数进行了详细说明。后附 CMOS 摄像头图像采集处理程序源码实例，用户可以结合本例程对 RL78/G13 单片机进行摄像头循迹等的拓展应用。

对象 MCU

R5F100LEA

目录

1. 系统功能概述 3
1.1 CMOS 摄像头循迹原理 3
1.2 CMOS 摄像头循迹例程功能 3
2. 系统硬件说明 4
2.1 开发板介绍 4
2.2 摄像头模组 4
2.2.1 模组实物图 4
2.2.2 接口定义 5
2.2.3 模组介绍 5
2.2.4 OV7620 时序分析 6
2.2.5 图像数据自动存储 7
2.2.6 OV7620 寄存器设置 7
2.2.7 模块连接 8
2.2.8 图像读取方法 9
2.2.9 软件滤波算法 9
2.2.10 黑线中心提取方法 10
3. 例程软件说明 11
3.1 集成开发环境 11
3.2 CMOS 摄像头循迹例程编写 11
3.2.1 CubeSuite+界面配置 12
3.2.2 添加摄像头模块程序 18
4. 例程测试结果 28
4.1 图像采集存储速度测试结果 28
4.2 二值化存储测试结果 29
4.3 分辨率测试结果 30
4.4 开窗测试结果 31
5. 参考文献 34
结束语 34

1. 系统功能概述

1.1 CMOS 摄像头循迹原理

通过图像传感器对实时图像的采集，得到道路的信息，对当前采集到的图像信息做出判断，从而得到道路的情况，之后通过进一步控制实现循迹功能。

1.2 CMOS 摄像头循迹例程功能

本例程利用 OV7620 CMOS 摄像头模块成像 1 米外路径图像（白底一条黑线），将一帧数据转换成“0”“1” 镜像到 MCU RL78/G13 的 RAM 中。如图 1.1 图像二值化存储示例。

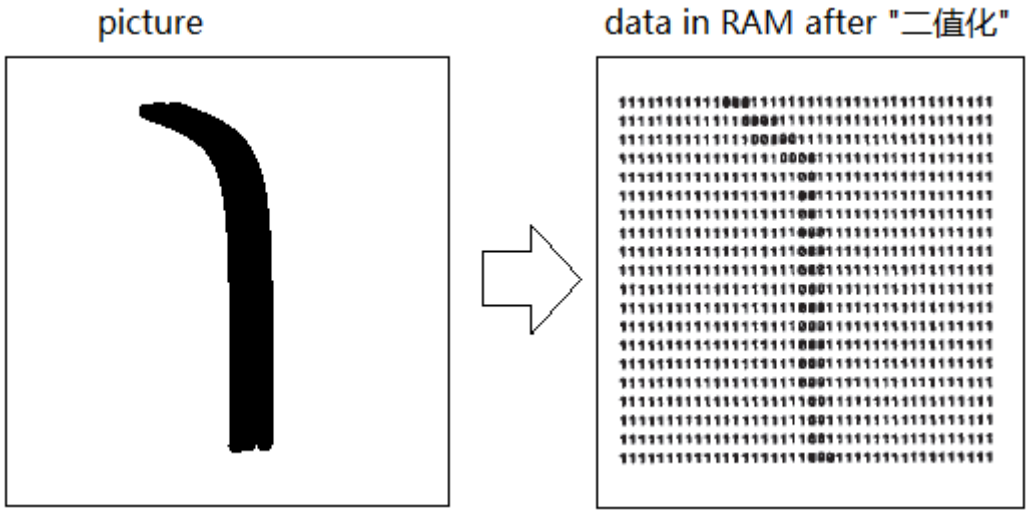

图 1.1 图像二值化示例

2. 系统硬件说明

2.1 开发板介绍

本例程是基于 RL78/G13 开发套件进行开发。下面是开发板实物图：

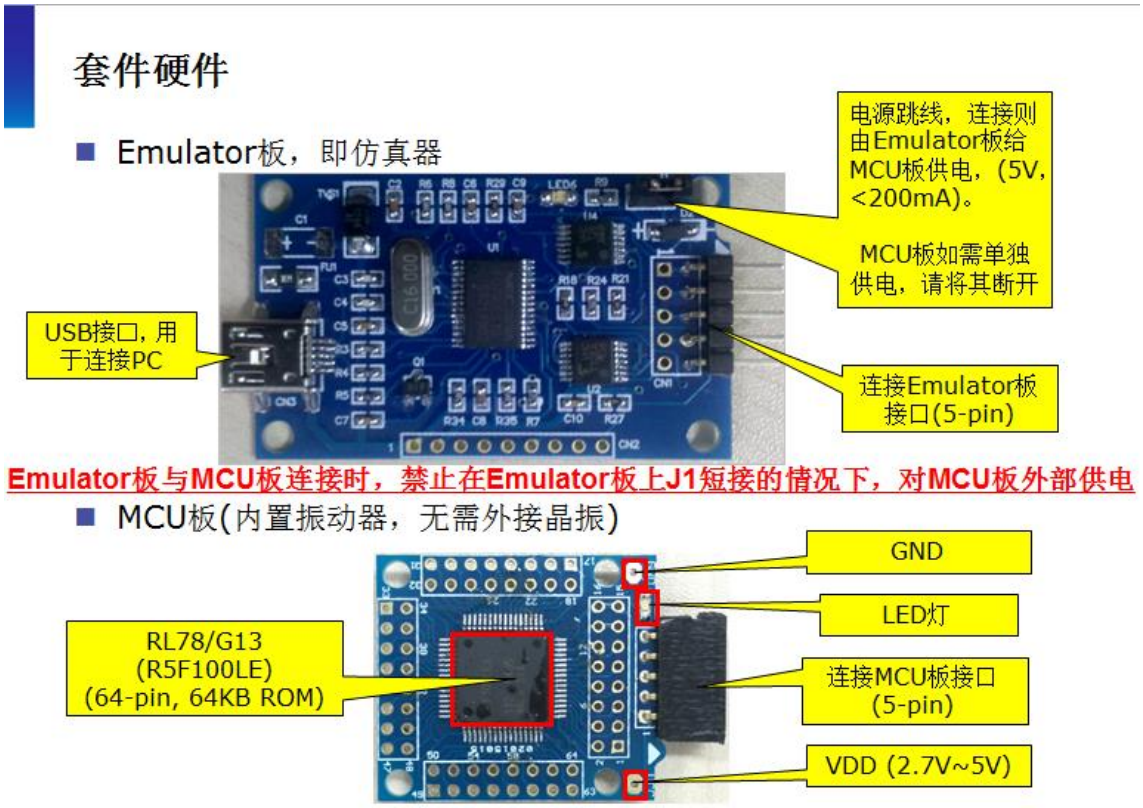

图 2.1　RL78/G13 开发套件硬件实物图

2.2 摄像头模组

2.2.1 模组实物图

本例程采用的是摄像头集成模组 FIFO-OV7620，该模组自带 FIFO，由 CMOS 摄像头模块 OV7620 和 FIFO 模块 AL422 组成，有效解决了摄像头与单片机之间的数据读取不同步的问题。该集成模组共有 20 个外接引脚，各引脚定义请参照 2.2.2 接口定义。

FIFO-OV7620 模组实物图如 2.2 所示，正面是 OV7620 摄像头，背面是 AL442FIFO

以及周边电路。

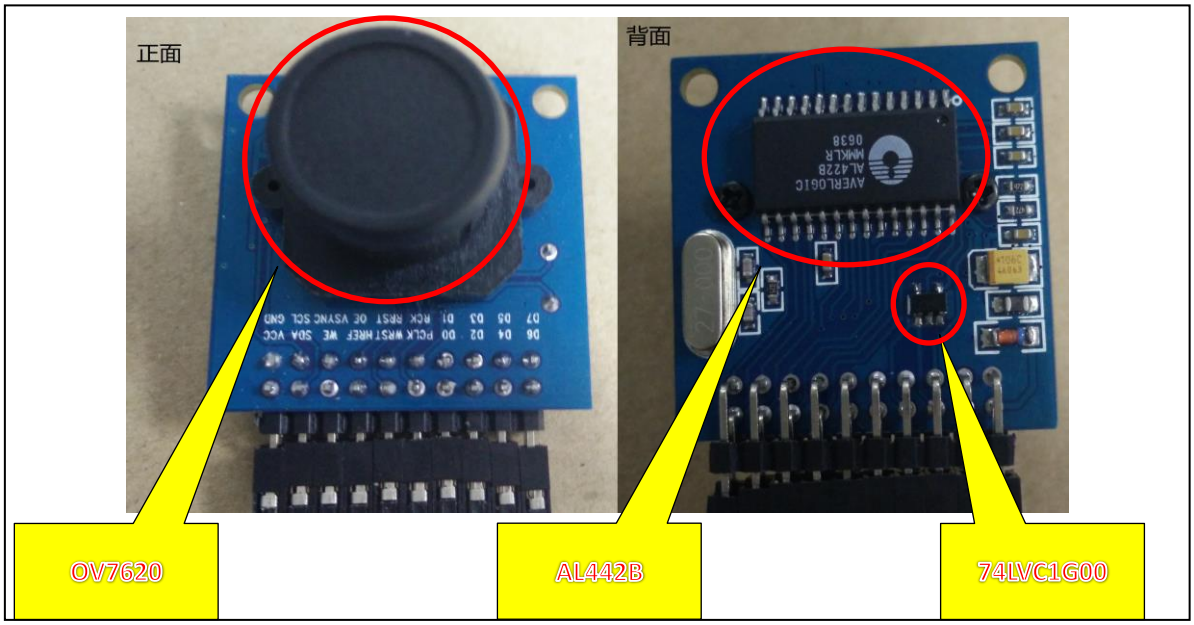

图 2.2　摄像头模组实物图

2.2.2 接口定义

引脚 1：VCC；电源
引脚 2：GND；地
引脚 3：SDA（摄像头 SCCB 总线 SDA 端口）；
引脚 4：SCL（摄像头 SCCB 总线 SCL 端口）；
引脚 5：WE（摄像头往 FIFO 写入数据的使能端口，高有效）；
引脚 6：VSYNC（摄像头场信号端口）；
引脚 7：HREF（摄像头行信号端口）；
引脚 8：OE（FIFO 输出使能端口，低有效）；
引脚 9：WRST（FIFO 写复位端口，低有效）；
引脚 10：RRST（FIFO 读复位端口，低有效）；
引脚 11：PCLK（FIFO 写时钟端口）；
引脚 12：RCK（FIFO 读时钟端口，上升沿读取数据）；
引脚 13-20：D[7:0]（FIFO 输出数据端口）。

2.2.3 模组介绍

OV7620 是 OmniVision 公司的彩色/黑白 CMOS 图像传感器。图像输出最高速度可达 60S/s，最大图像分辨率为 644×492，5V 供电；它具有自动增益、自动曝光、自动白平衡、边缘增强、伽玛校正等控制功能；可以通过 IIC 总线进行设置；同

时 OV7620 具有图像开窗输出的功能，即允许用户可根据实际使用需要设置其内部寄存器，使其只输出完整图像中的任意一矩形区域内的信号，其范围从 4×2 到 644×492。这种功能从硬件上屏蔽了图像中不需要的部分，只保留用户需要的部分图像，大大减少了图像的数据量，提高了系统的效率。

AL422B 是由 AverLogic 公司推出的存储容量为 3Mbits 的视频帧存储器，由于目前 1 帧图像信息通常 包含 640×480 或 720×480 个字节， 而市面上很多视频存储器由于容量有限只能存储 1 场图像信息，无法存储 1 帧图像信息。AL422B 由于容量很大，可存储 1 帧图像的完整信息，其工作频率达 50MHz。

