FLEX: IMAGE ACQUISITION FIRMWARE FOR VIDEO SMART SENSOR

Author: Dario Di Stefano (University of Pisa)

Abstract

FLEX, born due to collaborative effort between Evidence Srl and Embedded Solutions Srl, is an electronic platform for real-time embedded applications. A firmware has been written in C30 for the FLEX on-board dsPIC33FJ256MC710 microcontroller (made by Microchip) for automatic acquisition of images. The main purposes of this firmware are to configure peripherals of the microcontroller and to manage image transfer using a remote camera, which is equipped with an RS232-like interface (on CMOS levels). This firmware was developed using Round Solution's CAMVGA100 module with a built-in JPEG compressor engine.

Index

1. Flex Board

FLEX is a platform board for embedded modular systems that exploits the potential of Microchip family's dsPIC DSC microcontroller. It is an ideal board for developing real-time applications because the on-board dsPIC33FJ256MC710 microcontroller supports advanced real-time kernel, such as the Evidence Srl's Erika Enterprise.

The basic configuration of FLEX is characterized by numerous pins (refer Figure 1 below); most of these connectors match with the pins of dsPIC. Moreover, there are power connectors to facilitate the use of complementary circuits.

The major electronic components of the FLEX Light version are:
• Microchip dsPIC (R) DSC microcontroller.
• Connector for programmer Microchip ICD2.
• Power Connector.
• LEDs tracking some features of the card.
• Connectors for Daughter boards piggybacking.

For more information on FLEX boards, visit http://www.evidence.eu.com/

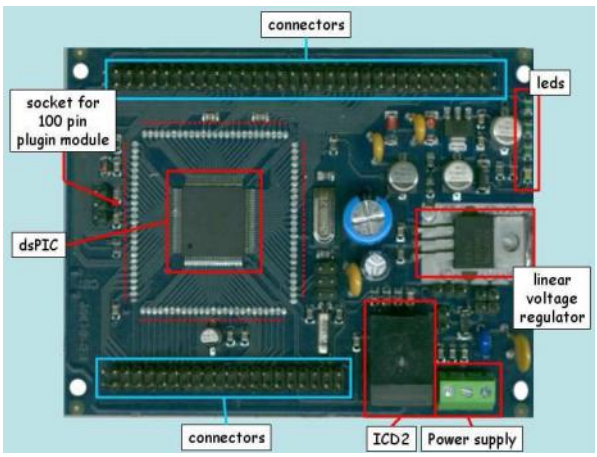

Figure 1: FLEX Light

2. CAMVGA100 module

The CAMVGA100 JPEG Compression Module performs as a video camera or a JPEG compressed still camera and, it can be attached to a host. Users can send out a snapshot command from the host in order tc capture a full resolution single-frame still picture. The picture is then compressed by the JPEG engine and transferred to the host via a serial link (UART on CMOS 3.3 V level).

The main features of CAMVGA100 (refer Figure 2 below) are:

- Is a Low-cost and low-powered solution for high resolution image capture.
- Has a built-in down-sampling, clamping and windowing circuits for VGA CIF SIF QCIF 160x128 anc 80x64 image resolutions.
- Uses RS-232 on CMOS level 3,3 Volt: 115.2K bps for transferring JPEG still pictures or 160x128 preview @8bpp with 0.75~6 fps.
- Has JPEG CODEC for different resolutions.
- Has a built-in color conversion circuit for 2-bit gray, 4-bit gray, 8-bit gray, 12-bit RGB, 16-bit RGB preview images.
- It auto detects baud rate while connecting the host.

The camera can transmit the image (in a chosen format) at several bus speeds.

For more information on CAMVGA100, visit http://www.roundsolutions.com/

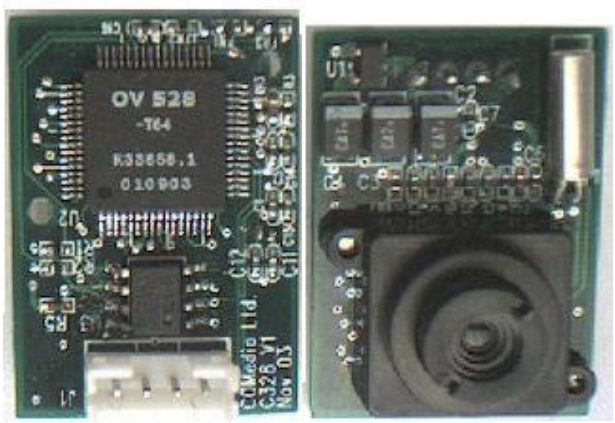

Figure 2: CAMVGA100

Figures 3 to 7 below describe the procedures for synchronization, image acquisition, image compression, and images decompression.

