图像识别模组用户手册

凌　阳
大学计划

Technology for easy living

凌阳科技股份有限公司

SUNPLUS TECHNOGY CO., LTD.

目录

图像识别模组使用手册

1 前言

图像识别技术目前在人们的生活中，应用越来越普遍，如我们较熟悉的数码相机、摄像头、具有摄像功能的手机、以及我们看见的一些智能玩具上都具有图像识别的功能。在此为大家介绍的此款图像识别模组主要是应用于交互式智能玩具、图像处理产品、教学中。此模组接口简单，应用方便，容易与 MCU 结合使用，若与凌阳公司的 SPCE061A 结合，使其声貌并具，更能体现产品的特效功能。

2 功能简介

该模组可以实现如下功能:

- 识别颜色、形状
- 识别位置

3 模组结构

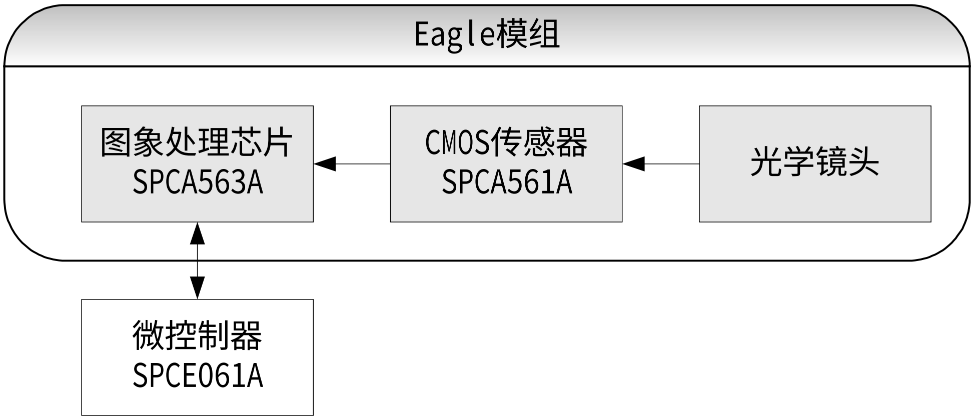


图3.1 图像识别模组结构框图

图像识别模组由光学镜头、CMOS 传感器（SPCA561A）、图像处理芯片（SPCA563A）组成。

SPCA563A 分析和处理 SPCA561A 传过来的图像信号，并得出相应的信息（颜色、形状等等）。此模组接口简单，应用方便，容易与 MCU 结合使用，我们下面将要介绍的 Demo 就是其与 SPCE061A 结合的例子。

图像识别模组和外界的接口共有 6 针，依次接在 SPCA563A 的：VCC、SCK、SD、RDY、图像识别 3_RESET 和 GND。

4 SPCA563A 的硬件介绍

图像识别模组采用的图像识别控制器是凌阳科技的 SPCA563A。此芯片主要应用于具有图像识别处理功能的交互式智能玩具中，芯片内置 AE/AWB 功能，能够处理把来自于 CMOS 传感器的数据处理成 CIF/QVGA 格式。其内部主要嵌入了图像捕获单元、特征识别单元、unSP 内核的 16 位 CPU 单元、ROM 单元等，使其具有颜色识别、形状识别等功能。用户能够使用这些辨识结果去控制一些交互式人机接口。

4.1 特点介绍

4.1.1 硬件概览：

1、 具有 8 位的 CMOS 图像传感器接口， 30fps 的 QQVGA（160×120）、QVGA（320×240）的图像；
2、 具有 AE/AWB 的测量方式的窗口和控制参数的功能；
3、 具有在黑暗处补偿的功能；
4、 具有图像插补的功能；
5、 具有色彩修正的功能；
6、 具有图像（灰度）校正，亮度校正，非线性校正的功能；
7、 可进行 7 种颜色的分辨；
8、 具有消噪功能；
9、 具有同时分割获得 7 种目标物体的尺寸、位置并把结果存贮在对应的 RAM 地址中的功能；
10、 具有获得目标物体的形状和中心位置的功能；
11、 具有主从通讯的串形通讯方式的接口；
12、 具有外接 ROM、RAM 的接口；
13、 工作电压为 3.3V；
14、 具有 100—Pin 的 QFP 封装。

4.1.2 嵌入的 ROM 代码：

1、 AE/AWB 控制代码；
2、 形状识别：目前可以识别圆形、三角形、正方形、长方形、五角形；
3、 串行通讯子程序

4.2 体系结构

下图虚框内就是 SPCA563A 内部结构框图，其中包括 16 位的 unSP 内核 CPU、CDSP、特征寄存器单元、传感器接口、PLL、串行数据接口等单元。芯片内部具有 2K Word 的 RAM、16K Word 的 ROM。

对 CMOS Sensor 采集到的图像信息具有分解的功能，并把需要的内容存贮到对应的地址中，外部的 Host CPU 可以通过读取图像特征寄存器中内容来获得数据；SPCA563A 可以通过专门的 USB 模组，直接与 PC 进行数据通讯；因为 SPCA563A 内部结构中具有 GLOBAL 的接口，使其扩展外部的 ROM、RAM 非常的容易。