OV7620 摄像头与 AL422BFIFO 搭配使用具有以下优点：

- ●功耗低(5V 供电，80mA 电流，功耗低)
- ●噪声少(摄像头数据自动存储在FIFO 芯片中，MCU 主动去读取，将不会因为MCU 与摄像头速度不同步而采集到无效数据)
- ●应用面广（适用于多种MCU）
- ●可配置(用户可以通过IO 随意配置摄像头寄存器已达到自己的理想效果)

2.2.4 OV7620 时序分析

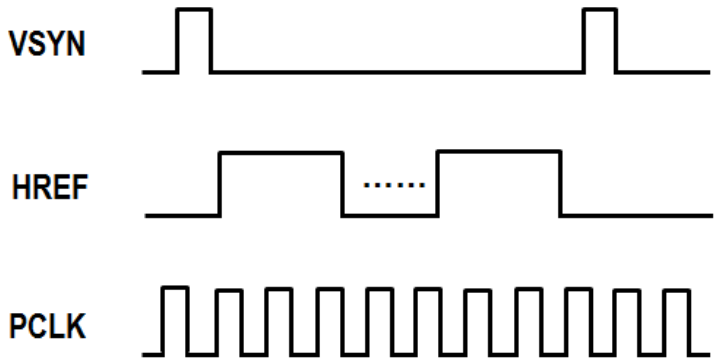

图 2.3 摄像头模组时序图

垂直同步信号 VSYN 为两个正脉冲之间扫描一帧的定时，即完整的一帧图像在两个正脉冲之间；水平同步信号 HREF 扫描该帧图像中各行像素的定时，即高电平时为扫描一行像素的有效时间；像素同步信号 PCLK 为读取有效像素值提供同步信号，高电平时输出有效图像数据，若当前图像窗口大小为 640×480，则在 VSYNC 两个正脉冲之间有 480 个 HREF 的正脉冲，即 480 行；在每个 HREF 正脉冲期间有 640 个 PCLK 正脉冲，即每行 640 个像素。这就是 VSYNC、HREF、PCLK 三个同步信号之间的关系。如图 2.3 中所示。

2.2.5 图像数据自动存储

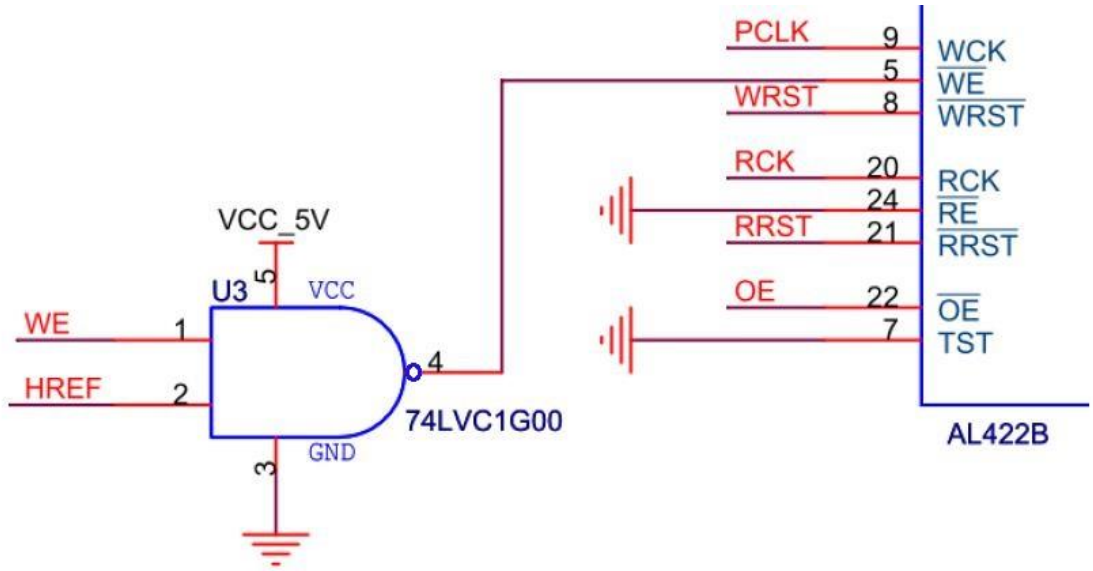

图 2.4　控制图像写入 FIFO 的电路

为了将 OV7620 输出的图像信号自动地存入 FIFO，只需要通过一个“与非门”就能产生符合 FIFO 要求的写时序，如图所示。将帧同步信号 VSYN 引入 MCU 中断输入口，复位后单片机 WE 端口置 0，“与非门”关闭，输出 1。当 MCU 检测到 VSYN 下降沿后，WE 端口置 1，打开“与非门”。当 O76620 输出有效像素时，HREF 为高，则 FIFO 的 WE 为低，写输入全部使能。PCLK 高电平时像素数据有效，PCLK 接 WCK，根据 PCLK 的上升沿，即 WCK 的上升沿，触发 FIFO 锁存 OV7620 输出的图像数据。

2.2.6 OV7620 寄存器设置

如果摄像头采集的分辨率过大会增大数据采集时间，减慢程序的实时响应，另外过大的窗口很有可能会采集到跑道以外的不必要信息，所以摄像头初始化时需要设定图片的采集格式和开窗大小。

OV7620 可以输出两种图像格式，即 VGA（640*480）和 QVGA（320*240），采集跑道信息时不需要太高分辨率，所以将 OV7620 的数据格式设置为 QVGA，根据手册资料设置 0x14（COMC）寄存器为 0x24 即可实现。

开窗大小主要通过 OV7620 的四个寄存器来设置，即 0x17（HS），0x18（HE），0x19（VS），0x20（VE），如下图所示，四个寄存器分表表示上下左右的窗口位置。QVGA 格式的窗口大小计算方法如下：水平窗口= (Register [18] - Register [17])*2，垂直窗口= (Register [1A] - Register [19]+ 1)。程序设定开窗大小时，让开窗中心保持不变，即上下或左右变化的大小一致。图像采集时最后两列图像会采集到消隐信号，所以开窗设定时在指定窗口大小基础上增加了两列的数据采集，通过软件将最后两列的消隐信号去掉。

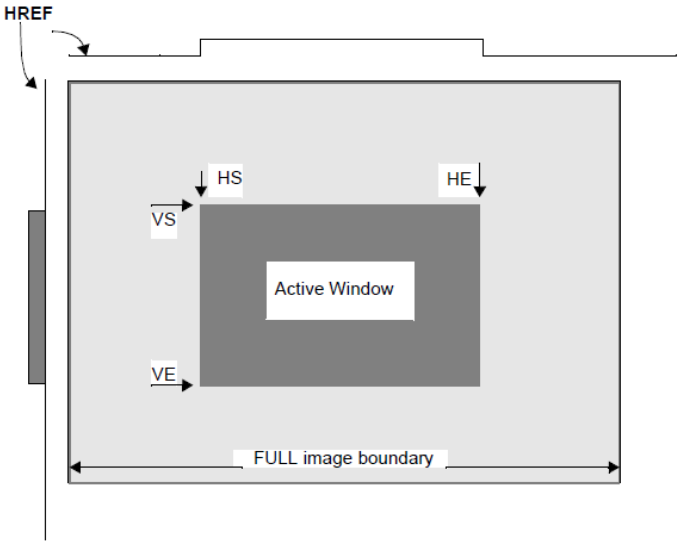

图 2.5　OV7620 开窗示意图

2.2.7 模块连接

本例程采用的 CMOS 摄像头模块 OV7620 搭配 FIFO 模块 AL422 与 R5F100LEA 引脚连接如下：

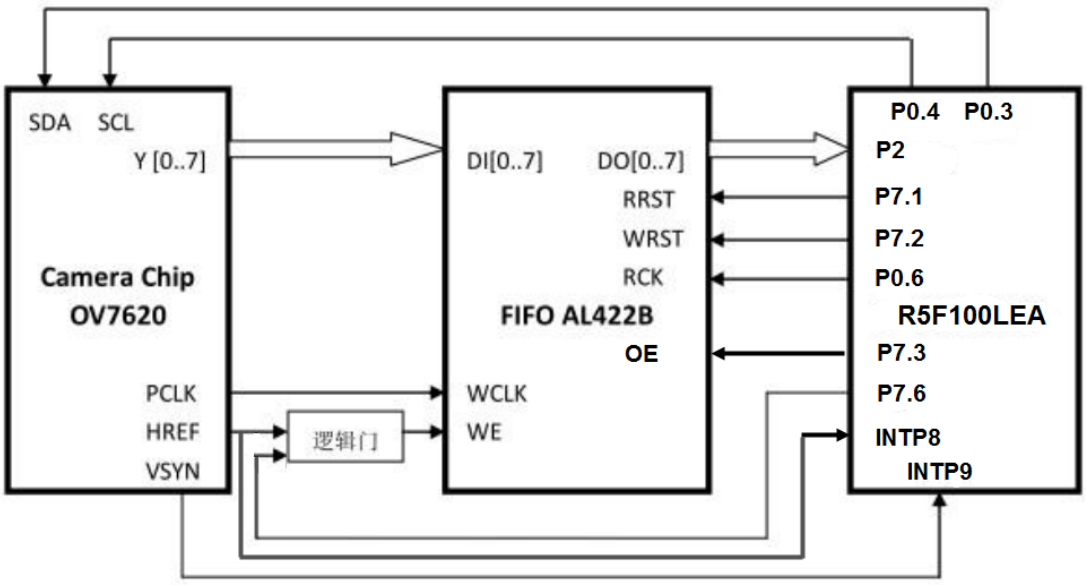

图 2.6　模块连接图

2.2.8 图像读取方法

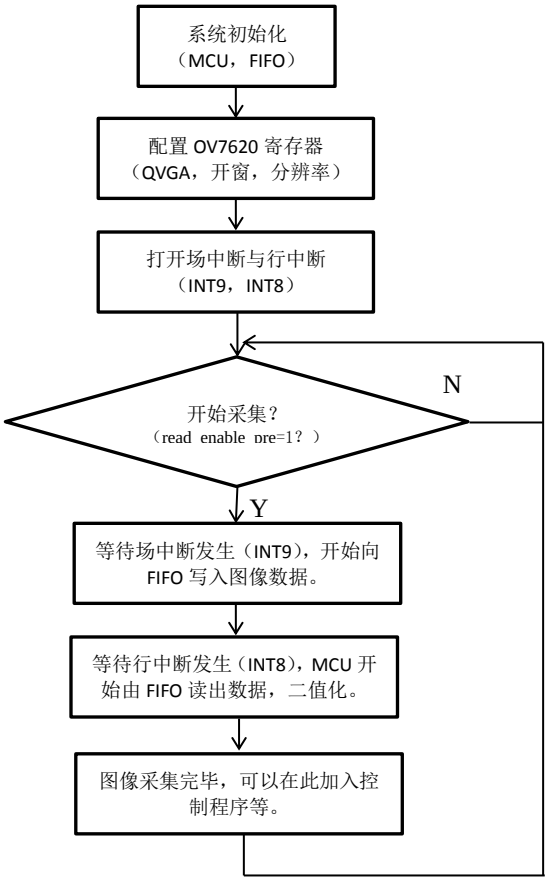

图 2.7　软件控制逻辑流程图

软件控制逻辑按照如上所示流程图，当控制变量 read_enable_pre=1 的时候，等待场中断发生即开始向 FIFO 写入图像数据，FIFO 中写入一幅图像的第一行数据后，行中断发生，MCU 便开始读取数据进行二值化处理（如果读追上写，则读等待，等待 FIFO 中再次被写入一行图像数据时 MCU 开始读数据）。当一幅图像数据全部写入 FIFO 后关闭 FIFO 写使能。当一幅图像的数据全部读出后即二值化处理也已完成，MCU 可以开始进行其他控制操作，当控制完成后，如果再次需要采集图像，需要重新设置控制变量 read_enable_pre=1，以此循环。