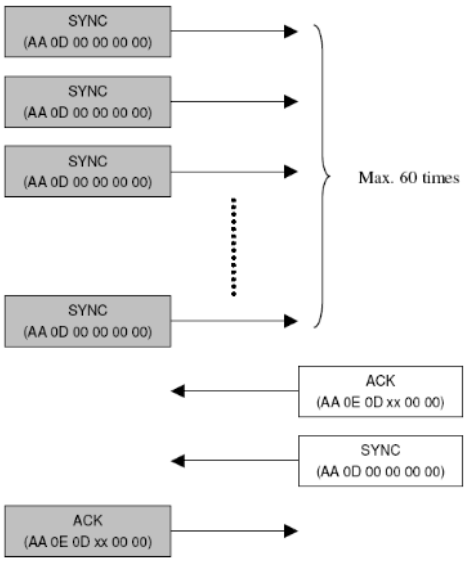

Figure 3: Synchronization procedure

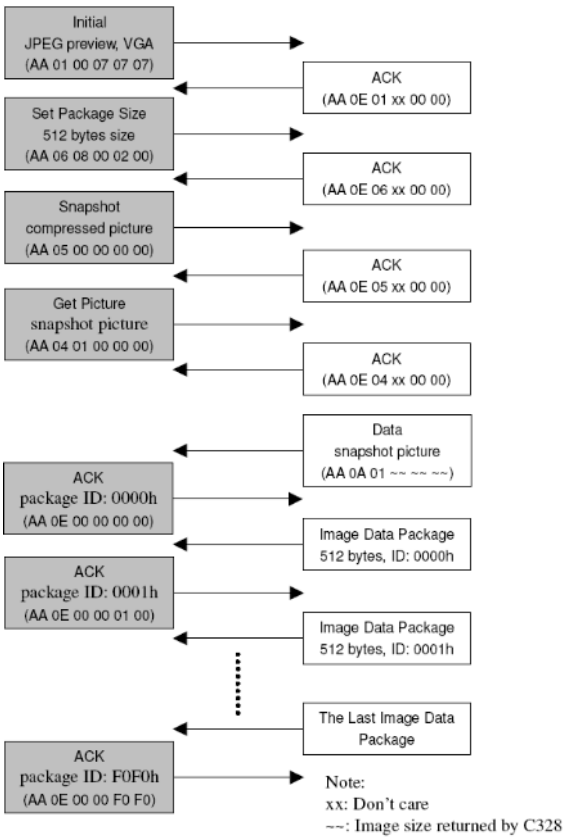

Figure 4: JPEG image acquisition procedure

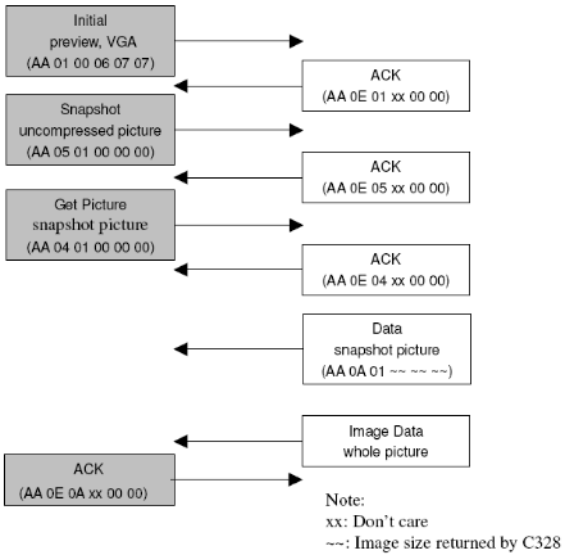

Figure 5: Uncompressed image acquisition procedure

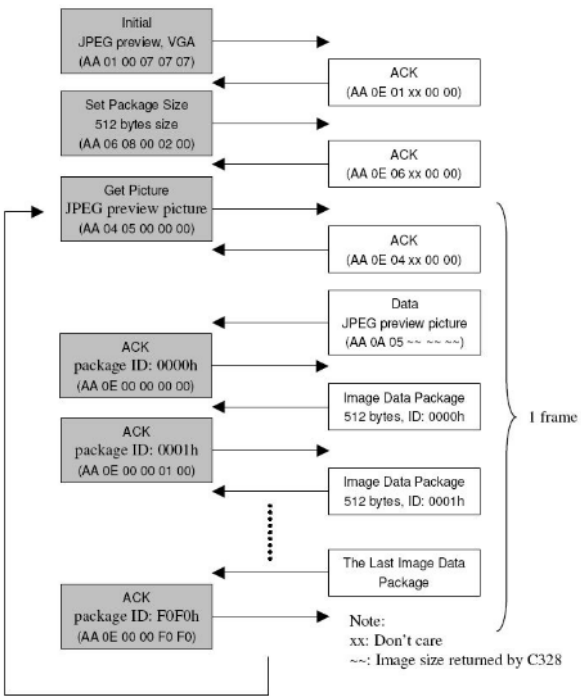

Figure 6: JPEG sequence acquisition

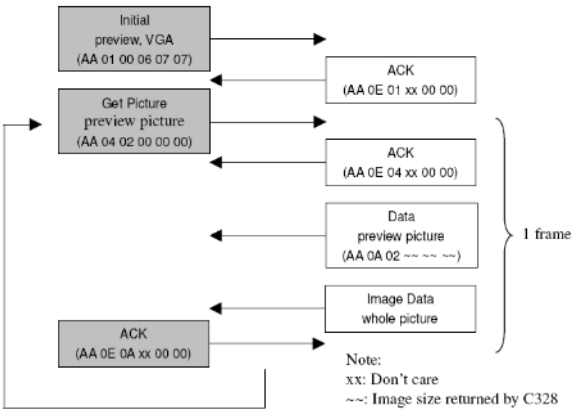

Figure 7: Uncompressed sequence acquisition procedure

Some clarification regarding RS-232 serial communication: The standard involves voltage levels of ± 12V but the camera supports only a direct asynchronous communication on CMOS levels. A conversion board is required to adapt to this standard, so as to ensure consistency with the computers COM port driver . The board (refer Figur 8 below) has a serial connector and a chip ADM3222AN, which makes the transformation request through appropriate capacitors that are mounted with a charge pump scheme.

Figure 8: Serial Adapter

3. Firmware Description

This section describes the functions and data structures used in the firmware.

Note

1. The code has been written for Microchip dsPIC33FJ256MC710 microcontroller and tested only on this device.
2. The code uses dsPIC33FJ256MC710 peripherals like UART1, UART2, TIMER8, and TIMER9. The firmware is not recommended to be used on devices which do not have such peripherals.
3. The application receives image from the camera and sends it to a remote PC and for this, the PC-FLEX serial bus baud-rate must be higher than the FLEX-CAMVGA100 baud-rate as lower baud-rates would lead to hardware incompatibility. The recommended baud-rates are 9600 bps on FLEX-CAMVGA100 side and 19200 bps on PC-FLEX side.

ATT!!!

The firmware uses UART1 module to transfer images to a remote PC and UART2 module to communicate with the CAMVGA100 module. The modu TIMER8 and TIMER9 are jointly used, resulting in a single 32-bit timer to create TIMED wait functions in order to prevent indefinite blockage of th program.

The data received by the camera are stored during the execution of the UART2 ISR. The routine starts after the complete receipt of each byte.

To maintain similarity with internal built-in dsPIC modules, the firmware is based on global register-like variables that contain information about the status and configuration of the camera module, in order to have the desired image format.

The following pages provide detailed description on the firmware's functions, data structures and global variables.

CAMSTATUS																
N° bit	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Name	ERROR								IMAGE	SYNC	NAK	DATA	ACK	WAIT	READY	SYNCHRO
Type	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Legend:
R = Readable bit; W = Writable bit; U = Unimplemented bit (read as '0'); '-1' = Bit is set (default); '-0' = Bit is cleared (default); '-x' = Bit is unknown

SYNCHRO (BIT 00): Synchronization bit.
1 = Synchronization completed successfully;
0 = Camera not synchronized;

READY(BIT 01): Command Reception bit.
1 = Command received;
0 = Command not received;

WAIT (BIT 02): Command wait bits
1 = CPU is waiting for the Camera response command: ack; data; nak; sync.
0 = CPU isn't waiting for a generic Camera response;

ACK (BIT 03): ACK command Reception bit.
1 = Camera module is waiting the ACK command;
0 = Camera module isn't waiting the ACK command;

DATA (BIT 04): DATA command Reception bit.
1 = Camera module is waiting the DATA command;
0 = Camera module isn't waiting the DATA command;

NAK(BIT 05): NAK command Reception bit.
1 = Camera module is waiting the NAK command either there is an error;
0 = Camera module isn't waiting the NAK command either there isn't an error;

SYNC (BIT 06): SYNC command Reception bit.
1 = Camera module is waiting the SYNC command;
0 = Camera module isn't waiting the SYNC command;

IMAGE (BIT 07): IMAGE Reception bit.
1 = Camera module is waiting an image;
0 = Camera module isn't waiting an image;

ERROR (BIT 08-09-10-11-12-13-14-15)

CAM_ERR_PICTURE_TYPE		0x01
CAM_ERR_PICTURE_UP_SCALE	0x02	
CAM_ERR_PICTURE_SCALE_ERROR		0x03
CAM_ERR_UNEXPECTED_REPLY	0x04	
CAM_ERR_SEND_PICTURE_TIMEOUT		0x05
CAM_ERR_UNEXPECTED_COMMAND		0x06
CAM_ERR_SRAM_JPEG_TYPE_ERROR		0x07
CAM_ERR_SRAM_JPEG_SIZE_ERROR		0x08
CAM_ERR_PICTURE_FORMAT_ERROR	0x09	
CAM_ERR_PICTURE_SIZE_ERROR		0x0a
CAM_ERR_PARAMETER_ERROR	0x0b	
CAM_ERR_SEND_REGISTER_TIMEOUT	0x0c	
CAM_ERR_COMMAND_ID_ERROR		0x0d
CAM_ERR_PICTURE_NOT_READY		0x0f
CAM_ERR_TRANSFER_PACKAGE_NUMBER		0x10
CAM_ERR_TRANSFER_PACKAGE_SIZE	0x11	
CAM_ERR_COMMAND_HEADER_ERROR	0xF0	
CAM_ERR_COMMAND_LENGTH_ERROR	0xF1	
CAM_ERR_SEND_PICTURE_ERROR		0xF5
CAM_ERR_SEND_COMMAND_ERROR		0xFF
CAM_ERR_SW_ERROR0	0xE0	
CAM_ERR_SW_ERROR1	0xE1	
CAM_ERR_SW_ERROR2	0xE2	
CAM_ERR_SW_ERROR3	0xE3	
CAM_ERR_SW_ERROR4	0xE4	
CAM_ERR_SW_ERROR5	0xE5	
CAM_ERR_SW_ERROR6	0xE6	
CAM_ERR_SW_ERROR7	0xE7	
CAM_ERR_SW_ERROR8	0xE8	
CAM_ERR_SW_ERROR9	0xE9	
CAM_ERR_SW_ERROR10	0xEA	
CAM_ERR_SW_ERROR11	0xEB	
CAM_ERR_SW_ERROR12	0xEC	