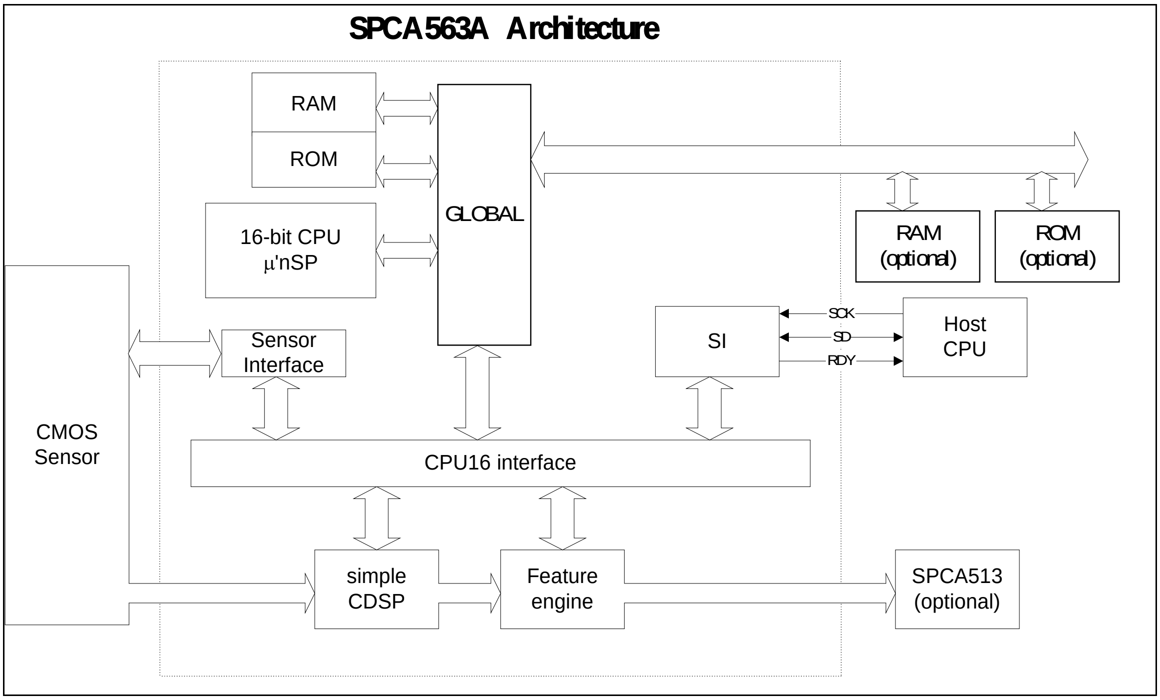


图4.1 SPCA563A 体系结构图

4.3 16 位 unSP 内核的 CPU

SPCA563A 内嵌的代码与其它单元在互通信息的时候，读写的时序如下图所示。

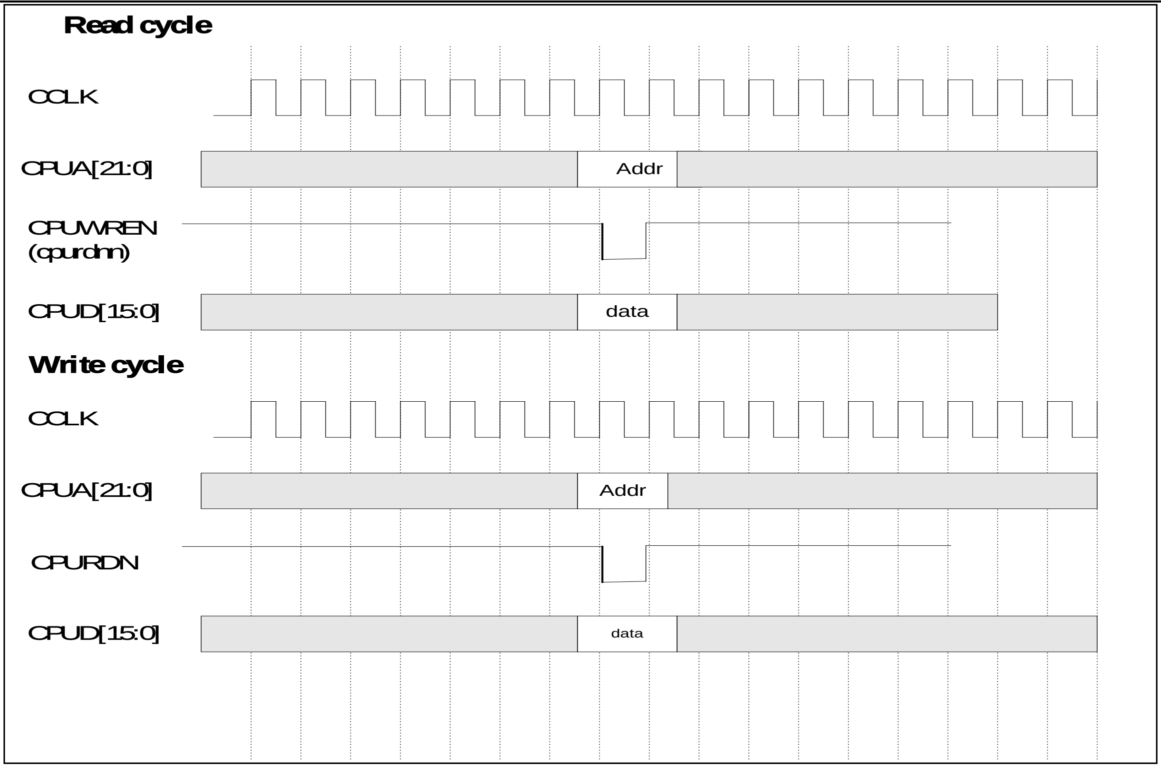


图4.2 读、写时序图

SPCA563A 的存储分布如下图所示。

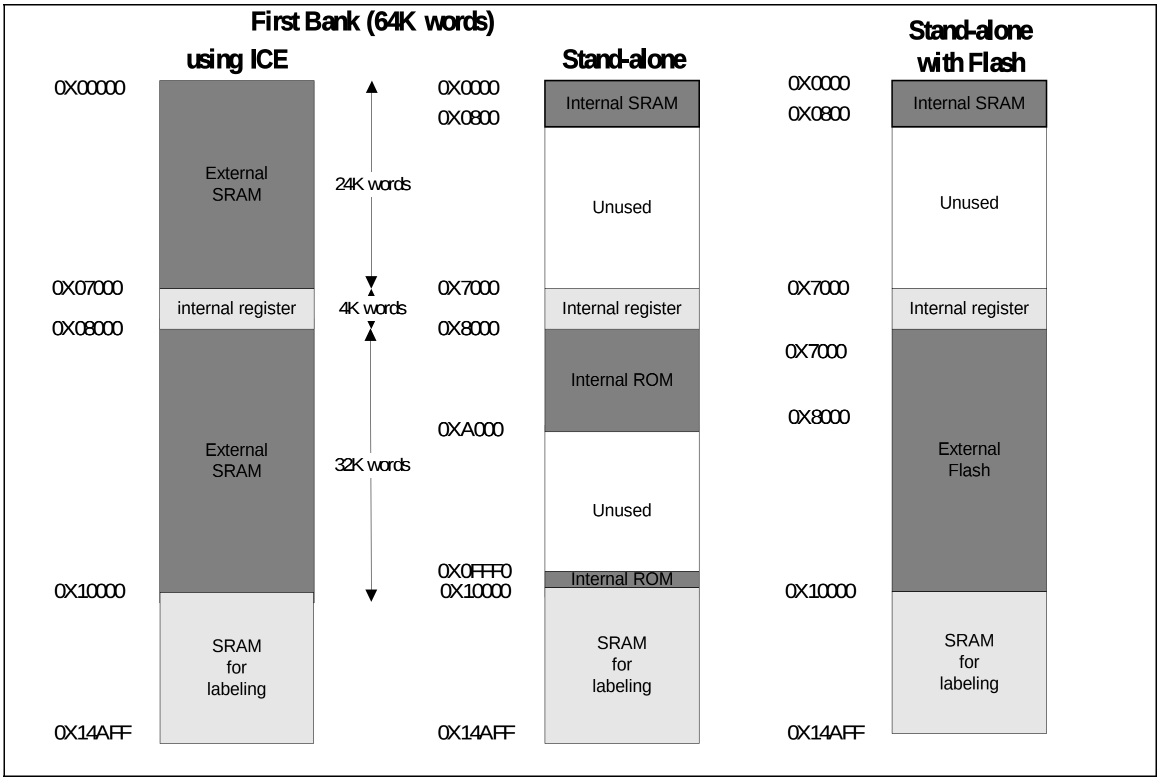


图4.3 存储分布图

4.4 串行接口

下面是 master CPU 与 SPCA563A（slave）串行通讯的时序图和流程图。

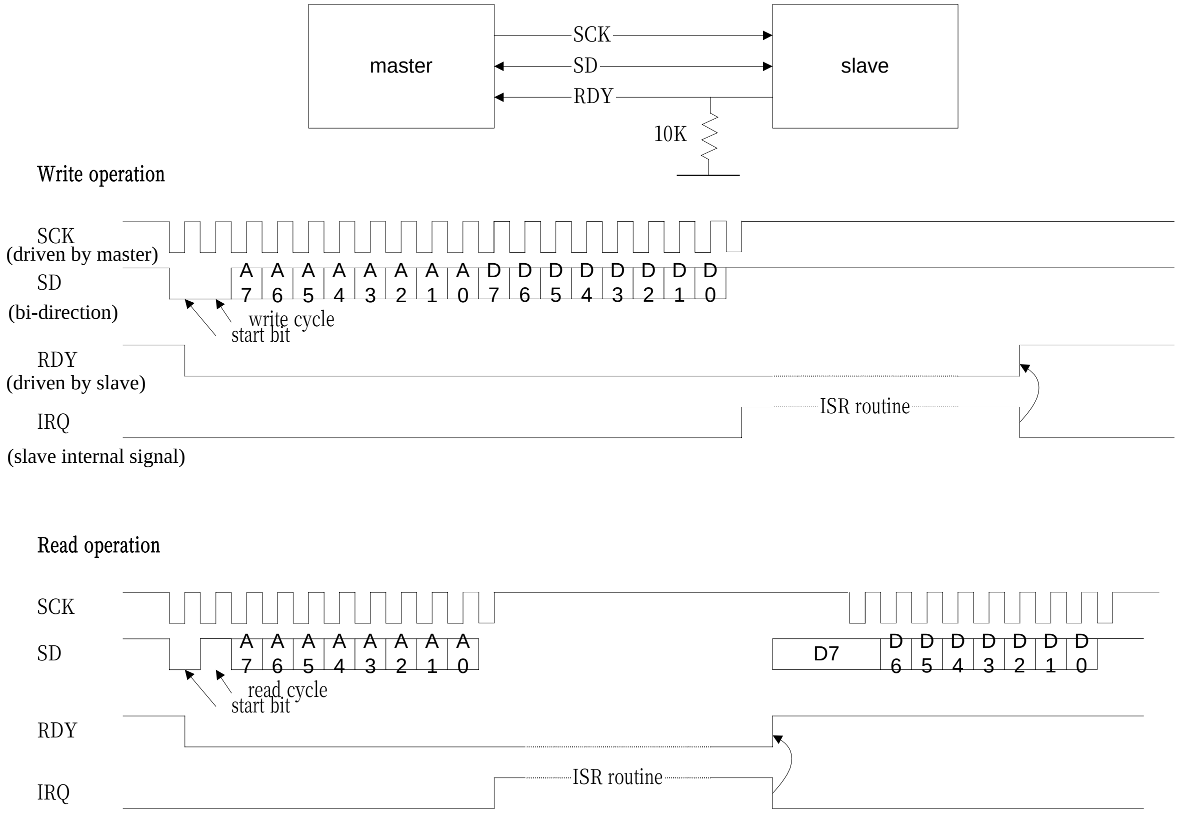


图4.4 主从机之间串行通讯时序图

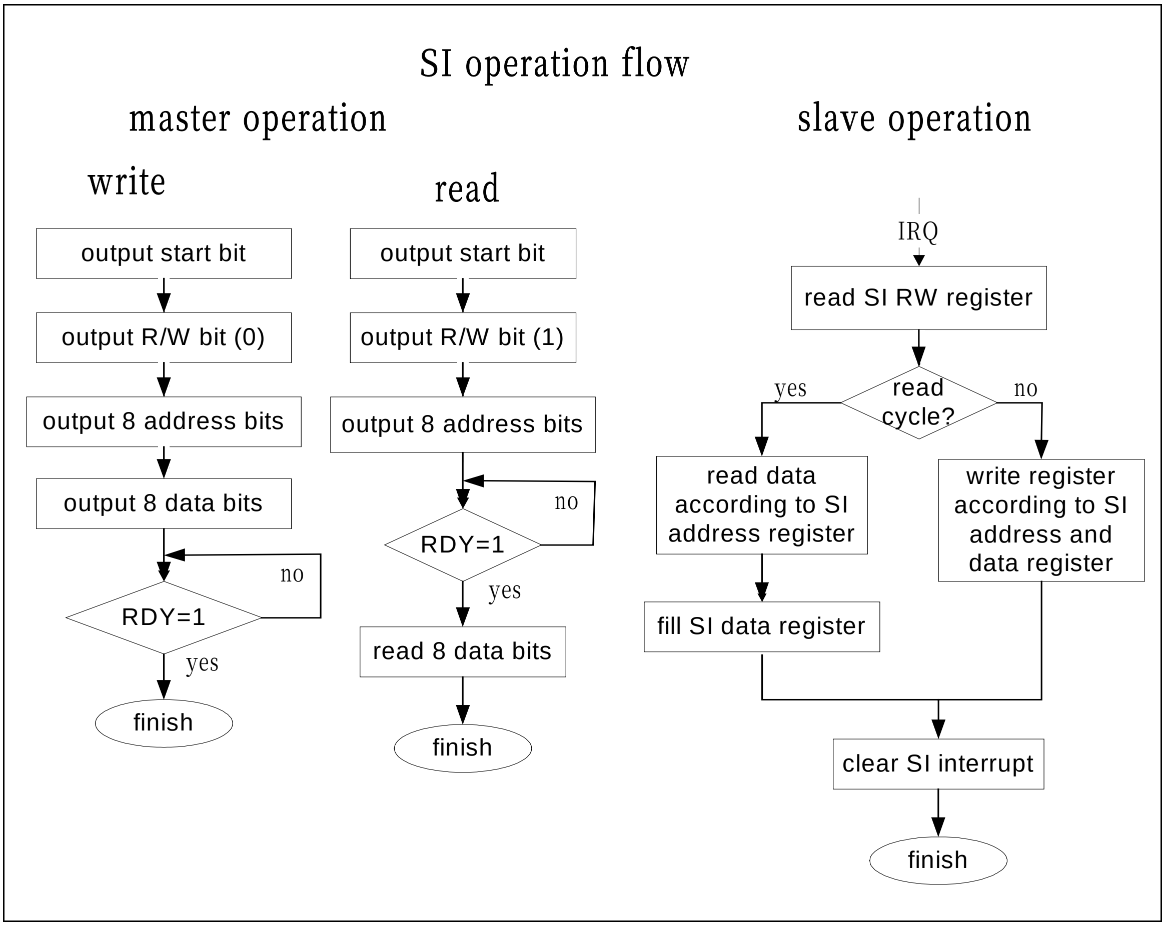


图4.5 串行通讯流程图

4.5 CDSP 结构

CDSP（color DSP）单元可以完成 AE（Auto Exposure）控制、AWB（Auto white Balancing）控制、黑暗处补偿、图像插补、色彩修正、图像（灰度）校正， 亮度校正、非线性校正、识别分离 7 种颜色的功能。

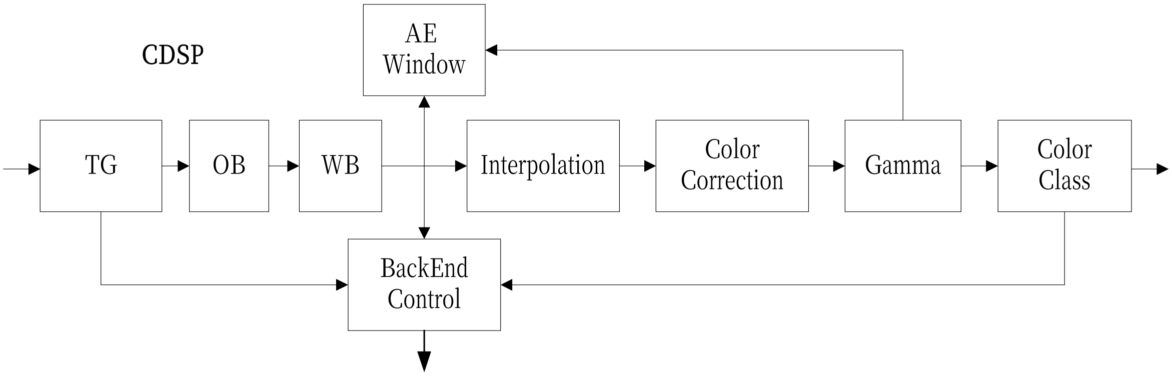


图4.6 CDSP 结构

4.6 特征处理单元

特征处理单元具有两个重要的功能：目标物体的分类、图像分解功能。图像分解的结果将被存放到特征寄存器中，外部 Host CPU 可以通过读的方式获得特征寄存器中的内容；目标分类的数据存放在图形存储寄存器中，SPCA563A 内部的 CPU 可以直接从这里获得目标物体的分类信息和亮度信息。