2.2.9 软件滤波算法

★本算法为附加功能，因为加入滤波后会加大 CPU 运算时间，使得 MCU 采集一幅图像的时间变长，所以本功能可根据实际情况选择使用。

由于二值化后的图像存在着一些噪点，这些噪点对赛道识别与后续的算法有一些干扰。所以对二值化进行中值滤波，该方法是一种局部平均的平滑技术，对

脉冲干扰和椒盐噪声的抑制效果好，能有效保护图像的边。由于 3*3 中值滤波模板算法会过多的占用 CPU，使得 CPU 使用效率下降，而且 3*1 中值滤波可以滤掉大部分的噪点，所以本例程采取 3*1 窗口的简化中值滤波，即每个像素点与相邻两个点相加的均值来判断该点的像素。

由于采用滤波算法会降低 CPU 的使用效率，导致画面采集率降低，所以此功能只作为附加功能使用，根据自身的需要使用。

2.2.10 黑线中心提取方法

★本例程中并没有加入该功能，本部分说明仅供参考。

提取黑线中心部分是小车路径识别系统中最为重要的一个环节，关系到智能小车运行质量的好坏。

这里黑线中心提取方法为：先判断每行的第一个点是否为白点，如是白点则依次对白点进行计数(设计数为 a)，当遇到连续黑点时则计数黑点个数(设计数为 b)，再次遇到白点时则退出该行计数，此时黑线中心所在列为 a+b / 2；如果第一个为黑点，且不是噪声点(即为连续的黑点)，则直接对黑点计数(设计数为 b)，当遇到连续白点后则退出计数，这种情况下得到的黑线中心位于第 b / 2 列。最后将从一幅图像上得到的所有中心位置按行存人一个一维数组中。

但是不得不考虑两种比较特殊的情况，起始线和十字交叉线。通过观察发现这两种情况下每行出现黑点的数量远远大于黑线的黑点数(黑线一般能采集到 2—3 个黑点)，所以当采集到的黑点数超过正常值时，便判定这一行为特殊行，并赋予特殊标志位。

虽然已经有去噪处理，但是偶尔还是会有干扰，这样提取出来的某些行黑线中心位置就发生了跳变。同时，为了后续的控制模块得到准确的道路信息，需要对特殊行及跳变点进行插值处理，即赋予其前后两行的平均值作为黑线中心值。但是考虑到会出现这样的复杂情况：连续两行出现跳变点或者特殊行的前后出现跳变点，这时如果简单的按上面的方法进行插值，会插入一个误差很大的中心点，使道路产生弯点。于是，在这里需要对差值算法进行改进。图 2.8 为改进算法的流程图。改进算法考虑了连续两个中心点需要插值的情况，先将当前点与前一个点比较，计算其偏差值，如果偏差较大则为需要插值的点；接下来计算前一个点与当前点之后一点的偏差，后面一点如果是误差点，则当前点赋值前一点的值，如果后面一点为正常点则当前点取其前后两点的平均值。

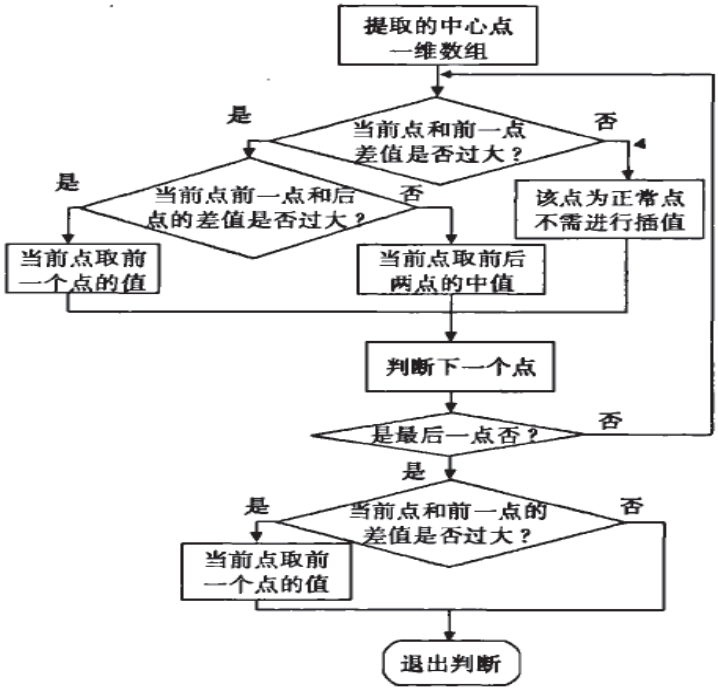

图 2.8 改进差值算法流程图

3. 例程软件说明

3.1 集成开发环境

项目	内容
集成开发环境	CubeSuite+ V2.00.00
C 编译器	CA78K0R V1.60
调试器	RL78 EZ Emulator

3.2 CMOS 摄像头循迹例程编写

下面介绍如何使用代码生成工具 CubeSuite+配置本例程相关模块，自动生成例程驱动代码，并在生成的驱动代码中添加用户拓展驱动程序。

3.2.1 CubeSuite+界面配置

首先参照[RL78/G13 开发套件快速入门教程]安装 CubeSuite+及相关组件，启动 CubeSuite+，新建 G13 下 R5F100LEA 工程。

注意：刚刚启动 CubeSuite+时，点击菜单栏中的 tool 选项，在 Plug-in Setting 中勾选上 Code Generator Plug-in 并重启软件。（参照[RL78/G13 开发套件快速入门教程]）

R5F100LEA 工程界面具体设置如下：

（一）. 配置 Clock Generator 模块

Pin assignment-默认设置

Clock setting– 默认设置 (使用内部振荡器 fMAIN=fIH=32MHz)

On-chip debug setting- On-chip debug operation setting 选择 used(使用片上调试)