CAMCONFIG																
N° bit	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Name	SNAPTYPE	BAUDDIV			SEND	PICTYPE			JPEGRES			PREVRES		COLOR		
Type	R/W-1	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1	R/W-1

Legend:
R = Readable bit; W = Writable bit; U = Unimplemented bit (read as '0'); '-1' = Bit is set (default); '-0' = Bit is cleared (default); '-x' = Bit is unknown

COLOR (BIT 00-01-02): Colour type bits.
000 = Nothing;
001 = 2-bit Gray Scale 01h;
010 = 4-bit Gray Scale 02h;
011 = 8-bit Gray Scale 03h;
100 = Nothing;
101 = 12-bit Color 05h;
110 = 16-bit Color 06h;
111 = JPEG 07h;

PREVRES (BIT 03-04-05): Preview resolution bits.
000 = Nothing;
001 = 80x60 01h;
010 = Nothing;
011 = 160x120 03h;
100 = Nothing;
101 = 320x240 05h; (?)
110 = Nothing;
111 = 640x480 07h; (?)

JPEGRES (BIT 06-07-08): JPEG resolution bits.
000 = Nothing;
001 = 80x64 01h;
010 = Nothing;
011 = 160x128 03h;
100 = Nothing;
101 = 320x240 05h;
110 = Nothing;
111 = 640x480 07h;

PICTYPE (BIT 09-10): Picture type bits.
00 = Nothing;
01 = Snapshot Picture (Uncompressed/compressed picture, see SNAPTYPE) 01h;
10 = Preview Picture (Uncompressed Video) 02h;
11 = JPEG Preview Picture (Compressed Video) 05h; (changed)

SEND (BIT 11): Serial communication management bit.
1 = The image will be transmitted by UART1;
0 = The image will be recorded in data memory;
ATT!!! The camera extern module uses the UART2 and may use the UART1 module to send images and to receive data.

BAUDDIV (BIT 12-13-14): UART baud-rate selection bits (1st divider).
000 = 7200 bps;
001 = 9600 bps;
010 = 14400 bps;
011 = 19200 bps;
100 = 28800 bps;
101 = 38400 bps;
110 = 57600 bps;
111 = 115200 bps;

SNAPTYPE (BIT 15): Snapshot type bit.
1 = Uncompressed Picture 01h;
0 = Compressed Picture 00h;

CAMFRAMESIZE																
N° bit	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Name	-	-	-	-			PACKSIZE									
Type	U-0	U-0	U-0	U-0	U-0	U-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend:
R = Readable bit; W = Writable bit; U = Unimplemented bit (read as '0'); '-1' = Bit is set (default); '-0' = Bit is cleared (default); '-x' = Bit is unknown

PACKSIZE (BIT 00-01-02-03-04-05-06-07-08-09): Package size bits. The value is bounded: [16,512].
(See the CAMVGA100 user manual)

0000000000 = Nothing;
…
0000001111 = Nothing;
0000010000 = 16 Bytes (each) compressed picture package (Min value);
0000010001 = 17 Bytes (each) compressed picture package;
…
1000000000 = 512 Bytes (each) compressed picture package (Max value);
1000000001 = Nothing;
…
1111111111 = Nothing;

CAMDROPFRAMES																
N° bit	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Name	-	-	-	-	SNAPSKIP											
Type	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend:
R = Readable bit; W = Writable bit; U = Unimplemented bit (read as '0'); '-1' = Bit is set (default); '-0' = Bit is cleared (default); '-x' = Bit is unknown

SNAPSKIP (BIT 11-12-13-14): Snapshot skip bits.

0x0000 = keeps the current frame;
0x0001 = captures the next frame;
…
0x000F = captures the 15° frame;
…
0xFFFF = captures the 65535° frame;

CAMRBCOUNTER																
N° bit	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08	Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Name	RBCOUNT															
Type	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

CAMRBCOUNTER: This variable is the received bytes U2RX interrupt counter.

The module also uses other global variables as listed below:

IMAGECOUNTER (unsigned long int): This is the Image vector counter used during the JPEG image transmission.

IMAGESIZE (unsigned long int): This is the expected image size. Its value is a part of the CAMVGA100 DATA command.

VIDEOSIZE (unsigned long int): This is the video counter (number of the images).

CAMSTOP (unsigned char): This is a variable used to stop the images wait.

CAMFCY (float): This is a variable used to record the CPU instructions frequency.

CAMBUF[6] (unsigned int): This is the array used like a buffer to record the 6 command bytes.

BCOUNTER (unsigned long int): This is the counter to recognize the end of an image transmission.

NEWLIMIT (unsigned long int): This is a variable used during the transmission of an uncompressed image.

SEND (unsigned char): This is another variable used during the transmission of an uncompressed image.

CTYPE (unsigned char): This is a variable to record the image compression type.

The following pages provide detailed description on the structures and functions used by the Camera extern module.

COMMAND STRUCTURE:

```
typedef struct CAMERACOMMAND
{
        unsigned int  Id;
        unsigned long int Cam_Cmd;
} CAMCMD;
```

The structure provides an unsigned int Id field for the command header and an unsigned long int Cam_Cmd field to specify the body of the same.

This structure has been used to send commands to the camera, as indicated in Table below:

Command	ID Number	Parameter1	Parameter2	Parameter3	Parameter4
Initial	AA01h	00h	Color Type	Preview Resolution	JPEG Resolution
Get Picture	AA04h	Picture Type	00h	00h	00h
Snapshot	AA05h	Snapshot Type	Skip Frame Low Byte	Skip Frame High Byte	00h
Set Package Size	AA06h	08h	Package Size Low Byte	Package Size High Byte	00h
Set Baudrate	AA07h	1st Divider	2nd Divider	00h	00h
Reset	AA08h	Reset Type	00h	00h	xxh*
Power Off	AA09h	00h	00h	00h	00h
Data	AA0Ah	Data Type	Length Byte 0	Length Byte 1	Length Byte 2
SYNC	AA0Dh	00h	00h	00h	00h
ACK	AA0Eh	Command ID	ACK counter	00h / Package ID Byte 0	00h / Package ID Byte 1
NAK	AA0Fh	00h	NAK counter	Error Number	00h

* If the parameter is 0xFF, the command is a special Reset command and the firmware responds to it immediately.