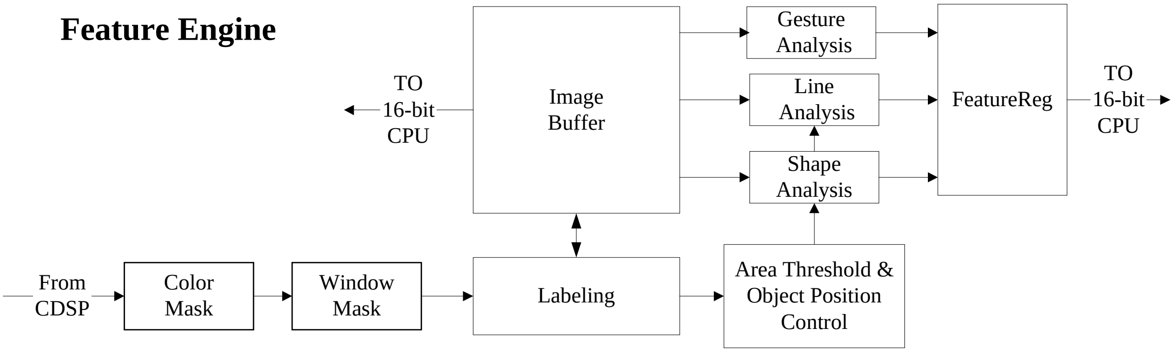


图4.7 特征处理单元的结构

在这个单元中的 Color Mask 可以完成要分解的 7 种颜色的预设置，寄存器地址是 0X7405。

其中的 Labeling 可以选择是分解 7 种物体还是 21 种物体，其寄存器单元是 0X7404。在这里主要可以获得颜色、尺寸、空间位置；如果选择分解 7 种物体那么还可以获得形状等其它信息。

4.7 SPCA563A 管脚列表

表 4.1 是 SPCA563A 所有管脚的列表，表 4.2~4.7 是把这些管脚按功能划分出来的列表。

表4.1 SPCA563A 管脚表

Pin#	Name	Pin#	Name	Pin#	Name	Pin#	Name
1	xvdd	35	ma14	60	ramcs_n	85	ovss
2	xtain	36	ma15	61	romcs_n	86	extvd
3	xtalout	37	ma16	62	cpu16irst	87	exthd
4	xvss	38	gpio6	63	cpu16sck	88	extck1x
5	s513rgb7	26	ma7	51	md10	76	rgbin0
6	s513ck	27	ma8	52	md11	77	rgbin1
7	s513hd	28	ma9	53	md12	78	rgbin2
8	s513vd	29	ma10	54	md13	79	rgbin3
9	gpio0	30	ma11	55	md14	80	rgbin4
10	gpio1	31	ma12	56	md15	81	rgbin5

11	gpio2	32	ma13	57	ovss	82	rgbin6
12	dvss	33	dvss	58	ovdd	83	rgbin7
13	dvdd	34	dvdd	59	ramromoe_n	84	ovdd
14	sck	39	md0	64	Cpu16adpad	89	extck2x
15	sd	40	md1	65	Cpu16wr_n	90	i2cscl
16	rdy	41	md2	66	cpu16rd_n	91	i2csck
17	ovdd	42	ovss	67	dvdd	92	dvdd
18	ovss	43	ovdd	68	dvss	93	dvss
19	ma0	44	md3	69	cpu16irq_n	94	s513rgb0
20	ma1	45	md4	70	cpu16fiq_n	95	s513rgb1
21	ma2	46	md5	71	cpu16clk	96	s513rgb2
22	ma3	47	md6	72	gpio3	97	s513rgb3
23	ma4	48	md7	73	gpio4	98	s513rgb4
24	ma5	49	md8	74	gpio5	99	s513rgb5
25	ma6	50	md9	75	prstnn	100	s513rgb6

表4.2 SRAM/Flash 接口

Name	Pin #	Type	Definition
ma[16:0]	19-32, 35-37	In/Out	SRAM/Flash Address.
md[15:0]	39-41, 44-56	In/Out	SRAM/Flash Data.
ramromoe_n	59	Out	SRAM/Flash Data Output Enable. Low active.
ramcs_n	60	Out	SRAM Chip Select Enable. Low active.
romcs_n	61	Out	Flash Chip Select Enable. Low active.
cpu16wr_n	65	In/Out	SRAM/Flash Data Write Enable. Low active.

表4.3 串行信号接口

Name	Pin #	Type	Definition
sck	14	In	Serial Interface Clock.
sd	15	In/Out	Serial Interface Data.
rdy	16	Out	Serial Interface Ready.

表4.4 16 位 unSP CPU 接口

Name	Pin #	Type	Definition
cpu16irst	62	In	ICE Reset.
cpu16sck	63	In	ICE Clock.
cpu16adpad	64	In/Out	ICE ADPAD.

cpu16wr_n	65	In/Out	16-bit CPU Write Enable. Low Active.
cpu16rd_n	66	In/Out	16-bit CPU Read Enable. Low Active.
cpu16irq_n	69	Out	16-bit CPU IRQ Enable. Low Active.
cpu16fiq_n	70	Out	16-bit CPU FIQ Enable. Low Active.
cpu16clk	71	Out	External 16-bit CPU clock

表4.5 CMOS 传感器接口

Name	Pin #	Type	Definition
rgbin[0:7]	76-83	In	RGB Raw Data.
extvd	86	In	External Vertical Sync.
exthd	87	In	External Horizontal Sync.
i2cscl	90	Out	Synchronous Serial Clock.
I2csda	91	In/Out	Synchronous Serial Data.
Extck1x	88	In	External Clock 1X
Extck2x	89	Out	External Clock 2X

表4.6 MISC 接口

Name	Pin #	Type	Definition
Gpio[0:6]	9-11,72-74,38	In/Out	General Purpose I/O.
prstnn	75	In	Power On Reset.
xvss	4	G	Ground Pin. For crystal pad & phase lock loop.
xtalin	2	In	Crystal In.
Xtalout	3	Out	Crystal Out.
xvdd	1	P	Power Pin. For crystal pad & phase lock loop.
ovdd	17,43,58,84	P	Power Pin. For I/O pads.
ovss	18,42,57,85	G	Ground Pin. For I/O pads.
dvdd	13,34,67,92	P	Power Pin. For core circuit.
dvss	12,33,68,93	G	Ground Pin. For core circuit.

表4.7 SPCA513 模拟信号接口

Name	Pin #	Type	Definition
s513rgb[0:7] (iotrap[0:7])	94-100, 5	In/Out	S513 RGB Raw Data. (I/O Trap Pin) Iotrap[0] : Reserved Iotrap[1] : Cache SRAM internal or external 0: Internal 1: External Iotrap[3:2]: Program location 2‘b00: Internal ROM

			2‘b01: External Flash 2‘b1x: External SRAM Iotrap[5:4]: PLL mode 2‘b00: 0: Disable 2‘b01: Reserved 2‘b10: Reserved 2;b11: The divisor is 1 and the multiplier is 8 (5 MHz) Iotrap[7:6]: Sensor mode 2‘b00: SPCA561A & PB111 else: Reserved
s513vd	8	Out	S513 External Vertical Sync.
s513hd	7	Out	S513 External Horizontal Sync.
s513clk	6	Out	S513 External Clock 1X

5 SPCA563A 的寄存器介绍

SPCA563A 内部寄存器分为中心控制寄存器、特征状态寄存器和识别输出寄存器 3 种。下面的小节是对这 3 种寄存器的分别介绍。

5.1 中心控制寄存器

中心控制寄存器的地址高字节是 0x70。具体内容见下表

表5.1 中心控制寄存器说明

Address 0x7000+	Bit	Att	Name	Description	Default value
0x80	7:0	r	RevisionID	Revision ID	8‘h0
0x90	7:0	r/w	GPIOOE	GPIO output 0: disable 1: enable	8‘hff
0x91	7:0	r/w	GPIOO	GPIO output data	8‘h0
0x92	7:0	r	GPIOI	GPIO input data	8‘h0
0xa0	0	r	Iotrap[0]	Reserved	1‘b0
	1	r	Iotrap[1]	Cache SRAM internal or external 0: internal 1: external	1‘b0
	3:2	r	Iotrap[3:2]	Program location 2‘b00: internal ROM 2‘b01: external Flash 2‘b1x: external SRAM	2‘b0

	5:4	r	Iotrap[5:4]	PLL mode, it defines internal PLL divisor and the multiplier. 2‘b00: disable, bypass PLL. 2‘b01: Reserved 2‘b10: Reserved 2‘b11: The divisor is 1 and the multiplier is 8 .(5 MHz)	2‘b11
	7:6	r	Iotrap[7:6]	Sensor mode 2‘b00: SPCA561A else: reserved	2‘b0
0xe0	0	r/w	FuncSel0	Function Select 0 : Color combination pattern analysis 0 : disable 1 : enable	1‘b0
	3	r/w	FuncSel3	Function Select 3 : Shape analysis 0 : disable 1 : enable	1‘b0
	6	r/w	FuncSel6	Reserved	1‘b0
	7	r/w	FuncSel7	Reserved	1’b0
0xe1	2:0	r/w	AErate	AE adjust frame rate	3‘b11
	3	r/w	LightSel	Light Select : 0: 60Hz 1: 50 Hz	1‘b0
	4	r/w	AEwinsize	AE window size 0: small 1: large	1’b0
	5	r/w	MWstatus	Movable window status 0: disable 1: enable	1‘b1
	6	r/w	AEreset	AE / Movable window reset 0: no reset 1: reset	1‘b0
	7	r/w	AEstatus	AE status 0: disable 1: enable	1‘b1
0xe2	2:0	r/w	AWBrate	AWB adjust frame rate	3‘b11
	5:3	r	AEFrmRateS	AE Frame Rate Status 3'h0: > 20 fps, 3'h1: 17.5 fps, 3'h2: 15 fps, 3'h3: 12.5 fps, 3'h4: 10 fps 3'h5: 7.5 fps 3'h6: 5 fps 3'h7: 2.5 fps	
	6	r/w	AWBreset	AWB reset 0: no reset 1: reset	1‘b0
	7	r/w	AWBstatus	AWB status 0: disable 1: enable	1‘b1
0xe3	7:0	r	AE_Stdnu	AE target number : offset to AE	8‘h0

			mi	(1‘s complement : from -128 to 127)	
0xe5	7:0	r/w	BufData0	Data0 for sensor buffer (high byte)	
0xe6	7:0	r/w	BufData1	Data1 for sensor buffer (low byte)	
0xe7	5:0	r/w	BufAddr	Buffer address	6‘b0
	6	r/w	BufAddrrw	0:read 1: write	1‘b0
	7	r/w	Addrrw_en	0:idle 1: enable. It will auto clear after the r/w action finished	1‘b0
0xe8	0	r/w	BwFuncSel 0	Function Select 6 : Motion tracking 0 : disable 1 : enable	1’b0
	1	r/w	BwFuncSel 1	Adjustment (Sensor position calibration) 0 : disable 1 : enable	1’b0
	2		BwFuncSel 2	Reserved for CPU r/w	1’b0
	3		BwFuncSel 3	Reserved for CPU r/w	1’b0
0xe9~ 0xee	7:0	r/w		Reserved	-