如图 3-1 Pin assignment 界面设置，3-2 Clock setting 界面设置

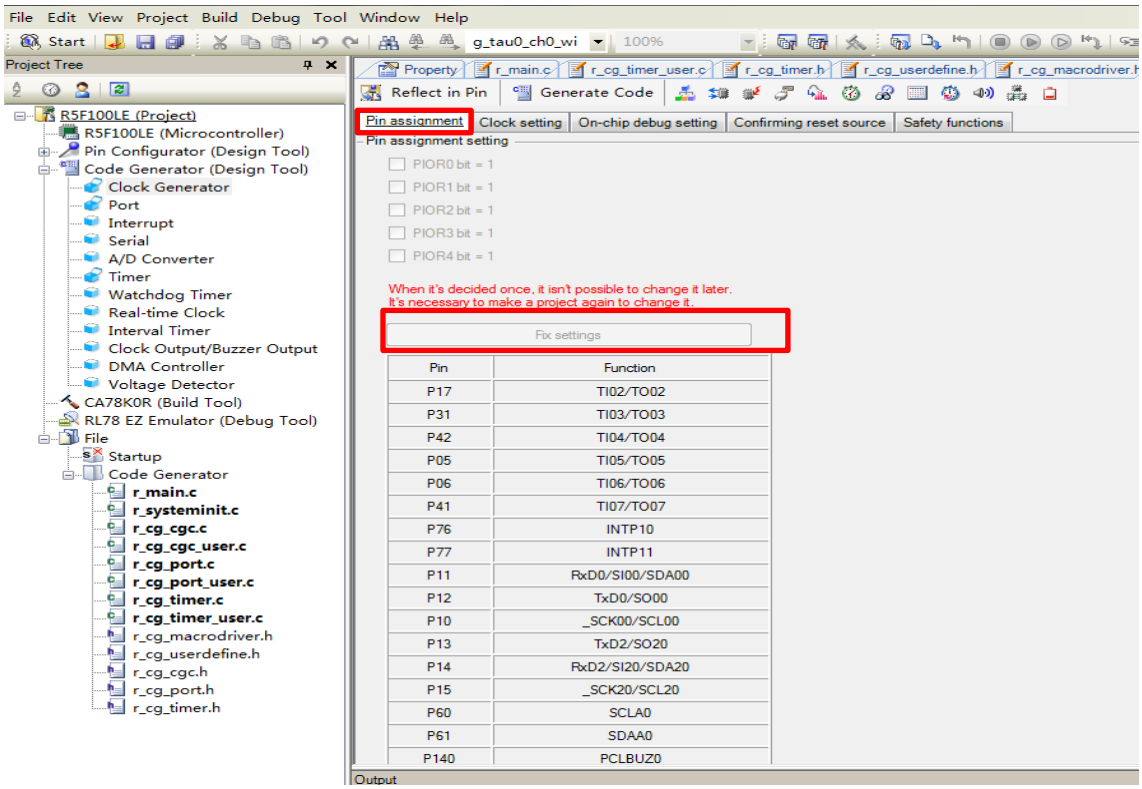

图 3.1 Pin assignment 界面设置

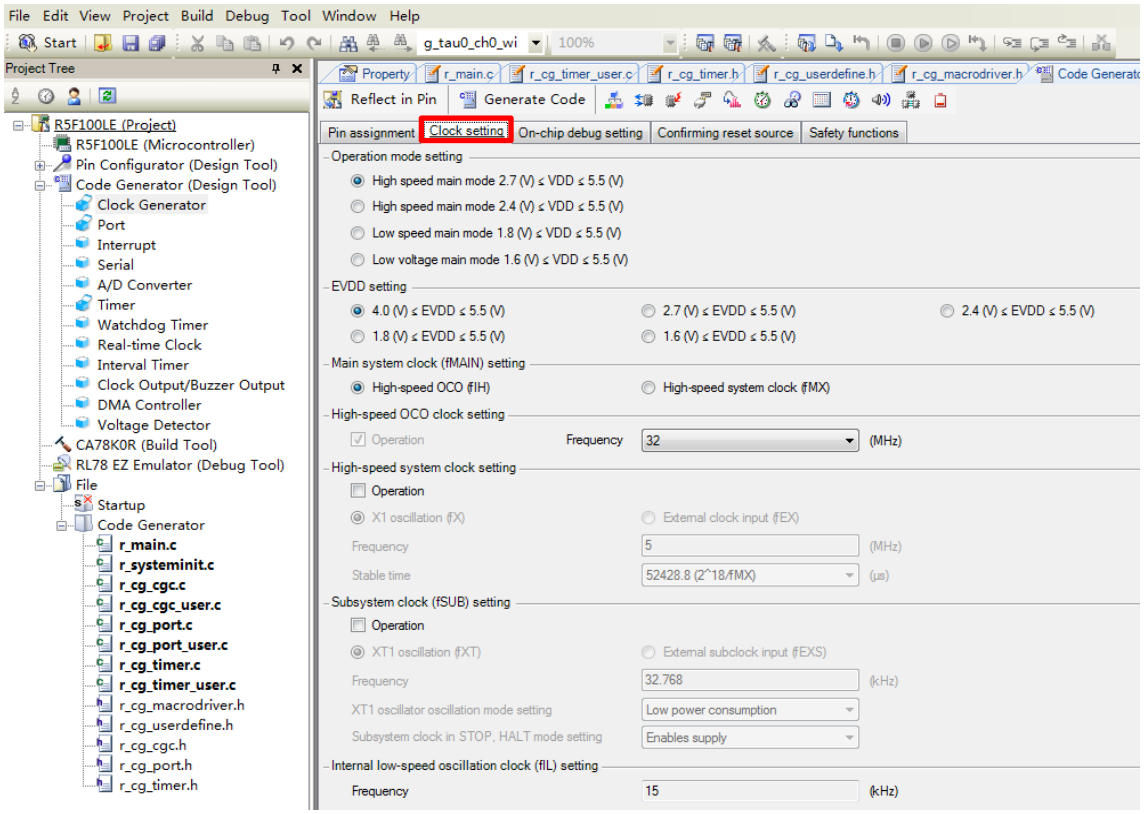

图 3.2　Clock setting 界面设置

（二）．配置 Port 模块

1.PORT0

Port0-P03 使用 SDA10 需要勾选 N-ch

Port0-P04 使用 SCL10 需要勾选 N-ch

Port0-P06 作为输出使用需要选 Out 。P06 与 FIFO 的 RCK 连接，作为 read clock。

如图 3-3 Port0 界面设置

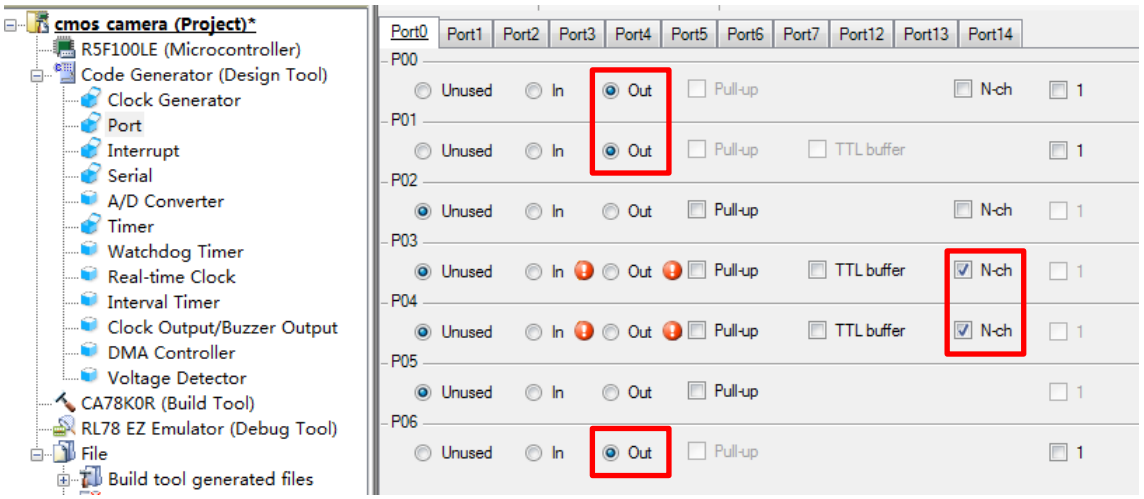

图 3.3　Port0 界面设置

注：红点代表引脚冲突，P03，P04 对应的引脚在本例程中被简易 IIC 模块的 SDA10 和 SCL10 占用，所以这两个引脚不能选择 In、Out。

2.PORT1

★TxD0 用于 UART 口发送数据到上位机，方便调试，所以 P12 口只对调试有关。

如图 3-4 Port1 界面设置

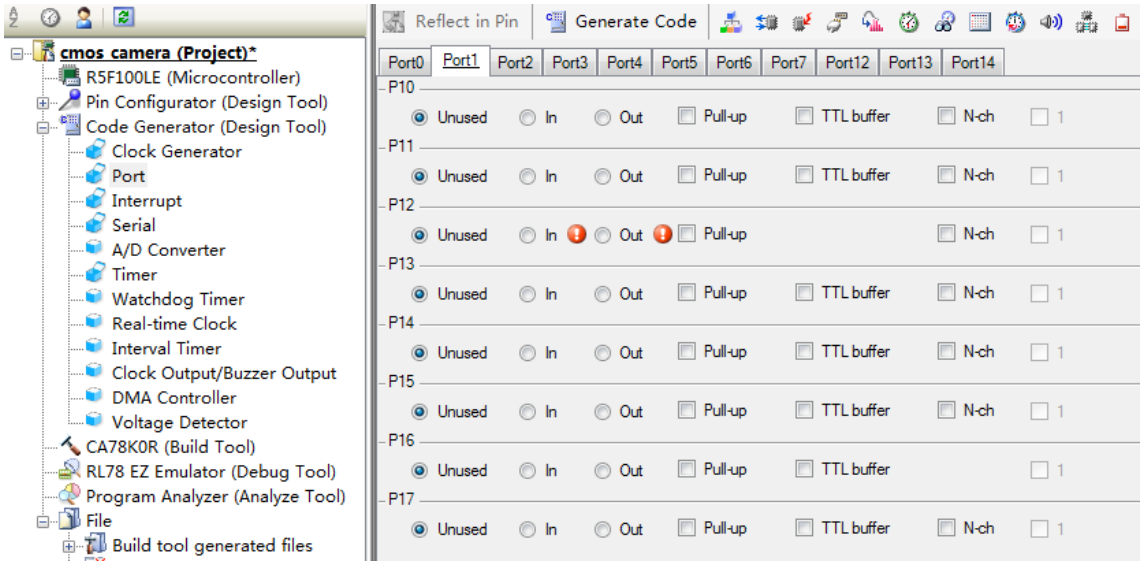

图 3.4　Port1 界面设置

注：红点代表引脚冲突，P12 对应的引脚在本例程中被 UART 模块的 TxD0 占用，所以这两个引脚不能选择 In、Out。

3.PORT2

Port2-P20~P27 作为输入使用需要选 In。P2 与 FIFO 的读数据总线连接。

如图 3-5 Port2 界面设置

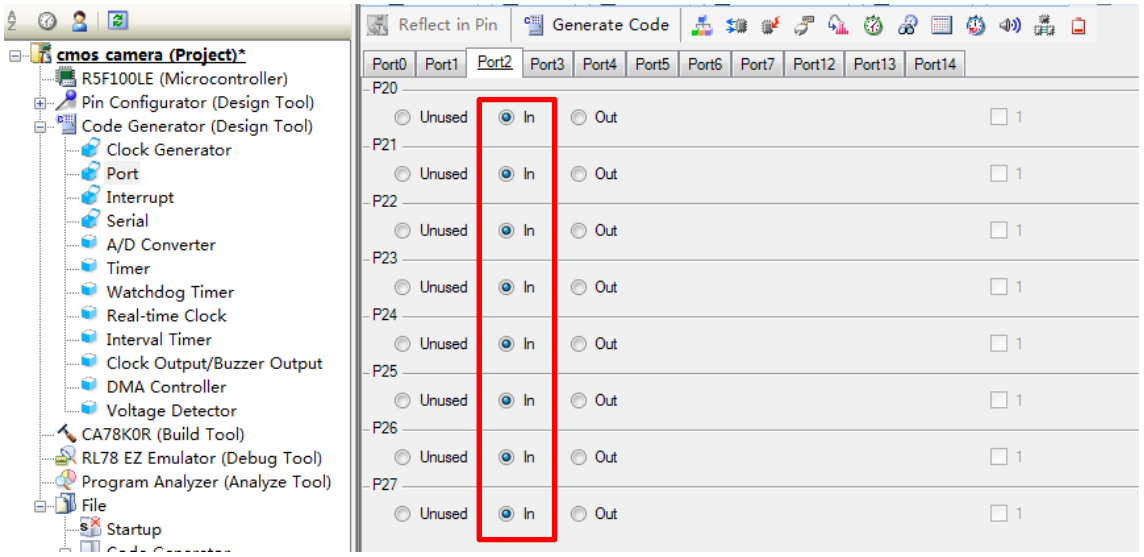

图 3.5　Port2 界面设置

4.PORT7

Port7-P71~P73、P76 作为输出使用需要选 Out。P7 连接关系如下：

P71----RRST，P72----WRST，P73----OE，P74----HREF，P75----VSYNC，P76----WE。

如图 3-6 Port7 界面设置

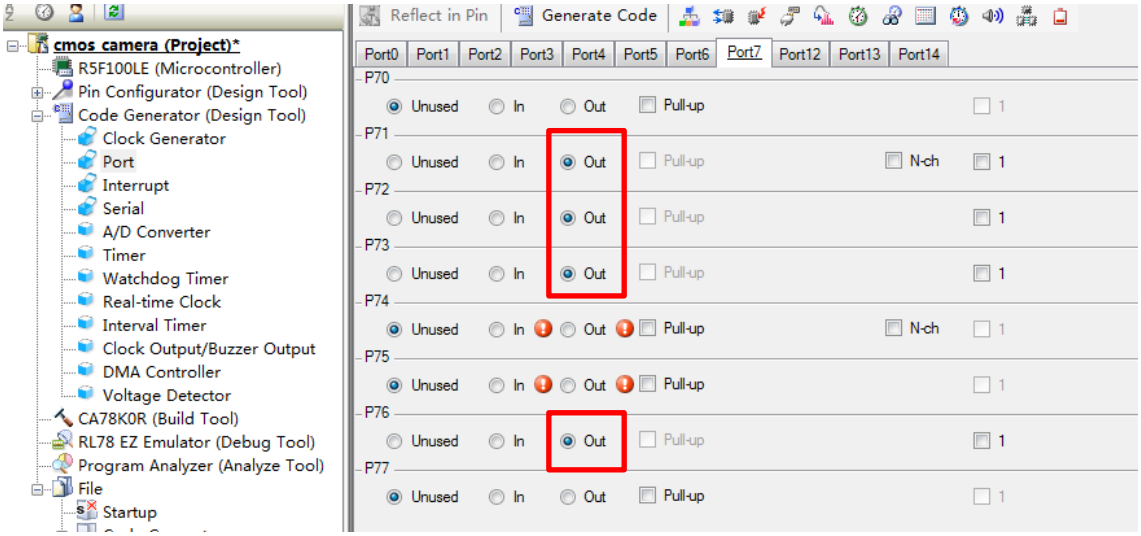

图 3.6　Port7 界面设置

注：红点代表引脚冲突，P74、P75 对应的引脚在本例程中被 INT 模块的 INTP8、INTP9 占用，所以这两个引脚不能选择 In、Out。

5.PORT13

Port13-P130 作为输出使用需要选 Out。测试时使用端子，可以不配置。

如图 3-7 Port13 界面设置

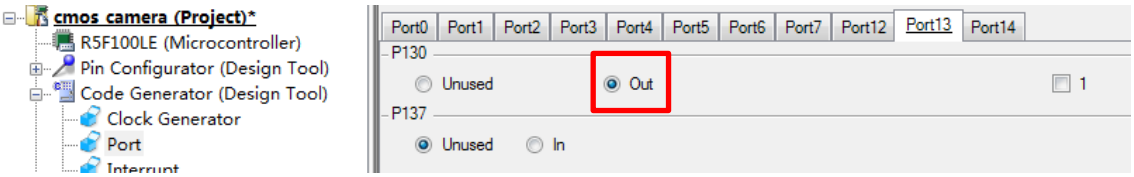

图 3.7　Port13 界面设置

（三）．配置 Serial 模块

Channel 0 中选择 UART0/Transmit function—UART0 tab 中:

1. 选择 Continuous transfer mode；
2. Baudrate 选择 38400（bps）。

Channel 2 中选择 IIC10—IIC10 tab 中:去掉 Master error。

如图 3-8 Serial 界面设置,3-9 UART0 tab 界面设置,3-10 IIC10 tab 界面设置

★TxD0 用于 UART 口发送数据到上位机，方便调试，所以上面设定的 P12 口只对调试有关，不调试时可以不使用。

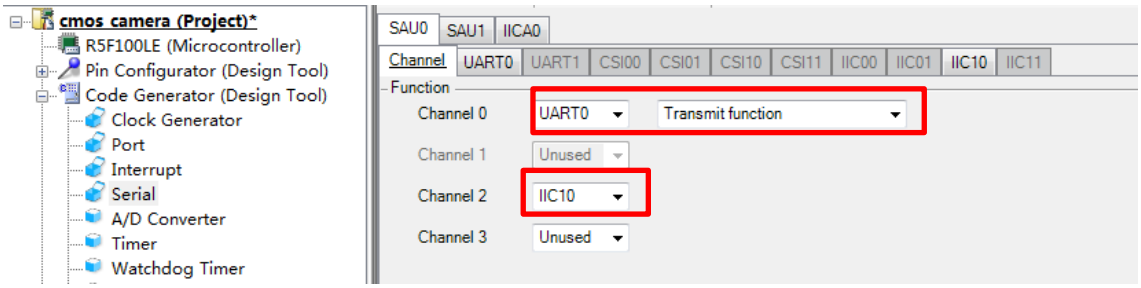

图 3.8　Serial 界面设置

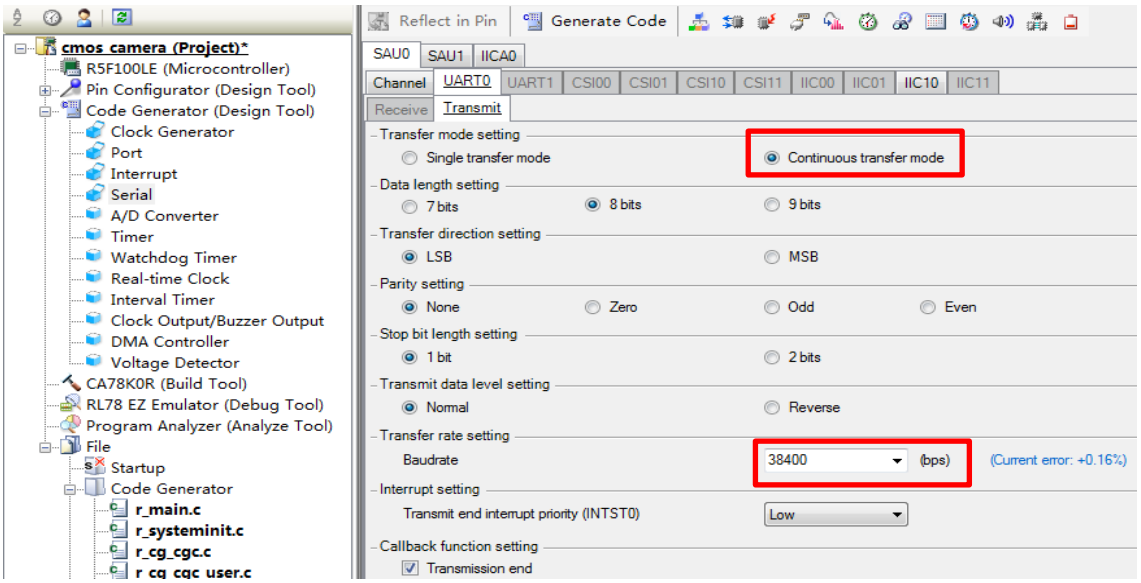

图 3.9　UART0 tab 界面设置

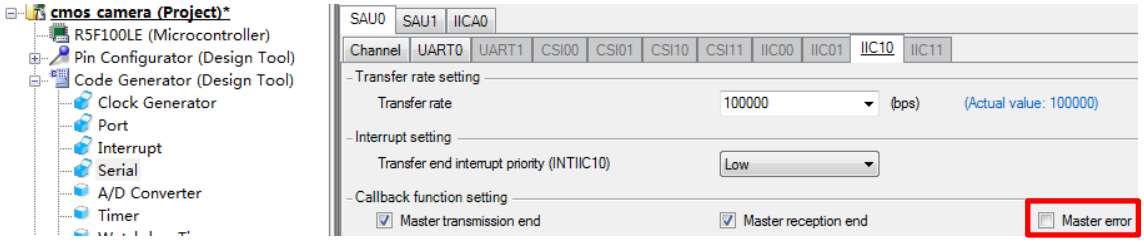

图 3.10　IIC10 tab 界面设置

（四）．配置 Interrupt 模块

在External Interrupt tab中：

勾选INTP8-Valid edge选择Falling-Priority选择High；

勾选INTP9-Valid edge选择Falling-Priority选择High；

如图 3-11 Interrupt-External Interrupt tab 界面设置。

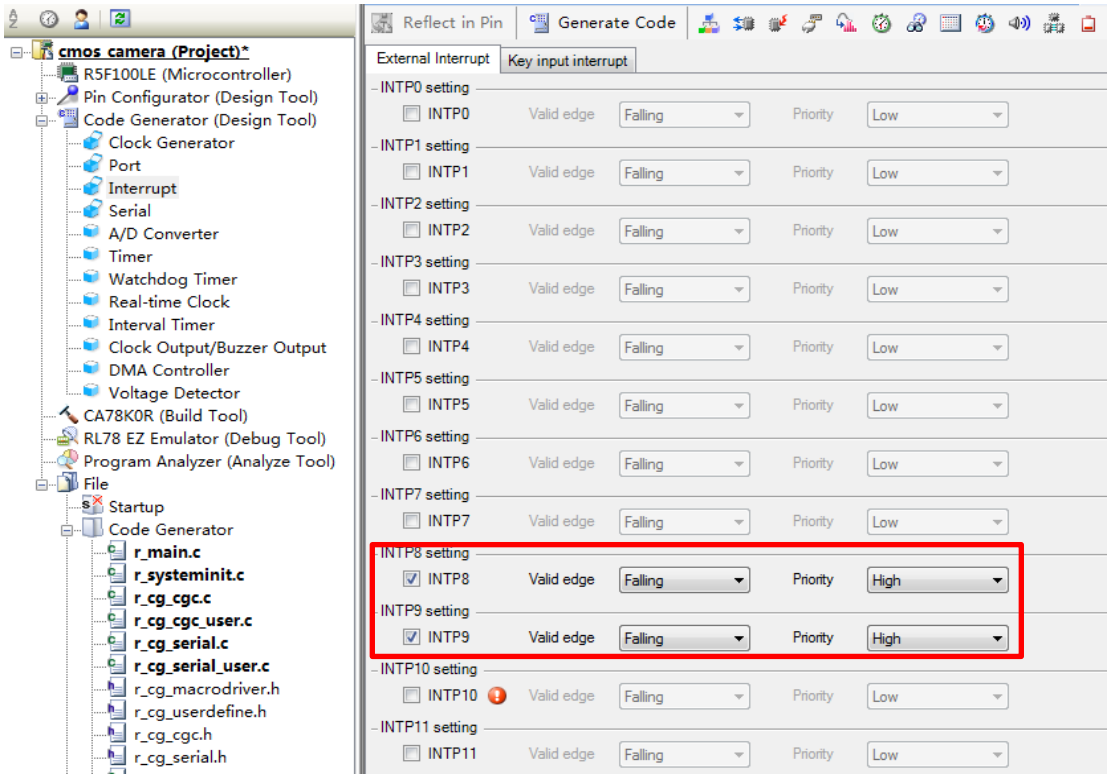

图 3.11 Interrupt-External Interrupt tab 界面设置

注：红点代表引脚冲突，INTP10 所在引脚 P76 在本例程中被用作输入，所以不能使用 INTP10。

（五）．配置 Watchdog Timer 模块

由于本例程采用片上调试，需要把看门狗模块关闭，以便在程序中加断点查看变量的值。

Watchdog timer operation setting 选择 unused

如图 3-12

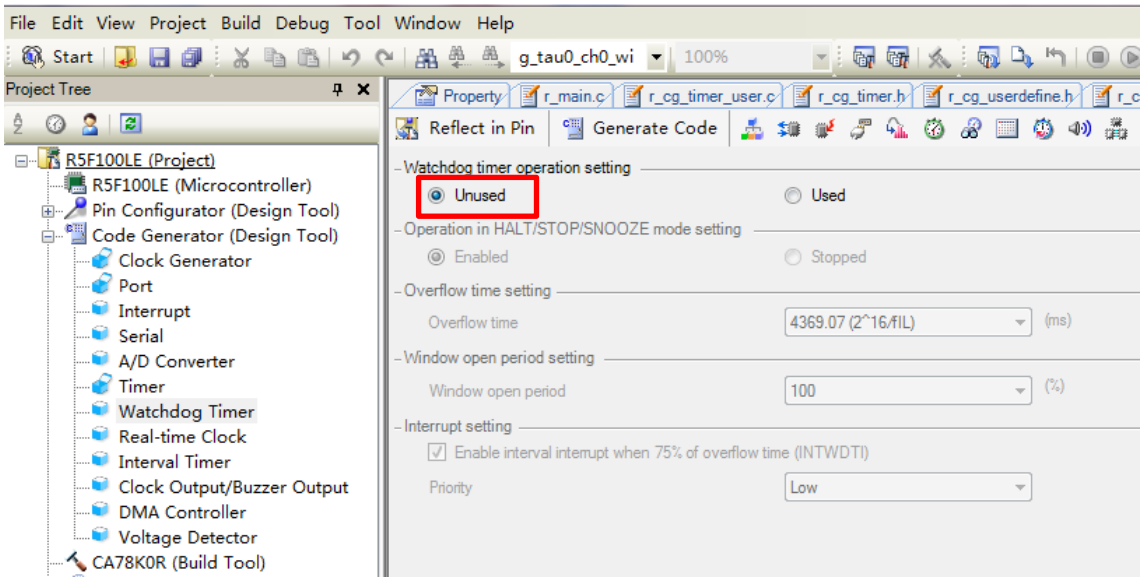

图 3.12　Watchdog Timer 界面设置

（五）．生成驱动代码

所有模块配置完成之后，点击 Generate Code 按钮，生成各模块驱动代码。

如图 3-13 生成驱动代码

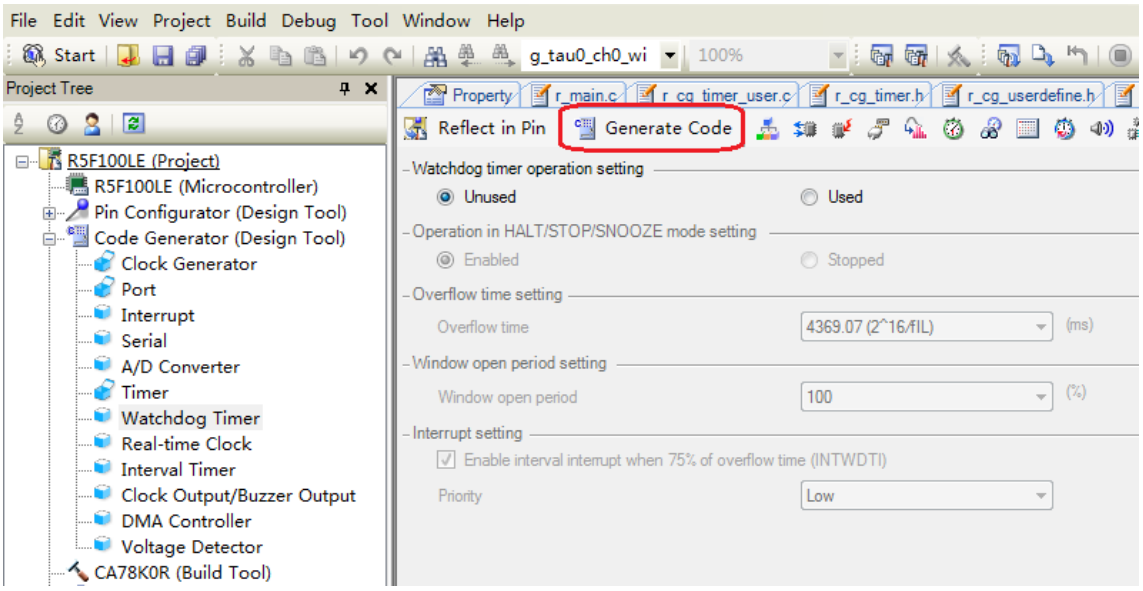

图 3.13　生成驱动代码

3.2.2 添加摄像头模块程序

生成模块驱动代码之后，需要用户根据自身需要添加拓展模块的驱动以及一些自定义功能代码。需要注意的是，自身添加的代码需要写在固定区域，以便每次生成代码的时候不会被冲掉。这里仅列出了函数添加区域，其他的类似。同时，添加的注释尽量是英文，否则每次生成代码之后注释会出现乱码。