The commands can be sent out with the function (described below): unsigned char Cam_SendCmd (CAMCMD * Hex_Cmd).

unsigned char Cam_InitialCmd(void)

This function modifies the main global variable CAMSTATUS and reads the information contained in the variable CAMCONFIG, initially modified by the user, to set the field Cam_Cmd of data structure CamInit. In particular, the function can read information about color, and resolution. CamInit is the data structure used by the function Cam_SendCmd (…), to prepare the transfer of the INITIAL command.

```
unsigned char Cam_InitialCmd(void)
{
     unsigned char pictype, snaptype;
     unsigned long int color, prevres, jpegres;
     CAMCMD CamInitial;

       CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

     pictype = (CAMCONFIG>>PICTYPE_FLAG) & PICTYPE_NUM;
     if( pictype==0 ) return 0;

     snaptype = (CAMCONFIG>>SNAPTYPE_FLAG) & SNAPTYPE_NUM;

     color = CAMCONFIG & COLOR_NUM;
     if(color == 0 || color == 4)  return 0;

     prevres = (CAMCONFIG>>PREVRES_FLAG) & PREVRES_NUM      ;
     if(prevres == 0 || prevres == 2 || prevres == 4 ||  prevres == 6) return 0;

     jpegres = (CAMCONFIG>>JPEGRES_FLAG) & JPEGRES_NUM;
     if(jpegres == 0 || jpegres == 2 || jpegres == 4 ||  jpegres == 6) return 0;

     CamInitial.Id = CAM_CMD_INITIAL;
     CamInitial.Cam_Cmd = (color<<16) + (prevres<<8) + jpegres;

     return Cam_SendCmd(&CamInitial);
}
```

unsigned char Cam_AckCmd(unsigned longint ackcmd)

This function modifies the main global variable CAMSTATUS and copies the field Cam_Cmd of data structure CamAck, the value ackcmd. CamAck is the data structure used by the function Cam_SendCmd (…),to prepare the transfer of the ACK command.

```
unsigned char Cam_AckCmd(unsigned long int ackcmd)
{
     CAMCMD CamAck;

     CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

       CamAck.Id = CAM_CMD_ACK;
     CamAck.Cam_Cmd = ackcmd;

     return Cam_SendCmd(&CamAck);
}
```

unsigned char Cam_PackSizeCmd(void)

This function modifies the main global variable CAMSTATUS and reads the information contained in the variable CAMFRAMESIZE, initially modified by the user, to set the field Cam_Cmd of data structure CamPackSize. In particular, the function goes to read the information on the package size used for the JPEG images transmission. CamPackSize is the structure used by the function Cam_SendCmd (…) to prepare the transfer of the SET SIZE PACKAGE command.

```
unsigned char Cam_PackSizeCmd(void)
{
     unsigned long int lowbyte, highbyte, packsize;
     CAMCMD CamPackSize;

       CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

     packsize = CAMFRAMESIZE & PACKSIZE_NUM;

     if(packsize<16)
     {
         packsize = 16;
         CAMFRAMESIZE = CAMFRAMESIZE & 0xFC00;
         CAMFRAMESIZE = CAMFRAMESIZE | 0x0010; // packsize=16
     }
     else if(packsize>512)
     {
         packsize = 512;
         CAMFRAMESIZE = CAMFRAMESIZE & 0xFC00;
         CAMFRAMESIZE = CAMFRAMESIZE | 0x0200; // packsize=512
     }

     highbyte = (packsize>>8)&x000000FF;
     lowbyte = packsize & 0x000000FF;

     CamPackSize.Id = CAM_CMD_SETPACKETSIZE;
     CamPackSize.Cam_Cmd = 0x08000000 +(lowbyte<<16) + (highbyte<<8);

     return Cam_SendCmd(&CamPackSize);
}
```

unsigned char Cam_GetCmd(void)

This function modifies the main global variable CAMSTATUS and reads the information contained in the variable CAMCONFIG, initially modified by the user, to set the field Cam_Cmd of data structure CamGet. In particular, the function can read the information about the requested streaming type (video or snapshot). CamGet is the structure used by the function Cam_SendCmd (…),to prepare the transfer of the GET PICTURE command.

```
unsigned char Cam_GetCmd(void)
{
     unsigned long int pictype;
     CAMCMD CamGet;

       CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

     pictype = (CAMCONFIG>>PICTYPE_FLAG) & PICTYPE_NUM;
     if( pictype == DATATYPE_JPEG_PREVIEW  ) pictype = 0x05;

     CamGet.Id = CAM_CMD_GETPICTURE;
     CamGet.Cam_Cmd = pictype<<24;

     return Cam_SendCmd(&CamGet);
}
```

unsigned char Cam_ResetCmd(unsigned longint restype,unsigned longint spreset)

This function modifies the main global variable CAMSTATUS and uses the arguments to set the field Cam_Cmd of data structure CamReset to specify the desired reset command type. CamReset is the structure used by the function Cam_SendCmd (…),to prepare the transfer of the RESET command.

```
unsigned char Cam_ResetCmd(unsigned long int restype,unsigned long int  spreset)
{
     CAMCMD CamReset;
     CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

     if( (restype!=RESET_SYSTEM && restype!=RESET_STATE_MACHINE) ||
         (spreset!=SIMPLE_RESET && spreset!=SPECIAL_RESET) ) return  0;

     CamReset.Id = CAM_CMD_RESET;
     CamReset.Cam_Cmd = (restype<<24) + spreset;

     return Cam_SendCmd(&CamReset);
}
```

unsigned char Cam_PowerOffCmd(void)

This function modifies the main global variable CAMSTATUS and prepares the CamPowerOff command. CamPowerOff once again represents the structure used by the function Cam_SendCmd (…),to prepare the transfer of the POWER OFF command. Please note that this command determines the passage in CAMERA SLEEP mode causing inevitably loss of synchronization.

```
unsigned char Cam_PowerOffCmd(void)
{
     CAMCMD CamPowerOff;

     CamPowerOff.Id = CAM_CMD_POWEROFF;
     CamPowerOff.Cam_Cmd = 0x00000000;

     if(Cam_SendCmd(&CamPowerOff))
     {
         CAMSTATUS = 0; // Camera Synchronization lost.
         return 1;
     } else return 0;
}
```

unsigned char Cam_SnapshotCmd(void)

This function modifies the main global variable CAMSTATUS and reads the information contained in the variable CAMCONFIG and in the variable CAMDROPFRAMES, to set the Cam_Cmd field of data structure CamSnap. In particular, the function can read the information about the compression type contained in the variable CAMCONFIG and the number of dropped frames contained in the variable CAMDROPFRAMES. CamSnap is the structure used by the function Cam_SendCmd (…),to prepare the transfer of the SNAPSHOT command.

```
unsigned char Cam_SnapshotCmd(void)
{
     unsigned long int snaptype, snapskipLB, snapskipHB;
     CAMCMD CamSnap;

     CAMSTATUS = CAMSTATUS & 0xFFFE; // CAMSTATUS Initialization

     snaptype = (CAMCONFIG>>SNAPTYPE_FLAG) & SNAPTYPE_NUM;
     snapskipLB = (unsigned long int)(CAMDROPFRAMES & 0x00FF);
     snapskipHB = (unsigned long int)((CAMDROPFRAMES>>8) & 0x00FF);

     CamSnap.Id = CAM_CMD_SNAPSHOT;
     CamSnap.Cam_Cmd = (snaptype<<24) + (snapskipLB<<16) + (snapskipHB<<8);

     return Cam_SendCmd(&CamSnap);
}
```

unsigned char Cam_SyncCmd(void)

This function modifies the main global variable CAMSTATUS and creates the CamSync command. CamSync is the structure used by the function Cam_SendCmd (…),to prepare the transfer of the SYNCHRONIZATION command.

```
unsigned char Cam_SyncCmd(void)
{
     CAMCMD CamSync;

       CAMSTATUS = 0; // CAMSTATUS Initialization

       CamSync.Id = CAM_CMD_SYNC;
     CamSync.Cam_Cmd = 0x00000000;

     return Cam_SendCmd(&CamSync);
}
```

void UART1_Config(unsigned int baudvalue)