5.2 特征状态寄存器

特征状态寄存器的地址高字节是 0x74。具体内容见下表

表5.2 特征状态寄存器说明

Address 0x7400+	Bit	Att	Name	Description	Default value
0x00	7:0	r/w	ImgWidth	Image width	8‘ha0
0x01	7:0	r/w	ImgHeight	Image height	8‘h78
0x04	6:0	r/w		Reserved	7'h7
	7	r/w	Obj21	0 : 7 object mode 1 : 21 object mode	1'b0
0x05	0	r/w	BlueMask	Blue select, 0: masked 1: unmasked	1‘b1
	1	r/w	GreenMask	Green select 0: masked 1: unmasked	1‘b1
	2	r/w	CyanMask	Cyan select 0: masked 1: unmasked	1‘b1
	3	r/w	RedMask	Red select 0: masked 1: unmasked	1‘b1
	4	r/w	MagentaMask	Magenta select 0: masked 1: unmasked	1‘b1
	5	r/w	SkinMask	Skin select 0: masked 1: unmasked	1‘b1
	6	r/w	YellowMask	Yellow selec 0: masked 1: unmasked	1‘b1

	7			Reserved	1‘b1
0x08	1:0	r/w	S513Sel	S513 mode select: 2‘b00 : raw data 2’b01 : reserved 2‘b10 : reserved 2’b11 : class data	2‘b0
	2	r/w	S513PBSel	0: Data transferred to PC using QVGA format 1: Data transferred to PC using QQVGA format	1‘b0
	7:3	r/w		Reserved	5‘b0
0xoe	7:0	r/w		Reserved	
0x0f	7:0	r/w		Reserved	
0x10	4:0	r	ObjNum	Object number	-
	5	r	ObjNumFull	Object number Full(>7) 0 : Normal 1 : Abnormal, At this case, you must set area threshold registers (Reg. 0x7402-0x7403) or enable adaptive area threshold function (Reg. 0x7404)	-
	6	r	FeatureStatus0	Feature Status 0 0: Normal, all feature register is correct. 1: Abnormal, feature registers may be not correct. At this case, input frame rate must be slow down or increase cpu16 clock (Reg. 0x7081).	-
	7	r	FeatureStatus1	Feature Status 1 0: Normal, all feature register is correct. 1: Abnormal, feature registers may be not correct. At this case, you must mask some input color (Reg. 0x7405).	-
0x11	5:0	r/w	Tarobj	Target object number, Hardware will automatically regulate area threshold if Reg. 0x7404 bit 0 is one.	6‘h7
0x1e	0	r/w	CCSelectb2	Pattern No.b2 (G in R) 0: masked 1: unmasked	1'b1
	1	r/w	CCSelectb3	Pattern No.b3 (B in R) 0: masked 1: unmasked	1'b1
	2	r/w	CCSelectb4	Pattern No.b4 (Y in R) 0: masked 1: unmasked	1'b1
	3	r/w	CCSelectb5	Pattern No.b5 (R in G) 0: masked 1: unmasked	1'b1
	4	r/w	CCSelectb6	Pattern No.b6 (R in Y) 0: masked 1: unmasked	1'b1
	5	r/w	CCSelectb7	Pattern No.b7 (G in Y) 0: masked 1: unmasked	1'b1
	6	r/w	CCSelectb8	Pattern No.b8 (B in G) 0: masked 1: unmasked	1'b1

	7	r/w	CCSelectb9	Pattern No.b9 (Y in B) 0: masked 1: unmasked	1'b1
0x1f	0	r/w	CCSelectba	Pattern No.ba (Y in G) 0: masked 1: unmasked	1'b1
	1	r/w	CCSelectbb	Pattern No.bb (B in Y) 0: masked 1: unmasked	1'b0
	2	r/w	CCSelectbc	Pattern No.bc (G in B) 0: masked 1: unmasked	1'b0
	3	r/w	CCSelectbd	Pattern No.bd (R in B) 0: masked 1: unmasked	1'b0
	7:4			Reserved	
0x1a~0x1f	7:0	r/w		Reserved for CPU read/write	-
0x3f-20	255:0	r	Obj1Feature	Mode 1 : the 7-object mode 0x27~ 0x20: Obj1 Feature, please refer to Feature Content table 0x3f ~ 0x28: reserved. Mode 2 : the 21-object mode 0x27~ 0x20: Obj1 Feature, 0x2f ~ 0x28: Obj2 Feature, 0x37~ 0x30: Obj3 Feature, 0x3f ~ 0x38: reserved.	-
0x5f-40	255:0	r	Obj2Feature	Mode 1 : the 7-object mode 0x47~ 0x40: Obj2 Feature, please refer to Feature Content table 0x5f ~ 0x48: reserved. Mode 2 : the 21-object mode 0x47~ 0x40: Obj4 Feature, 0x4f ~ 0x48: Obj5 Feature, 0x57~ 0x50: Obj6 Feature, 0x5f ~ 0x58: reserved.	-
0x7f-60	255:0	r	Obj3Feature	Mode 1 : the 7-object mode 0x67~ 0x60: Obj3 Feature, please refer to Feature Content table 0x7f ~ 0x68: reserved. Mode 2 : the 21-object mode 0x67~ 0x60: Obj7 Feature, 0x6f ~ 0x68: Obj8 Feature, 0x77~ 0x70: Obj9 Feature, 0x7f ~ 0x78: reserved.	-
0x9f-80	255:	r	Obj4Featur	Mode 1 : the 7-object mode	-

	0		e	0x87~ 0x80: Obj4 Feature, please refer to Feature Content table 0x9f ~ 0x88: reserved. Mode 2 : the 21-object mode 0x87~ 0x80: Obj10 Feature, 0x8f ~ 0x88: Obj11 Feature, 0x97~ 0x90: Obj12 Feature, 0x9f ~ 0x98: reserved.	
0xbf-a0	255:0	r	Obj5Feature	Mode 1 : the 7-object mode 0xa7~ 0xa0: Obj5 Feature, please refer to Feature Content table 0xbf ~ 0xa8: reserved. Mode 2 : the 21-object mode 0xa7 ~ 0xa0: Obj13 Feature, 0xaf ~ 0xa8: Obj14 Feature, 0xb7 ~ 0xb0: Obj15 Feature, 0xbf ~ 0xb8: reserved.	-
0xdf-c0	255:0	r	Obj6Feature	Mode 1 : the 7-object mode 0xc7~ 0xd0: Obj6 Feature, please refer to Feature Content table 0xdf ~ 0xc8: reserved. Mode 2 : the 21-object mode 0xc7 ~ 0xc0: Obj16 Feature, 0xcf ~ 0xc8: Obj17 Feature, 0xd7 ~ 0xd0: Obj18 Feature, 0xdf ~ 0xd8: reserved.	-
0xff-e0	255:0	r	Obj7Feature	Mode 1 : the 7-object mode 0xe7~ 0xe0: Obj7 Feature, please refer to Feature Content table 0xff ~ 0xe8: reserved. Mode 2 : the 21-object mode 0xe7~ 0xe0: Obj19 Feature, 0xef ~ 0xe8: Obj20 Feature, 0xf8 ~ 0xf0: Obj21 Feature, 0xff ~ 0xf9: reserved.	-

表5.3 特征状态内容说明

Address Offset	Bit	Att	Name	Description	Default value
0x00	7:0			Reserved	
0x01	2:0	r	ObjColor	Object Color 3’b001: Blue 3’b010: Green 3’b100: Red 3’b111: Yellow else: reserved	-
	3			Reserved	
	6:4	r/w	ObjShape	Object shape 3’b000: no shape 3’b001: triangle 3’b010: circle 3’b011: square 3’b100: rectangle 3’b101: pentagon else: reserved	-
	7			Reserved	
0x02	7:0	r	ObjStaX	Object starting horizontal index	-
0x03	7:0	r	ObjStaY	Object starting vertical index	-
0x04	7:0	r	ObjEndX	Object ending horizontal index	-
0x05	7:0	r	ObjEndY	Object ending vertical index	-
0x07-06	14:0	r	Object size	Object Size	-

5.3 识别输出寄存器

表5.4 识别输出寄存器说明

Address 0x7500+	Bit	Att	Name	Description	Default value
0x00	4	r	FuncStatus 4	Color & shape function 0: off 1: on	1’b0
	7	r/w	FuncStatus 7	0: Function select from 0x70e0 & 0x70e8 1: automatic function select	1’b0
0x01	4:0	r/w	CharNum	Amount of objects	-
0x04	0	r/w	AutoFuncSel	Automatic loop function select: Color combination pattern 0: off 1: on	1'b1
	2	r/w	AutoFuncSel2	Automatic loop function select: Color & Shape 0: off 1: on	1'b1
	7:5	r/w		Reserved	
0x05	7:0	r/w		Reserved	-
0x06	7:0	r/w	Output01	1st object/character output, See the definition of the object/character/number table	-
0x07	7:0	r/w	Output02	2nd object/character output	-
0x08	7:0	r/w	Output03	3rd object/character output	-

0x09	7:0	r/w	Output04	4th object/character output	-
0x0a	7:0	r/w	Output05	5th object/character output	-
0x0b	7:0	r/w	Output06	6th object/character output	-
0x0c	7:0	r/w	Output07	7th object/character output	-
0x0d	7:0	r/w	Output08	8th object / character output	-
0x0e	7:0	r/w	Output09	9th object / character output	-
0x0f	7:0	r/w	Output10	10th object / character output	-

表5.5 颜色、外形输出表示方式

Bit	Att	Name	Description
2:0	r/w	ObjColor	Object Color 3’b001: Blue 3’b010: Green 3’b100: Red 3’b111: Yellow else: reserved
3			Reserved
6:4	r/w	ObjShape	Object shape 3’b000: no shape 3’b001: triangle 3’b010: circle 3’b011: square 3’b100: rectangle 3’b101: pentagon else: reserved
7			Reserved

表5.6 概要

Function Select	0x7502-0x7503	0x7501	0x7506-0x750f
Color & Shape	N/A	Available	Available

6 SPCA563A 的使用方法

6.1 寄存器的读/写

外接的 Host CPU 可以直接读取 SPCA563A 内部寄存器地址 0x7000-0x7500 单元中的内容，在向这些地址中写内容的时候，必须先向高地址（0x70～0x75）中写入“ 0x0F ”，随后向低地址写入一个字节的内容就可以了。

6.2 识别物体的颜色和形状

6.2.1 识别颜色、外型模式设置

识别输出寄存器中的 0x7500 单元选择为 00（采用手动设置），具体设置模式选择内容

在中心控制寄存器的 0x70e0、0x70e8 中选择，内容见下表。

表6.1 识别颜色、外型模式设置表

模式	0x70e0 寄存器地址	0x70e8 寄存器地址
识别颜色、外型 Look Around	0x08	0x00

6.2.2 分配颜色

向 0x7405 单元写需要被分解的内容，之后再读这个单元，看写入的数值和读出的内容是否一致，一致的时候表示成功进行了颜色的分配。

6.2.3 读取识别出来的颜色和形状

读取寄存器 0x7421、0x7422、0x7423、0x7424、0x7425、0x7426、0x7427 可以得到识别的结果。取得的数据表示的意思如下表所示。