如图 3-14 用户添加代码区域

```
/* Start user code for adding. Do not edit comment generated here */

/************************************************************************
 * you can add your program here.
 ************************************************************************/

/* End user code. Do not edit comment generated here */
```

图 3.14　用户添加代码区域

本例程中需要用户自己添加修改的函数分为三部分，分别为 main 函数，中断函数，摄像头采集图像的自定义函数。添加了两个自定义文件，r_camera.c 与 r_camera.h 用来存放摄像头采集图像的自定义函数。

（一）摄像头程序使用定义说明

在 r_ camera.h 中添加如下定义：

1. WE：配置 P7.6 端口为 WE；
2. OE：配置 P7.3 端口为 OE；

3. RCK：配置 P0.6 端口为 RCK；
4. WRST：配置 P7.2 端口为 WRST；
5. RRST:配置 P7.1 端口为 RRST；
6. CAMERA_DATA:配置 P2 端口为 CAMERA_DATA;
7. H_RESOLUTION:配置开窗每行像素点的个数；[注 1]
8. V_RESOLUTION:配置开窗行数；[注 1]
9. H_STEP:配置水平分辨率使用（详见 4.3 分辨率测试结果）;
10. V_STEP:配置垂直分辨率使用（详见 4.3 分辨率测试结果）;
11. THRESHOLD:配置二值化的阈值；
12. ARRAY_SIZE:配置二值化后 RAM 存储空间的大小。
13. DISPLAY：调试用，如果打开显示功能，即把从 FIFO 读出的数据通过 UART 传送到上位机显示。

注 1：在不使用滤波函数时，需要满足 H_RESOLUTION*V_RESOLUTION 是 8 的倍数，在使用滤波函数时，需要满足 H_RESOLUTION 是 8 的倍数。

注 2：7~13 可根据用户需求修改配置。

如图 3.15

```
/*****************************************************************************************
Macro definitions
*****************************************************************************************
#define WE      P7.6
#define OE      P7.3
#define RCK     P0.6
#define WRST    P7.2
#define RRST    P7.1
#define CAMERA_DATA     P2
#define H_RESOLUTION 120  //maximum:320,multiple of 4
#define V_RESOLUTION 64   //maximum:240,multiple of 2
#define H_STEP 1    //for change Horizontal resolution
#define V_STEP 1    //for change Vertical resolution
#define THRESHOLD 0x96
#define ARRAY_SIZE 3000

#define DISPLAY           //using windows software to display pictures grabbed by camera
/*****************************************************************************************
Typedef definitions
*****************************************************************************************
```

图 3.15　r_camera.h 关于定义的代码

（二）摄像头采集图像的自定义函数说明

如图 3.16 中是本例程在 r_ camera.h 中添加的自定义函数声明。

```
/*****************************************************************************
Global functions
*****************************************************************************
void User_Delay(int);
void R_FIFO_Create(void);
void R_FIFO_Write_Reset(void);
void R_FIFO_Read_Reset(void);
void R_Picture_Grab_Gray(void);
void R_Picture_Grab_Bin(void);
void R_Picture_Grab_Smoothing(void);
void R_OV7620_Configure(void);
/* Start user code for function. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#endif
```

图 3.16　自定义函数一览

下面对图 3.16 中声明的函数即 r_camera.c 中定义的子函数进行说明。

1. **User_Delay**

本函数式自定义延时函数，用于配置摄像头以及初始化 FIFO 读写时的延时等待。延时函数描述如图 3.17。

```
/*****************************************************************
* Function Name: User_Delay
* Description  : CPU will wait for a period of time according to i;
* Arguments    : i -
*                    delay time
* Return Value : None
*****************************************************************
void User_Delay(int i)
{
        int j,k;
        for(j=i;j--;j>0)
                for(k=100;k--;k>0);
}
```

图 3.17　User_Delay 函数描述

2. **R_FIFO_Create**

本函数是 FIFO 初始化函数，把读写使能关闭，然后进行读写复位。如图 3.18。

```
/*****************************************************
* Function Name: R_FIFO_Create
* Description  : This function initialize the fifo.
* Arguments    : None
* Return Value : None
*****************************************************
void R_FIFO_Create(void)
{
        WE = 0U;                //disable write
        OE = 1U;                //disable read
        R_FIFO_Write_Reset();
        R_FIFO_Read_Reset();
}
```

图 3.18　R_FIFO_Create 函数描述

3. **R_FIFO_Write_Reset**

本函数是 FIFO 写复位函数，用来复位 FIFO 的写入。如图 3.19。

```
/*****************************************************************
* Function Name: R_FIFO_Write_Reset
* Description  : The data of FIFO input address will be set to 0.
* Arguments    : None
* Return Value : None
*****************************************************************
void R_FIFO_Write_Reset(void)
{
        WRST = 0U;        //enable write reset
        User_Delay(1);
        WRST = 1U;
}
```

图 3.19　R_FIFO_Write_Reset 函数描述

4. R_FIFO_Read_Reset

本函数是 FIFO 读复位函数，用来复位 FIFO 的读出。如图 3.20。

```
/*****************************************************************
* Function Name: R_FIFO_Read_Reset
* Description  : The data of FIFO output address will be set to 0.
* Arguments    : None
* Return Value : None
*****************************************************************
void R_FIFO_Read_Reset(void)
{
        RRST = 0U;        //enable read reset
        User_Delay(1);
        RCK = 1U;
        User_Delay(1);
        RCK = 0U;
        User_Delay(1);
        RRST = 1U;
}
```

图 3.20　R_FIFO_ Read _Reset 函数描述

5. R_Picture_Grab_Gray

本函数用来读取灰度图像数据以供调试用。如果把数据上传到上位机，需要在 r_camera.h 中把#define DISPLAY 前的注释符号去掉，即可以通过串口将图像传到上位机。波特率=38400bps，数据位=8，无校验位，停止位=1。函数描述如图 3.21。

```
/*****************************************************************************************************************
* Function Name: R_Picture_Grab_Gray
* Description  : This function grab gray data of pictures from camera.
* Arguments    : None
* Return Value : None
*****************************************************************************************************************/
void R_Picture_Grab_Gray(void)
{
        unsigned char pixel=0,bin_data=0,count=0;
        unsigned char i_rem,j_rem;
        unsigned int i=0,j=0,k=0;
        unsigned long read_fifo_count=0;

        OE = 0U;                                        //enable FIFO ouput
        R_FIFO_Read_Reset();                            //set the read address to 0

        for(i=1;i<=V_RESOLUTION;i++)                    //read each lines
        {
                for(j=1;j<=H_RESOLUTION+2;j++)          //read each pixels
                {
                    if(write_fifo_count>read_fifo_count) //if write_fifo_count>read_fifo_count you can read the fifo
                    {
                        read_fifo_count++;

                        RCK = 1U;                       //read clock set high

                        if(j<=H_RESOLUTION)             //remove the blanking signal added by hardware
                        {
                            i_rem=i%H_STEP;
                            j_rem=j%V_STEP;
                            if((i_rem==0)&(j_rem==0))    //change resolution by software
                            {
                                pixel = CAMERA_DATA;    //read the value of D[7:0]
                                #ifdef DISPLAY
                                  R_UART0_Send_Data(&pixel,1);
                                #endif
                             }
                        }

                        RCK = 0U;
                     }
                     else{j--;}
                }
        }
        OE = 1U;
        write_fifo_count = 0;
        read_fifo_count = 0;
}
```

图 3.21　R_Picture_Grab_Gray 函数描述

6. R_Picture_Grab_Bin

本函数用来读取二值化图像数据，并将二值化结果存于 RAM 中。如果把数据上传到上位机，需要在 r_camera.h 中把#define DISPLAY 前的注释符号去掉，即可以通过串口将图像传到上位机。波特率=38400bps，数据位=8，无校验位，停止位=1。函数描述如图 3.22。

```
/*****************************************************************************************************************
* Function Name: R_Picture_Grab_Bin
* Description  : This function grab binary data of pictures from camera.
* Arguments    : None
* Return Value : None
*****************************************************************************************************************/
void R_Picture_Grab_Bin(void)
{
        unsigned char pixel=0,bin_data=0,count=0;
        unsigned char i_rem,j_rem;
        unsigned int i=0,j=0,k=0;
        unsigned long read_fifo_count=0;

        OE = 0U;                                        //enable FIFO ouput
        R_FIFO_Read_Reset();                            //set the read address to 0

        for(i=1;i<=V_RESOLUTION;i++)                    //read each lines
        {
                for(j=1;j<=H_RESOLUTION+2;j++)          //read each pixels
                {
                    if(write_fifo_count>read_fifo_count) //if write_fifo_count>read_fifo_count you can read the fifo
                    {
                        read_fifo_count++;

                        RCK = 1U;                       //read clock set high
```

```
                    if(j<=H_RESOLUTION)             //remove the blanking signal added by hardware
                    {
                        i_rem=i%H_STEP;
                        j_rem=j%V_STEP;
                        if((i_rem==0)&(j_rem==0))     //change resolution by software
                        {
                              pixel = CAMERA_DATA;    //read the value of D[7:0]

                              if(pixel >= THRESHOLD)
                                 bin_data|=0x01;
                              else
                                 bin_data&=0xfe;
                              count++;
                              if(count > 7)
                              {
                                #ifdef DISPLAY
                                  R_UART0_Send_Data(&bin_data,1);//binary data to host to use URAT0
                                #endif
                                count = 0;
                                if(k<ARRAY_SIZE)
                                  picture[k++]=bin_data; //binary data to memory
                              }else{
                                bin_data=bin_data<<1;
                              }
                          }
                    }

                    RCK = 0U;
                  }
                  else{j--;}
              }
        }
        OE = 1U;
        write_fifo_count = 0;
        read_fifo_count = 0;
}

/*****************************************************************************************************************************/
```

图 3.22　R_Picture_Grab_Bin 函数描述

7. R_Picture_Grab_Smoothing

本函数用来读取滤波后的二值化图像数据。如果把数据上传到上位机，需要在 r_camera.h 中把#define DISPLAY 前的注释符号去掉，即可以通过串口将图像传到上位机。波特率=38400bps，数据位=8，无校验位，停止位=1。函数描述如图 3.23。

二值化滤波算法采用临近 3 点比较方式，3 点一致则数据有效，否则认为是无效数据。

```
/*****************************************************************************************************************************
* Function Name: R_Picture_Grab_Smoothing
* Description  : This function grab binary data of pictures from camera.
* Arguments    : None
* Return Value : None
*****************************************************************************************************************************/
void R_Picture_Grab_Smoothing(void)
{
        unsigned char pixel=0,bin_data=0,count=0,last_num=0x00,current_num=0x00,next_num=0x00;
        unsigned char i_rem,j_rem;
        unsigned int i=0,j=0,k=0;

        OE = 0U;                                    //enable FIFO ouput
        R_FIFO_Read_Reset();                        //set the read address to 0

        for(i=1;i<=V_RESOLUTION;i++)                //read each lines
        {
                for(j=1;j<=H_RESOLUTION+2;j++)      //read each pixels
                {
                        RCK = 1U;                   //read clock set high

                        if(j<=H_RESOLUTION+1)       //remove the blanking signal added by hardware
                        {
                            i_rem=i%H_STEP;
                            j_rem=j%V_STEP;
                            if((i_rem==0)&(j_rem==0))     //change resolution by software
                            {
                                pixel = CAMERA_DATA;    //read the value of D[7:0]

                                  if(j<=H_RESOLUTION)
                                  {
```

```
                                        if(pixel >= THRESHOLD)
                                           next_num = 0x01;
                                        else
                                           next_num = 0x00;
                                    }else{
                                        next_num = 0x01;
                                    }
                                    if(j-1)
                                    {
                                        switch(current_num)
                                        {
                                                case 0x00:
                                                  if(last_num&next_num)
                                                     bin_data|=0x01;
                                                  else
                                                     bin_data&=0xfe;break;
                                                case 0x01:
                                                  if(last_num|next_num)
                                                     bin_data|=0x01;
                                                  else
                                                     bin_data&=0xfe;break;
                                        }

                                        count++;
                                        if(count > 7)
                                        {
                                          #ifdef DISPLAY
                                            R_UART0_Send_Data(&bin_data,1);//binary data to host to use URAT0
                                          #endif
                                          count = 0;
                                          if(k<ARRAY_SIZE)
                                            picture[k++]=bin_data; //binary data to memory
                                        }else{
                                          bin_data=bin_data<<1;

                                         }
                                         last_num = current_num;
                                    }else{
                                         last_num = 0x01;
                                    }
                                    current_num = next_num;
                                }
                            }

                            RCK = 0U;
                        }
                    }
                    OE = 1U;
                }
/*****************************************************************************************************************************
```