This function configures the dsPIC33F UART1 module. In particular it enables microcontroller to receive the channel Rx ISR (Tx interrupt channel is unused) every time the buffer is full (4 bytes). The UART1 baud-rate value must be passed as argument. Finally, the chosen serial communication type has 8-bit data for the field, 1 stop bit and no parity bits.

```
void UART1_Config(unsigned int baudvalue)
{
      unsigned int U1MODEvalue; // Holds the value of uart config reg
      unsigned int U1STAvalue;  // Holds the information regarding uart TX & RX interrupt modes
      CloseUART1(); // Turn off  UART1 module

      // Configure uart1 receive and transmit interrupt
      IFS0bits.U1RXIF = 0;
      ConfigIntUART1(UART_RX_INT_EN & UART_RX_INT_PR3 & UART_TX_INT_DIS & UART_TX_INT_PR2);

      U1MODEvalue =     UART_EN & UART_UEN_00 &  UART_IrDA_DISABLE & UART_BRGH_SIXTEEN & UART_UXRX_IDLE_ONE &
      UART_IDLE_CON & UART_DIS_WAKE
                              & UART_DIS_LOOPBACK & UART_DIS_ABAUD & UART_NO_PAR_8BIT & UART_1STOPBIT;
      U1MODEvalue = U1MODEvalue & 0xFFF9; //becuase the UART_NO_PAR_8BIT mask gives  a wrong result
      U1STAvalue = UART_TX_ENABLE  & UART_ADR_DETECT_DIS  & UART_SYNC_BREAK_DISABLED  & UART_INT_RX_BUF_FUL  &
      UART_IrDA_POL_INV_ZERO;
      OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
      return;
}
```

void UART2_Config(unsigned int baudvalue)

This function configures the dsPIC33F UART2 module using also Microchip C30 functions. In particular it enables microcontroller to receive the channel Rx ISR (Tx interrupt channel is unused) every time a new byte is arrived. The UART2 baud-rate value must be passed as argument. Finally, the chosen serial communication type has 8-bit data for the field, 1 stop bit and no parity bits.

```
void UART2_Config(unsigned int baudvalue)/* Holds the value of baud register */
{

      unsigned int U2MODEvalue;  /* Holds the value of uart config reg */
      unsigned int U2STAvalue;   /* Holds the information regarding uart TX & RX
                                    interrupt modes */

      CloseUART2(); /* Turn off  UART2 module */

      /* Configure uart2 receive and transmit interrupt */
      IFS1bits.U2RXIF = 0;
      ConfigIntUART2(UART_RX_INT_EN & UART_RX_INT_PR5 & UART_TX_INT_DIS & UART_TX_INT_PR2);

      U2MODEvalue =     UART_EN & UART_UEN_00 &  UART_IrDA_DISABLE & UART_BRGH_SIXTEEN & UART_UXRX_IDLE_ONE  &
      UART_IDLE_CON & UART_DIS_WAKE
                              & UART_DIS_LOOPBACK & UART_DIS_ABAUD & UART_NO_PAR_8BIT & UART_1STOPBIT;
      U2MODEvalue= U2MODEvalue & 0xFFF9; //becuase the UART_NO_PAR_8BIT mask gives a wrong result
      U2STAvalue = UART_TX_ENABLE  & UART_ADR_DETECT_DIS & UART_SYNC_BREAK_DISABLED & UART_INT_RX_CHAR &
      UART_IrDA_POL_INV_ZERO;
      OpenUART2(U2MODEvalue, U2STAvalue, baudvalue);
      return;
}
```

void Cam_TIMER89Config(unsigned longint match_value)

This function configures the 32-bit dsPIC33F T89 timer using also Microchip C30 functions. The period must be passed as argument and when the current value of the timer coincides with the period, the module is configured to activate an ISR T9. The ISR, which can be found in the file "CameraMain.c", writes 0 in the WAIT flag to communicate that the timer has completed the period at least once.

```
void Cam_TIMER89Config(unsigned long int match_value)
{

	IFS3bits.T9IF = 0; // Clear Timer9 interrupt flag
	Cam_ConfigIntTimer89(T9_INT_PRIOR_4 & T9_INT_ON);

	Cam_WriteTimer89(0);

	Cam_OpenTimer89(T8_OFF & T8_GATE_OFF & T8_PS_1_256 & T8_32BIT_MODE_ON & T8_SOURCE_INT, match_value);
	return;
}
```

unsigned char Cam_ReadData(void)

This function starts every time a new byte, sent by the camera, is inserted into the buffer. This function starts inside the UART2 Rx interrupt routine. The function execution depends on the IMAGE flag because, according to the microcontroller is awaiting a command or an image, it must to follow different paths. If the microcontroller is awaiting a command the Cam_Readmd() function starts after the sixth received byte to recognize the command type, while if the microcontroller is awaiting an image the Cam_ReadDataImage() function starts after receipt of each image byte to store the received byte in the global vector FRAME. Another important feature of this function is the proper and fast setting of the IMAGE flag to avoid losing consistency as soon as the camera starts to send image data.

```
unsigned char Cam_ReadData(void)
{
    unsigned char xret=1;

      //rst[0] = (unsigned int)(ReadUART2()&0x00FF);
    CAMBUF[CAMRBCOUNTER] = (unsigned int)(ReadUART2()&0x00FF);
      CAMRBCOUNTER++;

      if( (CAMSTATUS>>IMAGE_FLAG)&IMAGE_NUM )
    {
        CAMRBCOUNTER = 0;
        xret = Cam_ReadDataImage();
    }
    else if( CAMRBCOUNTER == 6 )
    {
            if( (CAMSTATUS>>DATA_FLAG)&DATA_NUM )
                CAMSTATUS = CAMSTATUS | (IMAGE_NUM<<IMAGE_FLAG);      /* Set image flag */
            CAMRBCOUNTER = 0;
        xret = Cam_ReadCmd();
    }
    return xret;
}
```

unsigned char Cam_ReadCmd(void)

This function can recognize the received command type (sent by the camera).

```
unsigned char Cam_ReadCmd(void)
{
    unsigned int Cmd_ID;

    Cmd_ID = ((unsigned int)(CAMBUF[0]))<<;
    Cmd_ID = Cmd_ID + (unsigned int)(CAMBUF[1]);

    CAMSTATUS = CAMSTATUS | (READY_NUM<<READY_FLAG);            /* Set Ready command flag */

    if(Cmd_ID == CAM_CMD_ACK)
    {
        CAMSTATUS = CAMSTATUS & (~(ACK_NUM<<ACK_FLAG));         /* Clear ACK command flag */
        return 1;
    }
    if(Cmd_ID == CAM_CMD_NAK)
    {
        CAMSTATUS = CAMSTATUS & (~(NAK_NUM<<NAK_FLAG));         /* Clear NAK command flag */
        CAMSTATUS = CAMSTATUS & (~(ERROR_NUM<<ERROR_FLAG));     /* Clear err command flag */
            CAMSTATUS = CAMSTATUS + (CAMBUF[4]<<ERROR_FLAG);    // Error number
        return 0;
    }
    if(Cmd_ID == CAM_CMD_DATA)
    {
        CAMSTATUS = CAMSTATUS & (~(DATA_NUM<<DATA_FLAG));       /* Clear DATA command flag */
        return 1;
    }
    if(Cmd_ID == CAM_CMD_SYNC)
    {
        CAMSTATUS = CAMSTATUS & (~(SYNC_NUM<<SYNC_FLAG));       /* Clear SYNC command flag */
        return 1;
    }
    if( (CAMSTATUS>>ERROR_FLAG)&ERROR_NUM ) return 0;

    Cam_SWError(CAM_ERR_SW_ERROR0);
    return 0;
}
```

unsigned char Cam_ReadDataImage(void)