表6.2

Address offset	Bit	Att	Name	Description	Default value
0x01	2:0	r	ObjColor	Object Color 3'b001: Blue 3'b010: Green 3'b100: Red 3'b110: Yellow else: reserved	-
	3			Reserved	
	6:4	r/w	ObjShape	Object shape 3'b000: no shape 3'b001: triangle 3'b010: circle 3'b011: square 3'b100: rectangle 3'b101: pentagon else: reserved	-
	7			Reserved	
0x02	7:0	r	ObjStaX	Object starting horizontal index	-
0x03	7:0	r	ObjStaY	Object starting vertical index	-
0x04	7:0	r	ObjEndX	Object ending horizontal index	-
0x05	7:0	r	ObjEndY	Object ending vertical index	-
0x07-06	14:0	r	Object size	Object Size	-

6.2.4 识别颜色、形状流程图

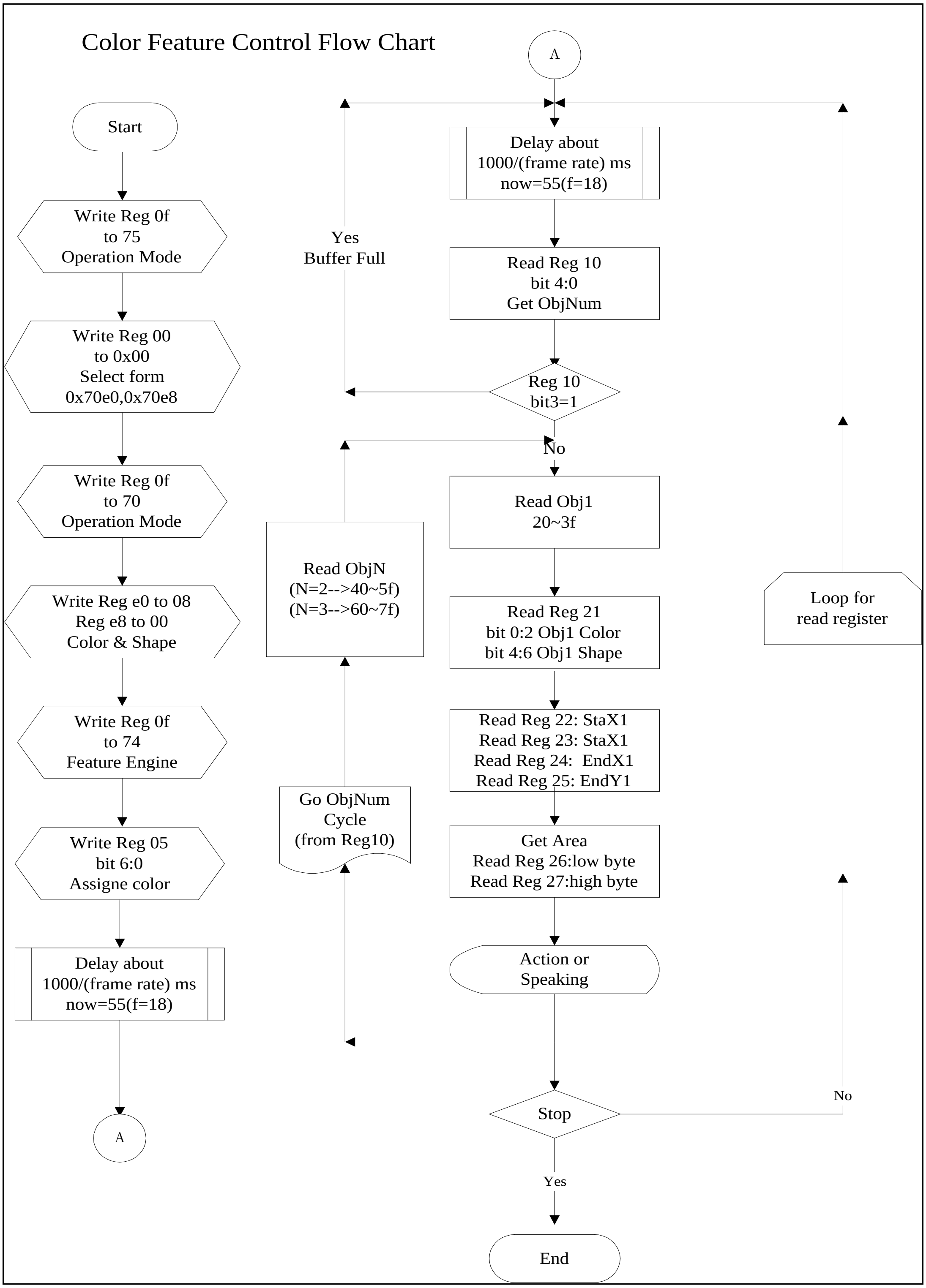

Color Feature Control Flow Chart
Start
Write Reg 0f to 75 Operation Mode
Write Reg 00 to 0x00 Select form 0x70e0,0x70e8
Write Reg 0f to 70 Operation Mode
Write Reg e0 to 08 Reg e8 to 00 Color & Shape
Write Reg 0f to 74 Feature Engine
Write Reg 05 bit 6:0 Assigne color
Delay about 1000/(frame rate) ms now=55(f=18)
A
A
Delay about 1000/(frame rate) ms now=55(f=18)
Read Reg 10 bit 4:0 Get ObjNum
Reg 10 bit3=1
Yes Buffer Full
No
Read Obj1 20~3f
Read Reg 21 bit 0:2 Obj1 Color bit 4:6 Obj1 Shape
Read Reg 22: StaX1
Read Reg 23: StaX1
Read Reg 24: EndX1
Read Reg 25: EndY1
Get Area Read Reg 26:low byte Read Reg 27:high byte
Action or Speaking
Read ObjN (N=2-->40~5f) (N=3-->60~7f)
Go ObjNum Cycle (from Reg10)
Loop for read register
Stop
No
Yes
End

图6.1 识别物体颜色、形状流程图

6.3 常规设置

6.3.1 设置 AE

表6.3

寄存器	Level	Value
0x74e3	15	0x7f
	14	0x70
	13	0x60
	12	0x50
	11	0x40
	10	0x30
	9	0x20
	8	0x10
	7	0x00 (default)
	6	0xf0
	5	0xe0
	4	0xd0
	3	0xc0
	2	0xb0
	1	0xa0
	0	0x90

6.3.2 数据模式设置

表6.4

寄存器	Mode	Value
0x7008	Class Mode	03

注：Class Mode：在与 SPCE061A 通讯中使用这个模式。

7 范例介绍

7.1 概述

在以下的 3 个范例介绍中，只介绍了主要的部分。需要完整的代码请到凌阳大学计划网站的下载区下载。其中 7.2 和 7.3 节介绍的内容请参看图 4.4(串行通讯时序图)、图 4.5（串行通讯流程图)。7.4 节介绍的主函数流程请参考第 6 章第 2 节识别物体颜色和形状的流程图。这 3 个范例的硬件接线方法如下表所示。

表7.1 图像识别模组和 SPCE061A 的连接方法

SPCA563A	61 板
VCC（+5V）	+5V
SCK	IOA8
SD	IOA9
RDY	IOA10
3_RESET	IOA11
GND	GND

图像识别模组的 6pin 接口 J1 或者 J2 标号依次为 VCC/3_RESET/ RDY/ SD/ SCK/ GND

7.2 SPCE061A 读取 SPCA563A 中的内容

```
//===========================================================
// Function Name: F_ReadOper
// Description:         读取数据.
// Input:               None
// Output:              None
// Destroy:
// Used:
// Stacks:
//===========================================================
F_ReadOper:
                r1 = 0x00;
                [R_WaitRDYTime] = r1;
                call  F_Set_SDA_Output;         //  设置 I/O 口
L_ReadStart:
                r1 = [P_IOA_Data];              //  判断  ready = 1?
                r1 &= 0x0400;
                jnz   L_ReadStartRDY;

                r1 = [R_WaitRDYTime];           //  忙，就等待 11ms
```

```
            cmp r1,11;
            jb    L_ReadStart;
            call  F_Reset 图像识别 3Again;          // 11ms 后初始化系统
            retf;

    L_ReadStartRDY:
            r1 = [P_IOA_Buffer];                  // DATA =1
            r1 |= 0x0200;
            [P_IOA_Buffer] = r1;

            r1 = [P_IOA_Buffer];                  // CLK =1
            r1 |= 0x0100;
            [P_IOA_Buffer] = r1;

            r1 = [P_IOA_Buffer];                  // CLK、DATA = 0
            r1 &= 0xfdff;
            [P_IOA_Buffer] = r1;

            r1 = [P_IOA_Buffer];
            r1 &= 0xfeff;
            [P_IOA_Buffer] = r1;

            r1 = [P_IOA_Buffer];                  // CLK = 1
            r1 |= 0x0100;
            [P_IOA_Buffer] = r1;

            r1 = [P_IOA_Buffer];                  // CLK  、DATA = 0
            r1 &= 0xfdff;
            [P_IOA_Buffer] = r1;
```

```
                r1 = [P_IOA_Buffer];                // CLK = 0
                r1 &= 0xfeff;
                [P_IOA_Buffer] = r1;

                r1 = [P_IOA_Buffer];                // DATA = 1
                r1 |= 0x0200;
                [P_IOA_Buffer] = r1;

                r1 = [P_IOA_Buffer];                // CLK = 1
                r1 |= 0x0100;
                [P_IOA_Buffer] = r1;

                r1 = [R_AddrBuffer];                // CLK = 0
                [R_WriteBuffer] = r1;
                call  F_WriteOneByte;
                r1 = [P_IOA_Buffer];
                r1 &= 0xfeff;
                [P_IOA_Buffer] = r1;

                r1 = [P_IOA_Buffer];                // CLK = 1
                r1 |= 0x0100;
                [P_IOA_Buffer] = r1;

                r1 = 0x00;
                [R_WaitRDYTime] = r1;
L_ReadDataNow:
                r1 = [P_IOA_Data];
                r1 &= 0x0400;
                jnz   L_ReadDataNowRDY;
//              jmp  L_ReadDataNow;
```

```
                r1 = [R_WaitRDYTime];
                cmp r1,11;
                jb    L_ReadDataNow;
                call  F_Reset 图像识别 3Again;
                retf;

L_ReadDataNowRDY:
                call  F_Set_SDA_Input;              //  设置 I/O  口
                NOP;
                NOP;
                NOP;
                NOP;
                NOP;
                NOP;
                NOP;
                NOP;
                r1 = [P_IOA_Buffer];
                r1 |= 0x0100;
                [P_IOA_Buffer] = r1;
                call  F_ReadOneByte;                //  读取一个字节

                r1 = [R_ReadBuffer];
                [R_ReadDataBuffer] = r1;

                r1 = [P_IOA_Buffer];
                r1 &= 0xfeff;
                [P_IOA_Buffer] = r1;

                r1 = [P_IOA_Buffer];
```

```
                    r1 |= 0x0200;
                    [P_IOA_Buffer] = r1;

                    r1 = [P_IOA_Buffer];
                    r1 |= 0x0100;
                    [P_IOA_Buffer] = r1;
                    retf;
```