图 3.23　R_Picture_Grab_Smoothing 函数描述

8. R_OV7620_Configure

本函数用来配饰 OV7620 摄像头，通过简易 IIC 总线向摄像头模块中的寄存器写入配置信息。本例成只配置摄像头为 QVGA 输出并配置开窗大小，其他配置信息请参考 OV7620 用户手册。函数描述如图 3.24。

```
/***********************************************************************************
* Function Name: R_OV7620_Configure
* Description  : The function is config the register of OV7620.
* Arguments    : None
* Return Value : None
***********************************************************************************/
void R_OV7620_Configure(void)
{
    unsigned char command[2] = {0x00,0x00};
    unsigned char h_dec = (320-H_RESOLUTION)/4;
    unsigned char v_dec = (240-V_RESOLUTION)/2;

    command[0] = 0x14;
    command[1] = 0x24;
    R_IIC10_Master_Send(0x42, command, 2U);  // change to QVGA mode
    User_Delay(1000);

    command[0] = 0x17;
    command[1] = 47+h_dec;
    R_IIC10_Master_Send(0x42, command, 2U);  // set Horizontal Window start
    User_Delay(1000);

    command[0] = 0x18;
    command[1] = 208-h_dec;
    R_IIC10_Master_Send(0x42, command, 2U);  // set Horizontal Window end
    User_Delay(1000);

    command[0] = 0x19;
    command[1] = 6+v_dec;
    R_IIC10_Master_Send(0x42, command, 2U);  // set Vertical Window start
    User_Delay(1000);

    command[0] = 0x1A;
    command[1] = 245-v_dec;
    R_IIC10_Master_Send(0x42, command, 2U);  // set Vertical Window end
    User_Delay(1000);


}
```

图 3.24　R_OV7620_Configure 函数描述

（三）中断函数说明

如图 3-25 中断函数中变量定义，图 3.26 中断函数描述

Line：行写入计数用参数。控制 FIFO 写入完成。

WriteFlag：控制行中断函数是否执行相应操作用 Flag。

read_enable_pre：FIFO 写开始控制变量。

read_enable：FIFO 读开始控制变量。

write_fifo_count：FIFO 写入数据计数，读 FIFO 时判断读取数据是否有效用。

```
/***********************************************************************************
Global variables and functions
***********************************************************************************/
/* Start user code for global. Do not edit comment generated here */
unsigned int Line=0;
unsigned char WriteFlag=0; //fifo writeing flag
extern unsigned char read_enable_pre;
extern unsigned char read_enable;
extern unsigned long write_fifo_count;      //write count
/* End user code. Do not edit comment generated here */

/***********************************************************************************
```

图 3.25　中断函数变量定义

下面对中断函数进行简单说明。r_intc9_interrupt 为场中断函数，该函数控制图像数据是否写入 FIFO。r_intc8_interrupt 为行中断控制函数，该函数主要控制 FIFO 中的图像数据何时可以开始读取，以及向 FIFO 写一帧图像的结束控制。

```
/***********************************************************************************
* Function Name: r_intc8_interrupt
* Description  : This function is INTP8 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************/
__interrupt static void r_intc8_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /*line interrupt*/
     if (WriteFlag)
    {
        Line++;
        write_fifo_count =write_fifo_count + H_RESOLUTION +2; //
        if(Line == 1)               //write fnish a line,start reading.
        {
            read_enable = 1;        //picture can be read
        }
        if(Line == V_RESOLUTION)    //a field of picture finished
        {
            WE = 0U;                //disable write of fifo
            WriteFlag = 0;          //disable WriteFlag.
        }
    }
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************
* Function Name: r_intc9_interrupt
* Description  : This function is INTP9 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************/
__interrupt static void r_intc9_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /*field interrupt*/
    if(read_enable_pre)
    {
        R_FIFO_Write_Reset();     //write the fifo from address 0
        WE = 1U;                  //fifo write enable
        Line = 0;
        WriteFlag = 1;            //fifo writeing flag
        read_enable_pre = 0;      //disable read_enable_pre
    }
    /* End user code. Do not edit comment generated here */
}
```

图 3.26　中断函数描述

（四）main 函数说明

main 函数需要修改部分如图 3.27 全局变量定义部分，图 3.28 main 函数描述部分。

```
/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
unsigned char send_end = 0;             //flag of UART0 data transfer completely
unsigned char read_enable = 0;          //flag of FIFO can be read
unsigned char read_enable_pre = 0;      //flag of FIFO can be write for read(After writing a line, read automatic start)
unsigned long write_fifo_count = 0;      //write count
unsigned char picture[ARRAY_SIZE]={0};  //data of picture store in this array
unsigned int count_test_1S = 0;         //for test
/* End user code. Do not edit comment generated here */
```

图 3.27　全局变量定义

```
/***********************************************************************************************************************
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    R_FIFO_Create();
    R_OV7620_Configure();
    #ifdef DISPLAY
      R_UART0_Start();
    #endif
    R_INTC8_Start();
    R_INTC9_Start();
    //R_TAU0_Channel0_Start();//for test
    read_enable_pre = 1;    //if you want to collect a picture,please set read_enable_pre == 1.
    while (1U)
    {
        if(read_enable)
        {
                #ifdef DISPLAY
                  R_UART0_Display_Start();
                #endif

                //R_Picture_Grab_Smoothing();
                R_Picture_Grab_Bin();
                //R_Picture_Grab_Gray();

                //count_test_1S++;  //for test

                #ifdef DISPLAY
                  R_UART0_Display_End();
                #endif
                read_enable = 0;    //if read finished,please set read_enable == 0.
                /*****************************************************************
                * you can add your program here.
                *****************************************************************/
                read_enable_pre = 1;//if you want to continue collect a picture,please set read_enable_pre == 1.

        }
    }
    /* End user code. Do not edit comment generated here */
}
```

图 3.28　main 函数描述

Main 中调用的图像数据采集读取函数有三个。R_Picture_Grab_Bin();----二值化图像采集函数（不含滤波去噪功能）。R_Picture_Grab_Gray();----灰度图像采集函数，**该函数是为了在上位机显示真实图像用的，实际的二值化图像数据处理时不需要**。R_Picture_Grab_Smoothing();----二值化图像采集函数（含有滤波去噪功能），由于某些黑线中心提取算法中带有滤波功能，故该函数也不一定被用到，用户可以根据需要进行选择使用。

用户可以使用变量 read_enable_pre 来控制图像采集开始。当 read_enable_pre = 1 时，CMOS 摄像头将采集到的当前图像数据开始写入 FIFO，为了提高数据处理速度，当 FIFO 内存入一行图像数据后 MCU 即开始读取数据进行二值化存储。当一幅图像处理完毕后，需要将 read_enable 设置为 0；此时用户可以加入自己的控

制程序，当控制完成后，如果需要读取当前图像，则只需要将 read_enable_pre 再次设置为 1，这样即立刻开始采集当前图像进行处理。如此循环。**两次图像读取的间隔时间决定了图像的刷新速度，请根据实际需要合理分配图像处理和系统应用控制的时间。**

4. 例程测试结果

代码编写完成之后，点击工具栏的 Build->Build Project 或按 F7 快捷键进行编译，确认编译无误后，将 MCU 板和仿真器连接，通过数据线连接到电脑 USB 插口，点击工具栏的 Debug->Build & Download 或按 F6 快捷键将程序下载到单片机中进行片上调试。

注：具体步骤请参照[RL78/G13 开发套件快速入门教程]

本章节对此系统的测试结果进行说明，包括图像采集速率，二值化存储，改变频率分辨率，改变窗口大小，滤波测试。

4.1 图像采集存储速度测试结果

由于 MCU 内部 RAM 最大为 4KB，所以在实际应用时需要根据实际情况配置窗口大小，或者用软件隔点读取的方式改变分辨率等方法减少数据量。表 4.1 是通过设置窗口大小，连续采集图像数据，测得的一组实验结果。表 4.2 是窗口大小恒定，通过设置分辨率，连续采集图像数据，测得的一组实验结果。

表 4.1 改变窗口大小的图像采集存储速度测试结果

图像格式	分辨率	窗口大小	图像采集速率（不含滤波）	采集一幅图像时间（不含滤波）*注	采集一幅图像时间（含滤波）*注	二值化图像 RAM 空间（KB）
QVGA 二值化	320*240	240*100	15S/s	45.95ms	74.35ms	2.93
		160*80	20S/s	23.55ms	39.35ms	1.56
		120*64	30S/s	14.20ms	23.75ms	0.94
		80*40	60S/s	6.05ms	10.05ms	0.39
		40*20	60S/s	1.65ms	2.65ms	0.098
		4*2	60S/s	0.25ms	-	0.001

注：这里的采集一幅图像时间是指，图像有效数据采集、读取、二值化、到一幅图像数据全部存入 RAM 的时间。

表 4.2 改变分辨率的图像采集存储速度测试结果

图像格式	分辨率	窗口大小	图像采集速率（不含滤波）*注 1	采集一幅图像时间（不含滤波）*注 2	采集一幅图像时间（含滤波）*注 2	二值化图像 RAM 空间（KB）
QVGA 二值化	320*240	240*100	15S/s	45.95ms	74.35ms	2.93
	160*120		15S/s	39.65ms	42.35ms	1.47
	80*60		20S/s	33.75ms	34.05ms	0.73

注 1：由于硬件电路已经固定，采取软件降低分辨率时，实际向 FIFO 写入与读出的数据量并没有减少，所以此处降低分辨率后对提高图像采集速率作用不是很明显，但是可以节省 RAM 空间以及扩大图像采集窗口。

注 2：这里的采集一幅图像时间是指，图像有效数据采集、读取、二值化、到一幅图像数据全部存入 RAM 的时间。

4.2 二值化存储测试结果

下面是分辨率=320*240，窗口大小=120*64 的二值化存储测试结果。图 4.1 为串口调试上位机显示结果，图 4.2 位 MCU 进行图像采集，二值化存储的结果。