This function is designed to store image bytes in the FRAME vector resetting simultaneously the T89 timer.

```
unsigned char Cam_ReadDataImage(void)
{
      unsigned char check=0;
      unsigned int packsize, pictype, snaptype;

      T8CONbits.TON = 0;      /* Disable Timer 8 */
      Cam_WriteTimer89(0);
      packsize = CAMFRAMESIZE & PACKSIZE_NUM;
      pictype = (CAMCONFIG >> PICTYPE_FLAG) & PICTYPE_NUM;
      snaptype = (CAMCONFIG>> SNAPTYPE_FLAG) & SNAPTYPE_NUM;

      T8CONbits.TON = 1;      /* Turn on Timer89 */
      if(CTYPE) FRAME[CAMCOUNTER] = CAMBUF[0];
      else FRAME[BCOUNTER] = CAMBUF[0];
      CAMCOUNTER++; BCOUNTER++;

      check= BCOUNTER==IMAGESIZE;
      if( (CAMCOUNTER >= packsize) || (check && BCOUNTER!=0) ) {
            if( (pictype==DATATYPE_PREVIEW) || (pictype==DATATYPE_SNAPSHOT && snaptype==UNCOMPRESSED_PICTURE)  ) {
                  if(check) CAMSTOP=1;
                  CAMCOUNTER = 0;
                  SEND=1;
            } else if( CAMCOUNTER>packsize ) {
                  Cam_SWError(CAM_ERR_SW_ERROR2); return 0;
            } else {
                  CAMSTATUS = CAMSTATUS | (READY_NUM<<READY_FLAG);      /* Set Ready command flag */
                  BCOUNTER = BCOUNTER-6;
                  JPEGDIM=CAMCOUNTER;
                  CAMCOUNTER = 0;
                  if(check) {
                        CAMSTOP=1;
                        CAMSTATUS = CAMSTATUS & (~(IMAGE_NUM<<IMAGE_FLAG));     /* Clear IMAGE flag */
                  }
            }
    }
    return 1;
}
```

unsigned char Cam_SendCmd(CAMCMD *Hex_Cmd)

This function creates a string of 6 bytes to send the commands to the camera by the function Send_by_UART2(…) .

```
unsigned char Cam_SendCmd(CAMCMD *Hex_Cmd)
{
    char sc[];
    while(BusyUART2());

    if( ((CAMSTATUS>>ERROR_FLAG)&ERROR_NUM) ) return 0;
    else
    {
        sc[0] = (char)( ((Hex_Cmd->Id)>>8) & 0x00FF );
        sc[1] = (char)(  (Hex_Cmd->Id) & 0x00FF );
        sc[2] = (char)( ((Hex_Cmd->Cam_Cmd)>>24) & 0x00FF );
        sc[3] = (char)( ((Hex_Cmd->Cam_Cmd)>>16) & 0x00FF );
        sc[4] = (char)( ((Hex_Cmd->Cam_Cmd)>>8) & 0x00FF );
        sc[5] = (char)(  (Hex_Cmd->Cam_Cmd) & 0x00FF );
                .
       Send_by_UART2((unsigned int *)sc,6); // Characters transimitted in hex.

    }
    return 1;
}
```

unsigned char Cam_SendPicture(void)

This function sends uncompressed image packages to a remote PC although the image transfer from the camera is not yet finished (to avoid loss of synchronization between two image requests).

.

```
unsigned char Cam_SendPicture(void)
{
      unsigned int packsize;
      char sd[];

      if(SEND==1)
      {
      SEND=0;
            packsize = CAMFRAMESIZE & PACKSIZE_NUM;
            NEWLIMIT+=(unsigned long int)packsize;
            if(NEWLIMIT>IMAGESIZE) NEWLIMIT=IMAGESIZE;
      }
      while(BusyUART1());
      if(TXCOUNTER<NEWLIMIT)
      {
            sd[0] = FRAME[TXCOUNTER];
            Send_by_UART1((unsigned int *)sd,1);
            TXCOUNTER++;
            if(TXCOUNTER==IMAGESIZE && IMAGESIZE!=0) return 0;
      }
      return 1;
}
```

unsigned char Cam_SendJPEGFrame(unsigned char send, unsigned int id)

This function check if the JPEG image packet is correct, with Cam_ReadJPEGFrame(…), and then sends out the packet.

```
unsigned char Cam_SendJPEGFrame(unsigned char  send, unsigned int id)
{
      unsigned char xret;
      unsigned char st[];
      unsigned int LOCALCOUNTER;

      LOCALCOUNTER = 4;
      xret = Cam_ReadJPEGFrame(id);
      if(!xret)   return 0;

      while(LOCALCOUNTER < JPEGDIM-2)
      {
        if( IMAGECOUNTER==(IMAGESIZE-1) )
        {
            if( LOCALCOUNTER==(JPEGDIM-3) )
                CAMSTOP=1;
            else
            {
                Cam_SWError(CAM_ERR_SW_ERROR4);
                  return 0;
            }
        }
        st[0] = FRAME[LOCALCOUNTER++];
        Send_by_UART1((unsigned int *)st,1);
            IMAGECOUNTER++;
      }

      return 1;




}
```

void Cam_Init(void)

This feature assigns default values to the main global variables to perform a new image stream.

```
void Cam_Init(void)
{
      JPEGDIM = 0;
      CAMSTATUS = CAMSTATUS & 0x0001;
      CAMCOUNTER = 0;
      CAMRBCOUNTER = 0;
      IMAGECOUNTER = 0;
      BCOUNTER = 0;
      TXCOUNTER=0;
      SEND=0;
      IMAGESIZE = 0;
      CAMSTOP = 0;
      NEWLIMIT = 0;
      return;
}
```

void Cam_Turnoff(void)

This feature closes the application disabling the UART2 and T89 peripherals.

```
void Cam_Turnoff(void)
{
    CloseUART2();
    Cam_CloseTimer89();
    return;
}
```

unsigned char Cam_GetStream(void)

This function initially calls the function Cam_Init () described above and then, reads the CAMCONFIG bits to activate the correct sequence according to the requests.

```
unsigned char Cam_GetStream(void)
{
         unsigned char xret = 0;
         unsigned int pictype, snaptype;

         Cam_Init();

         pictype = (CAMCONFIG>>PICTYPE_FLAG) & PICTYPE_NUM;
         if( pictype==DATATYPE_SNAPSHOT )
         {
             snaptype = (CAMCONFIG>>SNAPTYPE_FLAG) & SNAPTYPE_NUM;
             xret = Cam_GetPicture(snaptype);
         }
         else if(pictype==DATATYPE_PREVIEW)
                  xret = Cam_GetVideo(UNCOMPRESSED_VIDEO);
              else if(pictype==DATATYPE_JPEG_PREVIEW)
                       xret = Cam_GetVideo(COMPRESSED_VIDEO);
                   else    return 0;

             Cam_PowerOffCmd();
             if(xret) CAMSTATUS=0;
         return xret;
}
```

unsigned char Cam_GetPicture(unsigned char type)

This function calls the function Cam_UncompressePicture () or the function Cam_JpegPicture () in according to the value of the argument.

```
unsigned char Cam_GetPicture(unsigned char type)
{
         if( type == UNCOMPRESSED_PICTURE )
             return Cam_UncompressedPicture();
         else if(type == COMPRESSED_PICTURE)
                 return Cam_JpegPicture();
              else return 0;
}
```

unsigned char Cam_GetVideo(unsigned char type)

This function calls the function Cam_UncompresseVideo () or the function Cam_JpegVideo () in according to the value of the argument.

```
unsigned char Cam_GetVideo(unsigned char type)
{
         if( type == COMPRESSED_VIDEO )
             return Cam_JpegVideo();
         else if( type == UNCOMPRESSED_VIDEO )
                 return Cam_UncompressedVideo();
              else return 0;
}
```

unsigned char Cam_JpegPicture(void)