7.3 SPCE061A 向 SPCA563A 中写数据

```
//=====================================================
// Function Name: F_ReadOper
// Description:         写数据.
// Input:              None
// Output:              None
// Destroy:
// Used:
// Stacks:
//=====================================================
F_WriteOper:

                    r1 = 0x00;
                    [R_WaitRDYTime] = r1;

                    call  F_Set_SDA_Output;                 //  设置端口
L_WriteStart:
                    r1 = [P_IOA_Data];                      //  判断  ready = 1?
                    r1 &= 0x0400;
                    jnz   L_WriteStartRDY;
```

```
//                 jmp  L_WriteStart;
                   r1 = [R_WaitRDYTime];
                   cmp r1,11;
                   jb    L_WriteStart;

                   call  F_Reset 图像识别 3Again;
                   retf;

L_WriteStartRDY:
                   r1 = [P_IOA_Buffer];                    // CLK = 1
                   r1 |= 0x0100;
                   [P_IOA_Buffer] = r1;

                   r1 = [P_IOA_Buffer];                    // DATA = 1
                   r1 |= 0x0200;
                   [P_IOA_Buffer] = r1;

                   r1 = [P_IOA_Buffer];                    // CLK=0
                   r1 &= 0xfeff;
                   [P_IOA_Buffer] = r1;

                   r1 = [P_IOA_Buffer];                // DATA =0
                   r1 &= 0xfdff;
                   [P_IOA_Buffer] = r1;


                   r1 = [P_IOA_Buffer];                // CLK =1
                   r1 |= 0x0100;
                   [P_IOA_Buffer] = r1;
```

```
r1 = [P_IOA_Buffer];                  // CLK = 0
r1 &= 0xfdff;
[P_IOA_Buffer] = r1;

r1 = [P_IOA_Buffer];
r1 &= 0xfeff;
[P_IOA_Buffer] = r1;

r1 = [P_IOA_Buffer];                  // CLK = 1
r1 |= 0x0100;
[P_IOA_Buffer] = r1;

r1 = [P_IOA_Buffer];                  // CLK = 0
r1 &= 0xfdff;
[P_IOA_Buffer] = r1;

r1 = [R_AddrBuffer];
[R_WriteBuffer] = r1;

call  F_WriteOneByte;                 //  写地址

r1 = [R_WriteDataBuffer];
[R_WriteBuffer] = r1;
call  F_WriteOneByte;                 //  写数据

r1 = [P_IOA_Buffer];
r1 &= 0xfeff;
[P_IOA_Buffer] = r1;

r1 = [P_IOA_Buffer];
```

```
                    r1 |= 0x0200;
                    [P_IOA_Buffer] = r1;

                    r1 = [P_IOA_Buffer];
                    r1 |= 0x0100;
                    [P_IOA_Buffer] = r1;

                    r1 = 0x00;
                    [R_WaitRDYTime] = r1;
L_WWaitISRInt:
                    r1 = [P_IOA_Data];              // 判断是否写完
                    r1 &= 0x0400;
                    jnz  L_NonWaitRDYAgain;
//                  jmp  L_WWaitISRInt;

                    r1 = [R_WaitRDYTime];
                    cmp r1,11;
                    jb    L_WWaitISRInt;
                    call  F_Reset图像识别3Again;
L_NonWaitRDYAgain:
                    retf;
```

7.4 颜色、形状和位置识别范例

这个范例主要是让图像识别识别物体的颜色、外形和空间位置，并用 SPCE061A 进行语音播放和电机控制。

把模型放在图像识别的前方（约 30cm 处）。按 61 板上的 KEY1 键，小车就会说出对应物体的颜色和形状。按 KEY2 键，手持红色的物体让小车识别，小车识别出来后会跟着这个红色物体移动。注意本范例中图像识别只跟踪红色物体，而且物体移动的时候不要太快。

7.4.1 主程序介绍

```
#include "bsrsd.h"
#include "图像识别.h"
//======================== 定义全程变量 =========================
int gActivated = 0;                    // 该变量用于检测是否有触发命令，当有识别出语
句
                                       // 为触发名称则该位置 1
int VR_PrevResult;
int VR_PrevResult_Color;               // 执行命令后的结果
int VR_PrevResult_Shape;
int VR_PrevResult_CenterX;
int VR_PrevResult_CenterY;
int VR_PrevResult_AreaH;

int VR_OverFlag = 0 ;                       // 标志
int VR_TimeFlag = 0 ;
int VR_TimeDeldy = 0 ;
int Key = 0;
int SleepCount = 0;
int TimeCount = 0;
int   Ret  = 0;

//===================SCAM2000 播放方式 =======================

void PlayRespond(int Result)
{
    BSR_StopRecognizer();
    SACM_A2000_Initial(1);
    SACM_A2000_Volume(3);                           // Set Speech volume
    SACM_A2000_Play(Result, 3, 3);
    while((SACM_A2000_Status()&0x0001) != 0)
    {
        SACM_A2000_ServiceLoop();
        }
        SACM_A2000_Stop();
        BSR_InitRecognizer(BSR_MIC);
        BSR_EnableCPUIndicator();
    }
    //==================== 主程序 ==================================
    int main()
    {
        Initial_IOA();                      // 初试化
        Enable_1KHzAnd4HzInt();             // 设置 1KHZ、4KHZ 中断
        IntialTo 图像识别 3();                  // 初始化 SPCA563

        VR_TimeFlag = 0;                    // 延时 5S
        VR_TimeDeldy = 0;
        while (VR_TimeFlag !=1)
        {
            Delay(5000);
        }

        SetAWBOFF();                            // 关 AWB
        PlayRespond(S_Sir);                     // 回答：“ 先生，在这里 ”

    while(1)
        {
    PlayRespond(S_Yes);                         // SCAM2000 播放回答：是
            VR_Search_Color();                  // 辩识颜色和形状
    }
    }
    //==============================================
    // 函数名称:   VR_Search_Color(void)
```

```
// 功能描述:  获得颜色和形状后采用 SCAM2000 播放回答
// 入口参数:
// 出口参数:
// 破坏寄存器：
//==================================================
void VR_Search_Color(void)
{
	VR_PrevResult_Color = 0;
	VR_PrevResult_Shape = 0;
	VR_OverFlag = 0;
	LookAround_Initial();

	while (VR_OverFlag!=1)
	{
		LookAround();
	}
	switch(VR_PrevResult_Color)
	{

		case C_Red:				// 红色
			PlayRespond(S_Red);
			break;

		case C_Green:				// 绿色
			PlayRespond(S_Green);
			break;

		case C_Blue:				// 蓝色
			PlayRespond(S_Blue);
			break;
```

```
        case C_Yellow:                      // 黄色
            PlayRespond(S_Yellow);
            break;
        default:
            break;
    }

    switch(VR_PrevResult_Shape)
    {

        case C_Triangle:                    // 三角形
            PlayRespond(S_Triangle);
            break;
        case C_Circle:                      // 圆形
            PlayRespond(S_Circle);
            break;
        case C_Square:                      // 正方形
            PlayRespond(S_Square);
            break;
        case C_Rectangle:                   // 长方形
            PlayRespond(S_Rectangle);
            break;
        case C_Pentagon:                    // 五角
            PlayRespond(S_Pentagon);
            break;
        default:
            break;
    }
}
```