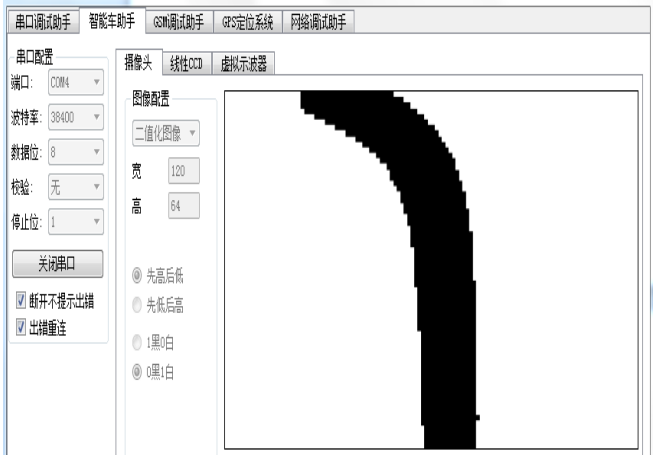

图 4.1　串口调试助手显示结果

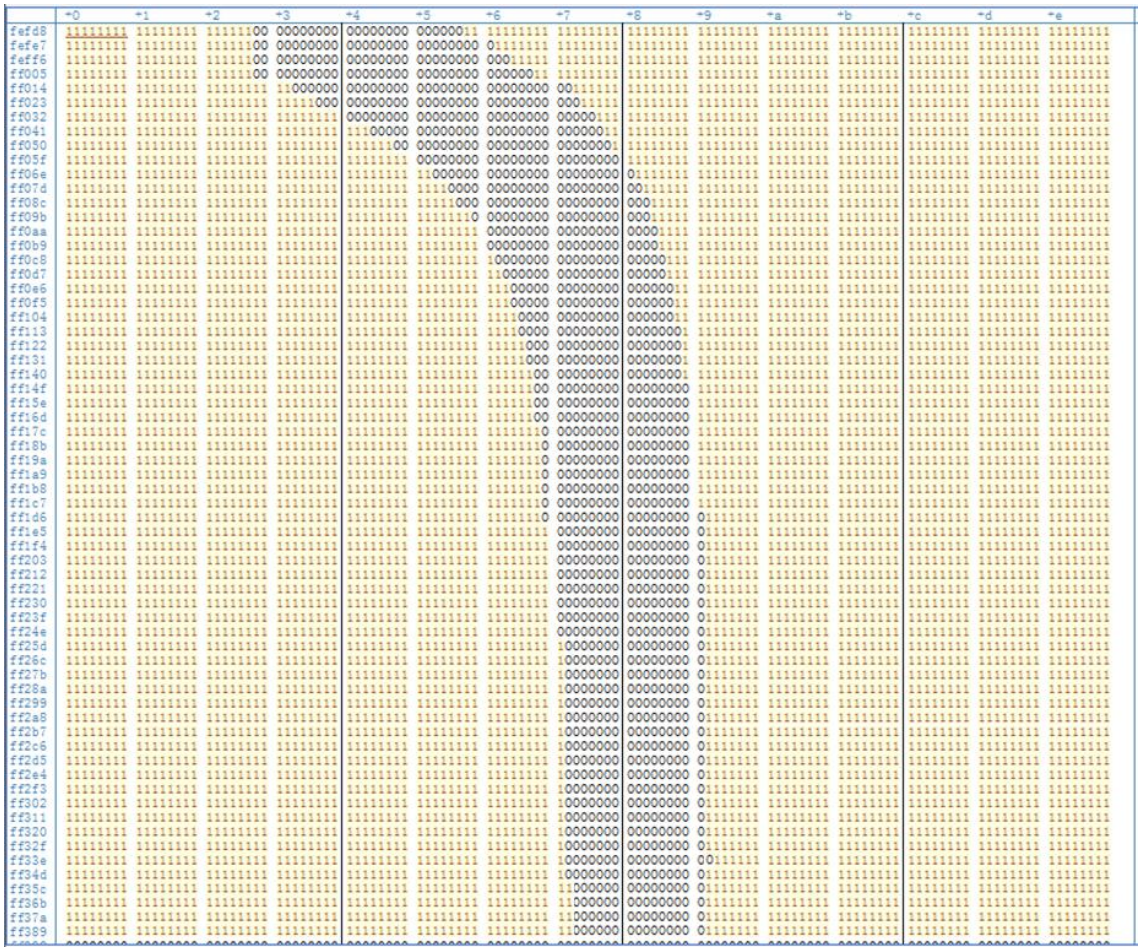

图 4.2　RAM 二值化存储结果

4.3 分辨率测试结果

摄像头配置为 QVGA 格式后的图像分辨率为 320*240。本例程可以实现软件改变采集图像的分辨率。程序"r_camera.h"中定义如下：

```
#define H_RESOLUTION 320    //maximum:320,multiple of 4
#define V_RESOLUTION 240    //maximum:240,multiple of 2
#define H_STEP 1        //for change Horizontal resolution
#define V_STEP 1        //for change Vertical resolution
```

这里用 H_RESOLUTION 设置水平像素点数，V_RESOLUTION 垂直像素点数，即控制窗口大小。参数 H_STEP 和 V_STEP 分别控制水平与垂直分辨率（1 表示读取所有像素点，2 则表示每相邻两个像素点只读取一个，4 则表示每 4 个相邻像素点读取一个）。用来改变图像分辨率。需要注意一点，H_RESOLUTION 与 H_STEP 的商应为整数；V_RESOLUTION 与 V_STEP 的商应为整数。

为了便于理解，这里采用灰度图像进行测试。图 4.3 为 320*240 分辨率的图像采集结果。图 4.4 为 160*120 分辨率的图像采集结果。图 4.5 为 80*60 分辨率的图像采集结果。

图 4.3　分辨率为 320*240 图像采集结果

图 4.4　分辨率为 160*120 图像采集结果

图 4.5　分辨率为 80*60 图像采集结果

4.4 开窗测试结果

摄像头通过配置可以改变图像采集窗口的大小。本例程可以通过程序"r_camera.h"中定义的下面两个参数来改变窗口大小。

```
#define H_RESOLUTION 320    //maximum:320,multiple of 4
#define V_RESOLUTION 240    //maximum:240,multiple of 2
```

这里用 H_RESOLUTION 设置水平像素点数，V_RESOLUTION 垂直像素点数，即控制窗口大小。

为了便于理解，这里同样采用灰度图像进行测试。分辨率为 320*240 不变。图 4.6 为窗口大小 300*200 的图像采集结果。图 4.7 为窗口大小 200*100 的图像采集结果。图 4.8 为 100*50 的图像采集结果。

图 4.6　窗口大小为 300*200 图像采集结果

图 4.7　窗口大小为 200*100 图像采集结果

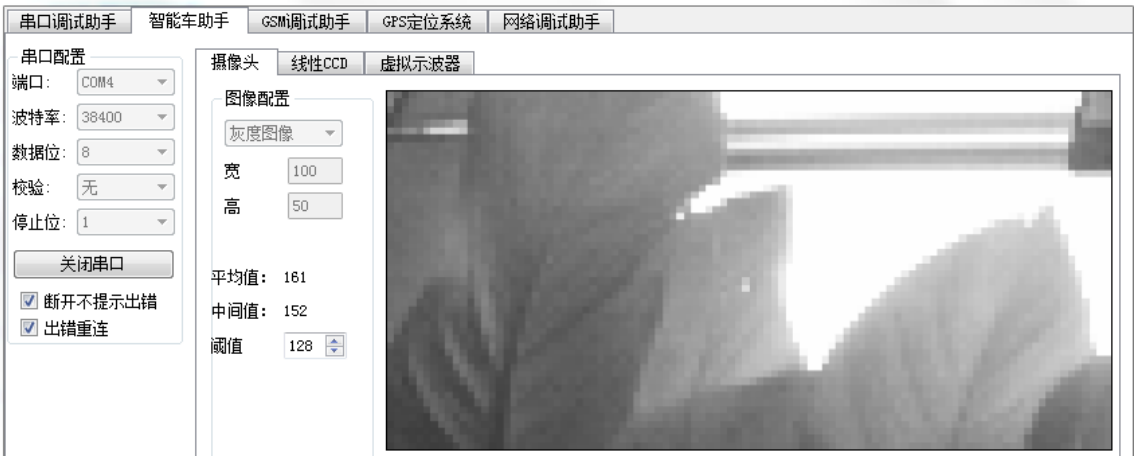

图 4.8　窗口大小为 100*50 图像采集结果

4.4 滤波测试结果

可以在 main 函数中调用 R_Picture_Grab_Smoothing()函数实现滤波功能。

如下面文件"r_camera.h"中定义，使用 80*60 的窗口来测试滤波函数的功能：

```
#define H_RESOLUTION 80    //maximum:320,multiple of 4
#define V_RESOLUTION 60     //maximum:240,multiple of 2
#define H_STEP 1       //for change Horizontal resolution
#define V_STEP 1       //for change Vertical resolution
#define THRESHOLD 0x70
#define ARRAY_SIZE 3000

#define DISPLAY                 //using windows software to display pictures grabbed by camera
```

如图 4.9 滤波测试实物图，在图中加入一些黑点，与十字线。

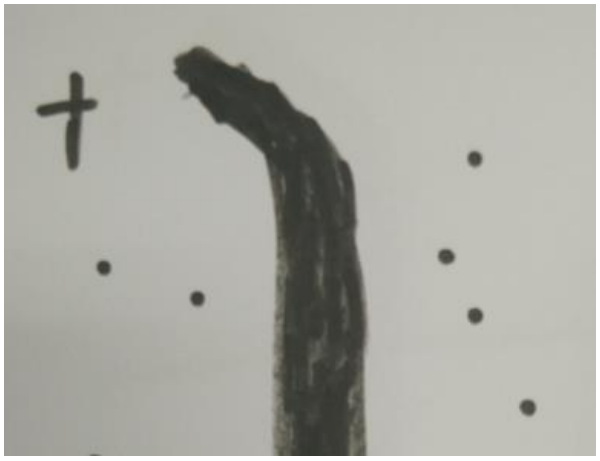

图 4.9　滤波测试实物图

未使用滤波函数时二值化后的测试结果如图 4.10。

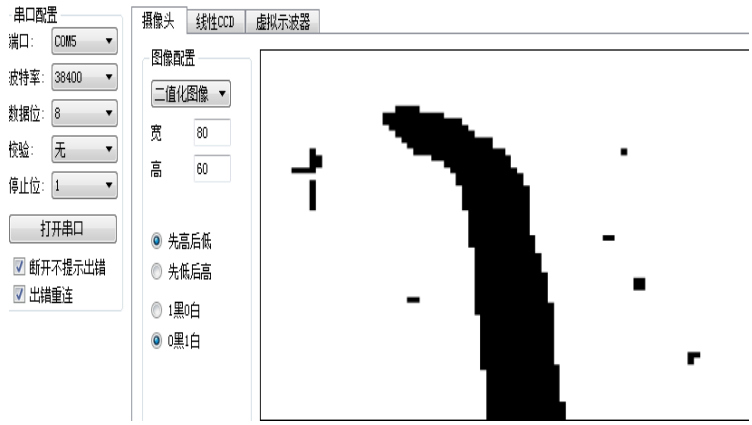

图 4.10　未滤波测试结果

使用滤波函数时二值化后的测试结果如图 4.11。

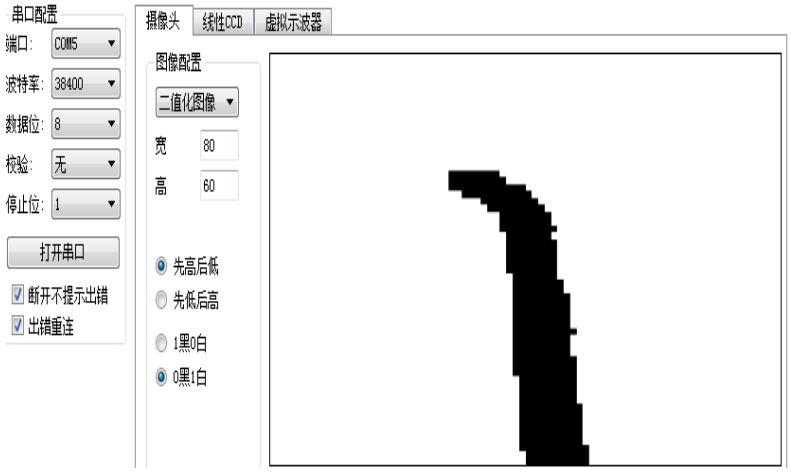

图 4.11　滤波后测试结果

5. 参考文献

用户手册：r01uh0146ej0300_rl78g13.pdf
RL78/G13 开发套件快速入门教程：快速入门教材.pdf

结束语

本例程意在于用 RL78/G13 芯片来实现 CMOS 摄像头图像采集处理。用户可以作为参考进行拓展应用。