This function implements the JPEG image acquisition procedure as described in the figure reported at the end of paragraph 2.

```
unsigned char Cam_JpegPicture(void) {
         unsigned char xret, send;
         unsigned int ind=;
            BCOUNTER=-6; CTYPE=1;
         send = (CAMCONFIG>>SEND_FLAG)&SEND_NUM;
         xret = Cam_InitialCmd();
         if(!xret) return 0;
         xret = Cam_Wait(ACK_FLAG, CAM_CMD_INITIAL, 0x0000FFFF);
         if(!xret) return 0;
         xret = Cam_PackSizeCmd();
         if(!xret) return 0;
         xret =  Cam_Wait(ACK_FLAG, CAM_CMD_SETPACKETSIZE, 0x0000FFFF);
         if(!xret) return 0;
         xret = Cam_SnapshotCmd();
         if(!xret) return 0;
         xret =  Cam_Wait(ACK_FLAG, CAM_CMD_SNAPSHOT, 0x0000FFFF);
         if(!xret) return 0;
         xret = Cam_GetCmd();
         if(!xret) return 0;
         xret = Cam_Wait(ACK_FLAG, CAM_CMD_GETPICTURE, 0x0000FFFF);
         if(!xret) return 0;
         xret = Cam_Wait(DATA_FLAG, DATATYPE_SNAPSHOT, 0x0000FFFF);
         if(!xret) return 0;
         xret = Cam_AckCmd((unsigned long int)ind);
         if(!xret) return 0;
         while() {
             xret = Cam_Wait(IMAGE_FLAG, ind, 0x0000FFFF);
             if(!xret) return 0;
             xret = Cam_SendJPEGFrame(send,ind);
             if(!xret) return 0;
             if(CAMSTOP) break;
             ind++; xret = Cam_AckCmd( (ind<<8)+(ind>>8) );
             if(!xret) return 0;
         }
         if(xret) xret = Cam_AckCmd(0x0000F0F0);
      return xret;
}
```

unsigned char Cam_UncompressedPicture(void)

This function implements the uncompressed image acquisition procedure as described in the figure reported at the end of paragraph 2.

```
unsigned char Cam_UncompressedPicture(void)
{
        unsigned char xret, send;
        send = (CAMCONFIG>>SEND_FLAG)&SEND_NUM;
            CTYPE=0;

        xret = Cam_InitialCmd();
        if(!xret) return 0;

        xret = Cam_Wait(ACK_FLAG,  CAM_CMD_INITIAL, 0x0000FFFF);
        if(!xret) return 0;

        xret = Cam_SnapshotCmd();
        if(!xret) return 0;

        xret = Cam_Wait(ACK_FLAG,  CAM_CMD_SNAPSHOT, 0x0000FFFF);
        if(!xret) return 0;

        xret = Cam_GetCmd();
        if(!xret) return 0;

        xret = Cam_Wait(ACK_FLAG,  CAM_CMD_GETPICTURE, 0x0000FFFF);
        if(!xret) return 0;

        xret = Cam_Wait(DATA_FLAG, DATATYPE_SNAPSHOT, 0x0000FFFF);
        if(!xret) return 0;

        xret = Cam_Wait(IMAGE_FLAG, 0, 0x000FFFFF);
        if(!xret) return 0;

            xret = Cam_AckCmd(0x0A000000);
        if(!xret) return 0;


        return xret;
}
```

unsigned char Cam_JpegVideo(void)

This function implements the JPEG video acquisition procedure as described in the figure reported at the end of paragraph 2.

```
unsigned char Cam_JpegVideo(void)
{
        unsigned char xret, ind=0, k=; CTYPE=1;
        xret = Cam_InitialCmd();
        if(!xret) return 0;
        xret = Cam_Wait(ACK_FLAG, CAM_CMD_INITIAL, 0x0000FFFF);
        if(!xret) return 0;
        xret = Cam_PackSizeCmd();
        if(!xret) return 0;
        xret = Cam_Wait(ACK_FLAG, CAM_CMD_SETPACKETSIZE, 0x0000FFFF);
        if(!xret) return 0;
        start1=1;
        while(k<VIDEOSIZE) {
                while(!start1); start1=0;
                BCOUNTER=-6; ind=0;
                xret = Cam_GetCmd();
                if(!xret) return 0;
                xret = Cam_Wait(ACK_FLAG, CAM_CMD_GETPICTURE, 0x0000FFFF);
                if(!xret) return 0;
                xret = Cam_Wait(DATA_FLAG, DATATYPE_JPEG_VIDEO, 0x0000FFFF);
                if(!xret) return 0;
                while(BusyUART1());
                xret = Cam_AckCmd(ind);
                if(!xret) return 0;
                while() {
                      xret = Cam_Wait(IMAGE_FLAG, ind, 0x0000FFFF);
                      if(!xret) return 0;
                      xret = Cam_SendJPEGFrame(1,ind);
                      if(!xret) return 0; if(CAMSTOP) break;
                      ind++; xret = Cam_AckCmd( (ind<<8)+(ind>>8) );   if(!xret) return 0;
                }
                xret = Cam_AckCmd(0x0000F0F0); if(!xret) break;
                k++; Cam_Init();
        }
        return xret;
}
```

unsigned char Cam_UncompressedVideo(void)

This function implements the uncompressed video acquisition procedure as described in the figure reported at the end of paragraph 2.

```
unsigned char Cam_UncompressedVideo(void)
{
          unsigned char xret,ind=0;// k=0, ind=0;
              CTYPE=0;

          xret = Cam_InitialCmd();
          if(!xret) return 0;
          xret = Cam_Wait(ACK_FLAG, CAM_CMD_INITIAL, 0x0000FFFF);
          if(!xret) return 0;
          start1=1;
          while(ind<VIDEOSIZE)
          {
                    while(!start1);
                    start1=0;

              xret = Cam_GetCmd();
              if(!xret) break;
              xret = Cam_Wait(ACK_FLAG, CAM_CMD_GETPICTURE, 0x0000FFFF);
              if(!xret) break; //continue;

              xret = Cam_Wait(DATA_FLAG, DATATYPE_PREVIEW,  0x0000FFFF);
              if(!xret) break;

              xret = Cam_Wait(IMAGE_FLAG, 0, 0x000FFFFF);
              if(!xret) break;

              xret = Cam_AckCmd(0x0A000000);
              if(!xret) break;
                    Cam_Init();
                    ind++;
          }
          return xret;
}
```

unsigned char Cam_Wait(unsigned int numf, unsigned int id,unsigned longint stbTime)

This is certainly the most important function of the firmware. A cyclic timed wait function is realized with this function. This function is called every time the microcontroller is waiting for a command or an image. The function initially configures the T89. The first argument numf specifies the expected command type and the second argument id is intended to specify any additional information useful in the function Cam_Check (numf, id), for example the identifier of the command just sent to the camera. The implementation of the cycle controls the occurrence of a certain event.