7.4.2 Look Around 主函数介绍

```
//===============================================================
//描述:
//              获得物体颜色、形状、中心水平值 X、中心垂直 Y、尺寸远近
//
//日期:
//              2001.08.08          ;编写
//              2003.12.01          ;修改
//===============================================================
.include  hardware.inc;
.include  a2000.inc;
.include  图像识别 3.inc;

.external _VR_OverFlag;                         // 程序结束标志
.external _VR_PrevResult_Color;                 // 存校正后的颜色
.external _VR_PrevResult_Shape;                 // 存校正后的形状
.external _VR_PrevResult_CenterX;               // 存校正后的中心 X
.external _VR_PrevResult_CenterY;               // 存校正后的中心 Y
.external _VR_PrevResult_AreaH;                 // 存校正后的尺寸高位数据
.CODE

//----------------------- Look Around Main Loop --------------------

//=================================================
//  函数名称: LookAround_Initial
//  功能描述: 初始化 LookAround
//  入口参数:
//  出口参数:
//  破坏寄存器：
```

```
//===================================================
.public    _LookAround_Initial;
.public    F_LookAround_Initial;
_LookAround_Initial:  .proc
F_LookAround_Initial:
                    r1 = 0x0000;
                    [R_Flag] = r1;
                    [R_SeekFlag] = r1;
                    [R_Color] = r1;
                    [R_Shape] = r1;
                    [R_PreColor] = r1;
                    [R_PreShape] = r1;
                    [R_PreAreaH] = r1;
                    [R_Temp1] = r1;
                    [R_Temp2] = r1;
                    [R_AreaH] =  r1;
                    [R_CenterX] = r1;
                    [R_CenterY] = r1;
                    [_VR_PrevResult_Color] = r1;
                    r1 = 0x01;                          // 程序入口偏移量
                    [R_Offset] = r1;
                    retf;
                    .endp
//==================================================
// 函数名称: T_JumpTable
// 功能描述: 程序入口
// 入口参数:
// 出口参数:
// 破坏寄存器：
//===================================================
.public    T_JumpTable;
T_JumpTable:
                        .dw  L_LookAroundStoped;                    // 0
                        .dw  L_ResetEngine;                         // 1
                        .dw  L_BeingDelay1stTime;                    // 2
                        .dw  L_BeingDelay2ndTime;                   // 3
                        .dw  L_BeingDelay3rdTime;                   // 4
                        .dw  L_BeingDelay4thTime;                   // 5
                        .dw  L_BeingDelay5thTime;                   // 6
                        .dw  L_BeingDelay6thTime;                   // 7
                        .dw  L_AssigneColorAgain;                   // 8
                        .dw  L_BeingDelay7thTime;                   // 9
                        .dw  L_BeingDelay9thTime;                   // A

L_LookAroundStoped:
                        retf;
//==================================================
//  函数名称: LookAround
//  功能描述: 获得物体颜色、形状、X、Y、Size
//  入口参数:
//  出口参数:
//  破坏寄存器：
//===================================================

.public    _LookAround;
.public    F_LookAround;
_LookAround:        .proc
F_LookAround:
                        r1 = [R_Flag];                              // R_Flag = 0  初始化
                        r1 &= 0x0004;
```

```
                    jnz   L_Restart;

                    r1 = T_JumpTable;
                    r1 += [R_Offset];
                    r1 = [r1];
                    pc = r1;
L_Restart:
                    call  F_LookAround_Initial;

L_ResetEngine:
                    r1 = 0x02;
                    [R_Offset] = r1;
                    r1 = 0x00;
                    [R_DelayTime] = r1;
                    retf;

L_BeingDelay1stTime:
                    call  F_Delay55ms;                    // 延时 55MS
                    r1 = [R_Flag];
                    r1 &= 0x8000;
                    jnz   L_HaveDelay1stTime;             // 判断时间到否
                    retf;

L_HaveDelay1stTime:
L_BeingDelay2ndTime:
L_BeingDelay3rdTime:
L_BeingDelay4thTime:

                    call  F_NormalOperMode;               // 写 0F 到高地址 75
                    call  F_OperMode;                     // 写 00 到低地址 00 ;允许 75E0、
```

```
75E8 使用
                    r1 = 0x06;
                    [R_Offset] = r1;
                    r1 = 0x00;
                    [R_DelayTime] = r1;
                    retf;

L_BeingDelay5thTime:
                    call  F_Delay66ms;                  //  延时 66MS
                    r1 = [R_Flag];
                    r1 &= 0x8000;
                    jnz   L_HaveDelay5thTime;
                    retf;

L_HaveDelay5thTime:
                    call  F_ShapeAnaly;                 //  写 0F 到高地址 70；写 08 到低地址 E0
                    call  F_Clear70E8;                  //  写 0F 到高地址 70；写 00 到低地址 E8；设置模式
L_FeatureEagine:
                    call  F_FeatureEngine;              //  写 0F 到高地址 74，
                    r1 = 0x07;
                    [R_Offset] = r1;
                    r1 = 0x00;
                    [R_DelayTime] = r1;
                    retf;

L_BeingDelay6thTime:
                    call  F_Delay66ms;                  //  延时 66MS
                    r1 = [R_Flag];
                    r1 &= 0x8000;
```

```
                jnz  L_HaveDelay6thTime;
                retf;

L_HaveDelay6thTime:

//---------------------- ResetEngine End -------------------------

L_AssigneColorAgain:
L_AssigneColor:
                call  F_LookAssigneColor;          // 写需要分识别的颜色到 7405 的低地址 05 中
                r1 = 0x09;
                [R_Offset] = r1;
                r1 = 0x00;
                [R_DelayTime] = r1;
                retf;

L_BeingDelay7thTime:
                call  F_Delay66ms;                 // 延时 66MS
                r1 = [R_Flag];
                r1 &= 0x8000;
                jnz  L_HaveDelay7thTime;
                retf;

L_HaveDelay7thTime:
                r1 = 0x0A;
                [R_Offset] = r1;
                r1 = 0x00;
                [R_DelayTime] = r1;
                retf;
```

```
L_BeingDelay9thTime:
                call  F_Delay5ms;                    //  延时 5MS
                r1 = [R_Flag];
                r1 &= 0x8000;
                jnz   L_HaveDelay9thTime;
                retf;

L_HaveDelay9thTime:

                call  F_GetObjNum;                   //  读 7410 单元中的内容
                r1 = [R_SeekFlag];                       //  判断标志位
                r1 &= 0x80;
                jnz   L_LookObjBufferFull1;          //  目标数>8，则转

                r1 = [R_SeekFlag];
                r1 &= 0xfe;
                [R_SeekFlag] = r1;                   //  目标数<8 ，清寄存器和标志
                r1 = 0x00;
                [R_PreAreaH] = r1;
                [R_PreColor] = r1;
                [R_PreShape] = r1;
                jmp       L_LookGetObjData;
L_LookObjBufferFull1:
                call  L_LookObjBufferFull
L_LookGetObjData:
                call  F_LookGetObjData;              //  读取颜色、形状、X、Y、SIZE
的值
                r1 = [R_SeekFlag];
                r1 &= 0x01;                          //  是否获得数值标志
                jz L_LookNonGetObj;
                r1 = [R_PreShape];                  // 是否获得形状
                cmp r1,0x00;
                je    L_LookNonGetObj;              // 相等则转
                r1 = [R_PreColor];
                cmp r1,[R_Temp1];
                jne   L_NotTheSameObj;              // 不相等则转
                r1 = [R_PreShape];
                cmp r1,[R_Temp2];
                je    L_LookNonGetObj;              // 相等则转
    L_NotTheSameObj:
                r1 = [R_PreColor];                  // 获得颜色
                [R_Temp1] = r1;
                [_VR_PrevResult_Color] = r1;

                r1 = [R_PreShape];                  // 获得形状
                [R_Temp2] = r1;
                [_VR_PrevResult_Shape] = r1;

                r1 = [R_CenterX]                    // 获得物体的中心水平值 X
                [_VR_PrevResult_CenterX] = r1

                r1 = [R_CenterY]                    // 获得物体的中心垂直值 Y
                [_VR_PrevResult_CenterY] = r1

                r1 = [R_AreaH]                      // 获得物体尺寸高位数据
                [_VR_PrevResult_AreaH] = r1

                r1 = 0x0001;
                [_VR_OverFlag] = r1;                    // 置结束标志
                retf;
```

```
L_LookObjBufferFull:
L_LookNonGetObj:
                r1 = 0x08;                  // 重新读数据
                [R_Offset] = r1;
                retf;
                .endp

//-------------------- Look Around Main Loop -----------------------

//------------------------ 结束 LookAround.asm ----------------------------
```

7.4.3 主要子函数介绍

```
//=================================================
// 函数名称:   NormalOperMode
// 功能描述:写 00 到 0x7500
// 入口参数:
// 出口参数:
// 破坏寄存器：
//=================================================
.public   _NormalOperMode;
.public   F_NormalOperMode;
_NormalOperMode:    .proc
F_NormalOperMode:
                call  F_HighAddr75;
                r1 = 0x00;
                [R_AddrBuffer] = r1;
                r1 = 0x00;
                [R_WriteDataBuffer] = r1;
                call  F_WriteOper;
```

```
                                retf;
                                        .endp


//==============================================================
//函数：F_OperMode
//描述：写 0F 到地址 0x70,允许使用
//参数：无
//返回：无
//破坏寄存器：
//==============================================================
.public    F_OperMode;
F_OperMode:
                                r1 = 0x0f;
                                [R_AddrBuffer] = r1;
                                r1 = 0x70;
                                [R_WriteDataBuffer] = r1;
                                call  F_WriteOper;
                                retf;
//==============================================================
//函数：F_ShapeAnaly
//语法：汇编调用
//描述：写 08 到地址 0x70e0 中，模式设置
//参数：无
//返回：无
//==============================================================
.public    F_ShapeAnaly;
F_ShapeAnaly:
                                call  F_HighAddr70;
                                r1 = 0xE0;
                                [R_AddrBuffer] = r1;
```

```
                r1 = 0x08;
                [R_WriteDataBuffer] = r1;
                call  F_WriteOper;
                retf;
//=====================================================
//函数：F_Clear70E8
//语法：调用
//描述：写 00 到地址 0x70E8，模式设置
//参数：无
//返回：无
//=====================================================
.public    F_Clear70E8;
F_Clear70E8:
                call  F_HighAddr70;
                r1 = 0xe8;
                [R_AddrBuffer] = r1;
                r1 = 0x00;
                [R_WriteDataBuffer] = r1;
                call  F_WriteOper;
                retf;
/=====================================================
//函数：F_FeatureEngine
//语法：汇编调用
//描述：写 0F 到地址 0x74,允许使用
//参数：无
//返回：无
//=====================================================

.public    F_FeatureEngine;
F_FeatureEngine:
```

```
                r1 = 0x0f;
                [R_AddrBuffer] = r1;
                r1 = 0x74;
                [R_WriteDataBuffer] = r1;
                call  F_WriteOper;
                retf;
//====================================================
//函数：F_LookAssigneColor
//描述：向 0x7405 单元依次写需要分解的颜色
//参数：无
//返回：无
//====================================================

.public    F_LookAssigneColor;
F_LookAssigneColor:
                call  F_HighAddr74;
                r1 = [R_ColorIndex];
                r1 += 1;
                [R_ColorIndex] = r1;

                r1 = [R_ColorIndex];
                cmp r1,0x03;
                jb L_LookGetColor;
                r1 = 0x01;
                [R_ColorIndex] = r1;
L_LookGetColor:
                r1 = T_LookColorTable;
                r1 += [R_ColorIndex];
                r1 = [r1];
                [R_WriteDataBuffer] = r1;
                r1 = 0x05;
                [R_AddrBuffer] = r1;
                call  F_WriteOper;
                retf;
T_LookColorTable:
                .dw  0x48;
                .dw  0x03;
                .dw  0x48;
                .dw  0x03;
                .dw  0x48;

//==================================================
//函数：F_GetObjNum
//描述：获得物体的数目
//参数：无
//返回：无
//==================================================

.public    F_GetObjNum;
F_GetObjNum:
                call  F_HighAddr74;
                r1 = 0x10;
                [R_AddrBuffer] = r1;
                r1 = 0x00;
                [R_ReadDataBuffer] = r1;
                [R_ObjNum] = r1;

                call  F_ReadOper;            //Get Obj Num

                r1 = [R_ReadDataBuffer];
                r1 &= 0x0F;
                [R_ObjNum] = r1;

                r1 = [R_ObjNum];
                r1 &= 0x08;
                jnz L_Bufferfull;            //Num Buffer full
                r1 = [R_SeekFlag];
                r1 &= 0x7f;
                [R_SeekFlag] = r1;
                retf;
L_Bufferfull:
                r1 = [R_SeekFlag];
                r1 |= 0x80;
                [R_SeekFlag] = r1;
                retf;
//==================================================
//函数：F_LookGetObjData
//语法：调用
//描述：获得颜色、外型、水平方向的数值、
//      垂直方向的数值、空间的远近的参数
//参数：无
//返回：无
//==================================================
.public   F_LookGetObjData;
F_LookGetObjData:
                r1 = [R_ObjNum];
                r1 &= 0x07;
                [R_ObjNum] = r1;
                r1 = [R_ObjNum];
                cmp r1,0x01;
                je    F_LookGetOneObjData;
                cmp r1,0x02;
                je    F_LookGetTwoObjData;
                cmp r1,0x03;
                je    F_LookGetThrObjData;
                cmp r1,0x04;
                je    F_LookGetFourObjData;
                cmp r1,0x05;
                je    F_LookGetFiveObjData;
                cmp r1,0x06;
                je    F_LookGetSixObjData;
                cmp r1,0x07;
                je    F_LookGetSevenObjData;
                retf;
//=======================================================
//函数：F_LookGetOneObjData
//语法：调用
//描述：获得一个目标物体
//参数：无
//返回：无
//=======================================================
F_LookGetOneObjData:
                call  F_LookGetObj1stData;
                retf;
//=======================================================
//函数：F_LookGetTwoObjData
//语法：调用
//描述：获得二个目标物体
//参数：无
//返回：无
```

```
//=====================================================
F_LookGetTwoObjData:
                call  F_LookGetObj1stData;
                call  F_LookGetObj2ndData;
                retf;
//=====================================================
//函数：F_LookGetThrObjData
//语法：调用
//描述：获得三个目标物体
//参数：无
//返回：无
//=====================================================
F_LookGetThrObjData:
                call  F_LookGetObj1stData;
                call  F_LookGetObj2ndData;
                call  F_LookGetObj3rdData;
                retf;
//=====================================================
//函数：F_LookGetFourObjData
//语法：调用
//描述：获得四个目标物体
//参数：无
//返回：无
//=====================================================
F_LookGetFourObjData:
                call  F_LookGetObj1stData;
                call  F_LookGetObj2ndData;
                call  F_LookGetObj3rdData;
                call  F_LookGetObj4thData;
                retf;
```

```
//====================================================
//函数：F_LookGetFiveObjData
//语法：调用
//描述：获得五个目标物体
//参数：无
//返回：无
//====================================================
F_LookGetFiveObjData:
                call  F_LookGetObj1stData;
                call  F_LookGetObj2ndData;
                call  F_LookGetObj3rdData;
                call  F_LookGetObj4thData;
                call  F_LookGetObj5thData;
                retf;
//====================================================
//函数：F_LookGetSixObjData
//语法：调用
//描述：获得六个目标物体
//参数：无
//返回：无
//====================================================
F_LookGetSixObjData:
                call  F_LookGetObj1stData;
                call  F_LookGetObj2ndData;
                call  F_LookGetObj3rdData;
                call  F_LookGetObj4thData;
                call  F_LookGetObj5thData;
                call  F_LookGetObj6thData;
                retf;
//====================================================
//函数：F_LookGetSevenObjData
//语法：调用
//描述：获得目标物体
//参数：无
//返回：无
//=======================================================
F_LookGetSevenObjData:
                    call  F_LookGetObj1stData;
                    call  F_LookGetObj2ndData;
                    call  F_LookGetObj3rdData;
                    call  F_LookGetObj4thData;
                    call  F_LookGetObj5thData;
                    call  F_LookGetObj6thData;
                    call  F_LookGetObj7thData;
                    retf;
//=======================================================
//函数：F_LookGetObj1stData
//语法：调用
//描述：获得第一个目标物体的数据
//参数：无
//返回：无
//=======================================================

.public   F_LookGetObj1stData;
F_LookGetObj1stData:
                    call  F_GetObj1stData;
                    call  F_LookCheckTheObj;
                    retf;
//=======================================================
//函数：F_LookGetObj2ndData
```

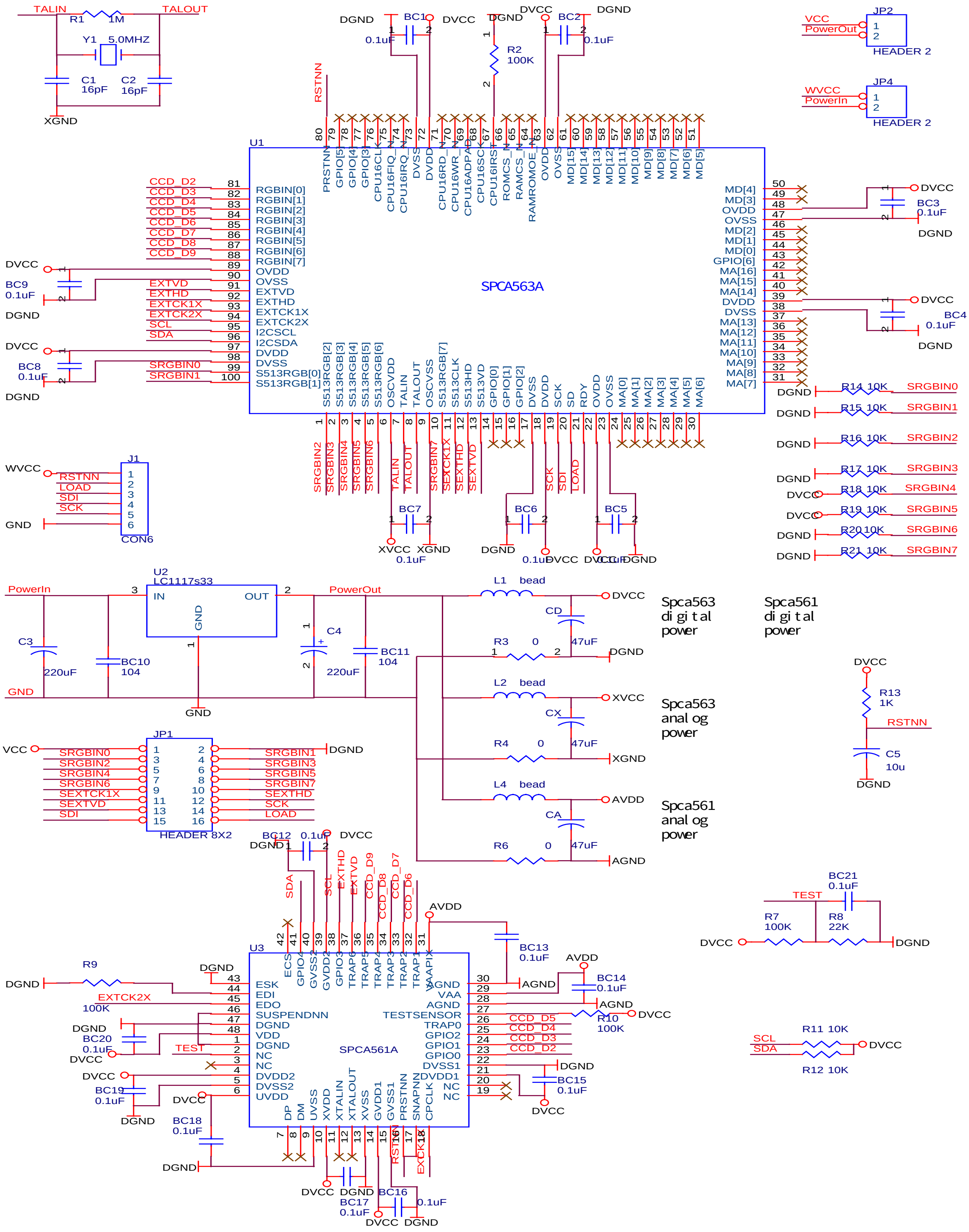
```
//语法：调用
//描述：获得第二个目标物体的数据
//参数：无
//返回：无
//======================================================
.public   F_LookGetObj2ndData;
F_LookGetObj2ndData:
                    call  F_GetObj2ndData;
                    call  F_LookCheckTheObj;
                    retf;
//======================================================
//函数：F_LookGetObj3rdData
//语法：调用
//描述：获得第三个目标物体的数据
//参数：无
//返回：无
//======================================================
.public   F_LookGetObj3rdData;
F_LookGetObj3rdData:
                    call  F_GetObj3rdData;
                    call  F_LookCheckTheObj;
                    retf;
//======================================================
//函数：F_LookGetObj4thData
//语法：调用
//描述：获得第四个目标物体的数据
//参数：无
//返回：无
//======================================================
.public   F_LookGetObj4thData;
```

```
F_LookGetObj4thData:
                        call  F_GetObj4thData;
                        call  F_LookCheckTheObj;
                        retf;
//====================================================
//函数：F_LookGetObj5thData
//语法：调用
//描述：获得第五个目标物体的数据
//参数：无
//返回：无
//====================================================
.public    F_LookGetObj5thData;
F_LookGetObj5thData:
                        call  F_GetObj5thData;
                        call  F_LookCheckTheObj;
                        retf;
//====================================================
//函数：F_LookGetObj6thData
//语法：调用
//描述：获得第六个目标物体的数据
//参数：无
//返回：无
//====================================================
.public    F_LookGetObj6thData;
F_LookGetObj6thData:
                        call  F_GetObj6thData;
                        call  F_LookCheckTheObj;
                        retf;
//====================================================
//函数：F_LookGetObj7thData
```

```
//语法：调用
//描述：获得第七个目标物体的数据
//参数：无
//返回：无
//=====================================================
.public   F_LookGetObj7thData;
F_LookGetObj7thData:
                    call  F_GetObj7thData;
                    call  F_LookCheckTheObj;
                    retf;
//=====================================================
//函数：F_LookCheckTheObj
//语法：调用
//描述：获得在合适的范围内的颜色、外型、远近的数值
//参数：无
//返回：无
//=====================================================
.public   F_LookCheckTheObj;
F_LookCheckTheObj:
                    r1 = [R_Shape];
                    cmp r1,0x00;
                    je    L_LookNotRightShape;

                    r1 = [R_Shape];
                    cmp r1,0x05;
                    jae   L_LookNotRightShape;

                    r1 = [R_AreaH];
                    cmp r1,0x01;
                    jb    L_LookNotRightShape;
```

```
L_LookHaveFoundTheObj:
                r1 = [R_PreAreaH];
                cmp r1,0x00;
                je     L_Fund1stObj;

                r1 = [R_AreaH];
                cmp r1,[R_PreAreaH];
                jb L_LookNotRightShape;
L_Fund1stObj:
                r1 = [R_Color];
                [R_PreColor] = r1;

                r1 = [R_Shape]
                [R_PreShape] = r1;

                r1 = [R_AreaH];
                [R_PreAreaH] = r1;

                r1 = [R_CompX];
                [R_PreCompX] = r1;
                r1 = [R_CompY];
                [R_PreCompY] = r1;

                r1 = [R_SeekFlag];
                r1 |= 0x01;
                [R_SeekFlag] = r1;
                retf;
L_LookNotRightShape:
                retf;
```

8 DEMO 板原理图


TALIN
TALOUT
R1 1M
Y1 5.0MHZ
C1 16pF
C2 16pF
XGND
DGND
BC1 0.1uF
DVCC
BC2 0.1uF
R2 100K
RSTNN
VCC
PowerOut
JP2
HEADER 2
WVCC
PowerIn
JP4
HEADER 2
U1
SPCA563A
CCD_D2
CCD_D3
CCD_D4
CCD_D5
CCD_D6
CCD_D7
CCD_D8
CCD_D9
RGBIN[0]
RGBIN[1]
RGBIN[2]
RGBIN[3]
RGBIN[4]
RGBIN[5]
RGBIN[6]
RGBIN[7]
OVDD
OVSS
EXTVD
EXTHD
EXTCK1X
EXTCK2X
I2CSCL
I2CSDA
DVDD
DVSS
S513RGB[0]
S513RGB[1]
SCL
SDA
SRGBIN0
SRGBIN1
BC9 0.1uF
BC8 0.1uF
MD[4]
MD[3]
GPIO[6]
MA[16]
MA[15]
MA[14]
MA[13]
MA[12]
MA[11]
MA[10]
MA[9]
MA[8]
MA[7]
BC3 0.1uF
BC4 0.1uF
R14 10K
R15 10K
R16 10K
R17 10K
R18 10K
R19 10K
R20 10K
R21 10K
SRGBIN2
SRGBIN3
SRGBIN4
SRGBIN5
SRGBIN6
SRGBIN7
BC7 0.1uF
BC6 0.1uF
BC5 0.1uF
XVCC
J1
CON6
LOAD
SDI
SCK
GND
U2
LC1117s33
IN
OUT
C3 220uF
BC10 104
C4 220uF
BC11 104
L1 bead
CD 47uF
R3 0
L2 bead
CX 47uF
R4 0
L4 bead
CA 47uF
R6 0
AVDD
AGND
Spca563 digital power
Spca561 digital power
Spca563 analog power
Spca561 analog power
R13 1K
C5 10u
JP1
HEADER 8X2
SEXTCK1X
SEXTVD
SEXTHD
BC12 0.1uF
U3
SPCA561A
BC13 0.1uF
BC14 0.1uF
R10 100K
BC15 0.1uF
R9 100K
EXTCK2X
BC20 0.1uF
TEST
BC19 0.1uF
BC18 0.1uF
BC17 0.1uF
BC16 0.1uF
BC21 0.1uF
R7 100K
R8 22K
R11 10K
R12 10K