Possible escapes from this cycle are:

- There is an error, ERROR non-zero byte.
- The T89 triggers an interrupt and resets the WAIT flag, so the maximum waiting time has expired and CAMSTOP is zero (the image transmiss is not yet finished).
- The received command is different than expected.
- Expected command is arrived.

```
unsigned char Cam_Wait(unsigned int numf, unsigned int id,unsigned long  int stbTime)
{
     unsigned char xret = 0, waitf;

     T8CONbits.TON = 0;        /* Disable Timer 8 */
       Cam_WriteTimer89(0);
       PR8 = stbTime;
     PR9 = stbTime>>16;
     T8CONbits.T32 = 1;

       CAMSTATUS = CAMSTATUS | (WAIT_NUM<<WAIT_FLAG);       /* Set timer flag */
       CAMSTATUS = CAMSTATUS & (~(READY_NUM<<READY_FLAG));     /* Clear Ready command flag */

     CAMSTATUS = CAMSTATUS | (0x01<<numf);     /* Set response command flag */
       T8CONbits.TON= 1;                         /* Turn on Timer89 */
```

```
    while( ((CAMSTATUS>>numf)&0x01) && (((CAMSTATUS>>WAIT_FLAG)&WAIT_NUM) || CAMSTOP) &&
(((CAMSTATUS>>ERROR_FLAG)&ERROR_NUM)==0) && (((CAMSTATUS>>READY_FLAG)&READY_NUM)==0) && Cam_SendPicture()  );

      T8CONbits.TON = 0;  /* Disable Timer 8 */
      waitf = ((CAMSTATUS>>WAIT_FLAG)&WAIT_NUM)==0;
      CAMSTATUS = CAMSTATUS & (~(READY_NUM<<READY_FLAG));      /* Clear Ready command flag */

    if( ((CAMSTATUS>>ERROR_FLAG)&ERROR_NUM) )
    {
          CAMSTATUS = CAMSTATUS & (~(WAIT_NUM<<WAIT_FLAG));       /* Clear timer flag */
          CAMSTATUS = CAMSTATUS & (~(0x01<<numf));                   /* Clear exp command flag */
        return 0;
    }
    if( waitf )
    {
          CAMSTATUS = CAMSTATUS & (~(WAIT_NUM<<WAIT_FLAG));       /* Clear timer flag */
          CAMSTATUS = CAMSTATUS & (~(0x01<<numf));                   /* Clear exp command flag */
        if(id != CAM_CMD_SYNC) Cam_SWError(CAM_ERR_SW_ERROR5);
        return 0;
    }
    if( numf==IMAGE_FLAG )
    {
          CAMSTATUS = 0x0001;
          if(!CAMSTOP) CAMSTATUS = CAMSTATUS | (IMAGE_NUM<<IMAGE_FLAG);      /* Set image flag */
        return 1;
    }
    if( ((CAMSTATUS>>numf) & 0x0001)== )
    {
        CAMSTATUS = CAMSTATUS & (~(ERROR_NUM<<ERROR_FLAG));      /* Clear ERROR flags */
          CAMSTATUS = CAMSTATUS & (~(WAIT_NUM<<WAIT_FLAG));          /* Clear timer flag */
        xret = Cam_Check(numf,id);
        return xret;
    }
    else
    {
          CAMSTATUS = CAMSTATUS & (~(WAIT_NUM<<WAIT_FLAG));       /* Clear timer flag */
          CAMSTATUS = CAMSTATUS & (~(0x01<<numf));                   /* Clear exp command flag */
          Cam_SWError(CAM_ERR_SW_ERROR7);
        return 0;
    }
    return 0;
}
```

unsigned char Cam_Check(unsigned int numf, unsigned int id)

This function checks the correctness of the received ACK command and also stores the image size value sent by the camera.

```
unsigned char Cam_Check(unsigned int numf, unsigned  int id)
{
     unsigned long int a,b,c;
     if(numf==ACK_FLAG)
     {
             if( CAMBUF[2] != (id&0x00FF) )
         {
              Cam_SWError(CAM_ERR_SW_ERROR8);
              return 0;
         }
     }
     if(numf==DATA_FLAG)
     {
        if( CAMBUF[2] != (id&0x00FF) )
        {
            Cam_SWError(CAM_ERR_SW_ERROR9);
            return 0;
        }

      a = (unsigned long int)(CAMBUF[3]);
        b = (unsigned long int)(CAMBUF[4]);
        c = (unsigned long int)(CAMBUF[5]);
        IMAGESIZE = a + (b<<8) + (c<<16);
     }
     return 1;
}
```

unsigned char Cam_ReadJPEGFrame(unsigned int ind)

This function checks the consistency of each received JPEG package. In particular the code analyzes the field ID, the length of the package and the final field called Verify Code.

```
unsigned char Cam_ReadJPEGFrame(unsigned int  ind)
{
    unsigned int id,size,sum=0,i,vcode,a,b;

    a = FRAME[0];
    b = FRAME[1];
    id = a + (b<<8);
    if(id!=ind)
    {
        Cam_SWError(CAM_ERR_SW_ERROR10);
        return 0;
    }
    a = FRAME[2];
    b = FRAME[3];
    size = a + (b<<8);
    if(size!= (JPEGDIM-6))
    {
        Cam_SWError(CAM_ERR_SW_ERROR11);
        return 0;
    }

    for(i=;i<size+4;i++)
    {
        sum += (FRAME[i]);
    }
    vcode = FRAME[JPEGDIM-2];
    if((sum & 0x00FF) != vcode)
    {
        Cam_SWError(CAM_ERR_SW_ERROR12);
        return 0;
    }
    return 1;
}
```

unsigned char Cam_Synchronization(float Fcy)

This function makes connection with the camera performing the synchronization procedure. Moreover, the function configures (making use of function Cam_RoundFloat (…)) the UART2 module, which manages the serial bus for the communication with the camera, and the T89 timer to implement the timed wait functions.

```
unsigned char Cam_Synchronization(float Fcy)
{
     unsigned char xret = 1, i=, camerr;
       unsigned int bdiv;
     float baudvalue;

       Cam_Init();
     CAMFCY = Fcy;
       CAMSTATUS = CAMSTATUS & (~(SYNCHRO_NUM<<SYNCHRO_FLAG));      /* Clear Syncronization flag */
     bdiv = (CAMCONFIG>>BAUDDIV_FLAG) & BAUDDIV_NUM;
     switch(bdiv)
     {
         case 0:
                bdiv = (unsigned int)((BR7200+1)*);     // 7200 bps
                break;
         case 1:
                bdiv = (unsigned int)((BR9600+1)*);     // 9600 bps (default)
                break;
         case 2:
                bdiv = (unsigned int)((BR14400+1)*);    // 14400 bps
                break;
         case 3:
                bdiv = (unsigned int)((BR19200+1)*);    // 19200 bps
                break;
         case 4:
                bdiv = (unsigned int)((BR28800+1)*);    // 28800 bps
                break;
         case 5:
                bdiv = (unsigned int)((BR38400+1)*);    // 38400 bps
                break;
         case 6:
```

```
                bdiv = (unsigned int)((BR57600+1)*);    // 57600 bps
                break;
        case 7:
                bdiv = (unsigned int)((BR115200+1)*); //   115200 bps
                break;
        default:
                return 0;
    }
    baudvalue = 14745600.0/((float)bdiv);
    baudvalue = Fcy/16.0/baudvalue-1.0;
    baudvalue = Cam_RoundFloat(baudvalue);
    UART2_Config((unsigned int)baudvalue);
      Cam_CloseTimer89();
      Cam_TIMER89Config(0);
    while()
    {
        i++;
            camerr = (CAMSTATUS>>ERROR_FLAG)&ERROR_NUM;
        if( (i>) || camerr )
        {
          xret=0;
          if(camerr==CAM_ERR_COMMAND_HEADER_ERROR)
                xret=1;
                  break;
             }
        Cam_Sleep(0x00000FFF);
        xret = Cam_SyncCmd();
        if( !xret ) continue;
        xret = Cam_Wait(ACK_FLAG, CAM_CMD_SYNC, 0x0000FFFF);   // (AA 0E 0D xx 00 00) ACK
        if( !xret ) continue;
        xret = Cam_Wait(SYNC_FLAG, 0, 0x0000FFFF);   // (AA 0D 00 00 00 00) SYNC
        if( !xret ) break;
        xret = Cam_AckCmd(0x0D000000);// send final ACK command
        if( !xret ) continue;
        break; // Syncronization is successfully...
    }
    if(xret) CAMSTATUS = 0x0001;
    else CAMSTATUS = CAMSTATUS & 0xFFFE;
    return xret;
}
```