

# RTP Project Report

Section ID: 035516 Real-Time Programming

## *CMOS-Camera based Motion Detection*



Wolfgang Paulus

6606 Daylily Drive  
Carlsbad, CA 92009

mailto: [wolf@paulus.com](mailto:wolf@paulus.com)



<b>1. PROJECT DESCRIPTION.....</b>	<b>3</b>
BACKGROUND TASK AND INTERRUPT-SERVICE-ROUTINES (PSEUDO CODE) .....	3
<b>2. HARDWARE DESIGN.....</b>	<b>4</b>
2.1. CMOS CAMERA (VV6301 SENSOR).....	4
2.2. 16-KEY KEYPAD .....	5
2.2.1. Key, ASCII Character, Behavior Mappings .....	7
2.3. HARDWARE BLOCK DIAGRAM .....	7
<b>3. SOFTWARE DESIGN.....</b>	<b>8</b>
3.1 SOFTWARE MODULS OVERVIEW .....	8
3.2. SYSTEM RESOURCES TABLE.....	8
3.3. SOFTWARE FLOW DIAGRAM.....	9
3.3.1 Timing.....	10
Timer-2 has to be disabled during the data download .....	10
3.4. DATA STRUCTURES .....	11
3.5. ESSENTIAL CODE EXPLAINED .....	11
3.5.1. KeypadDriver.C.....	11
3.5.2. Watchdog.C.....	13
3.5.3. Camera – Driver: camdriver.c .....	14
3.5.3.1. Protocol.....	14
3.5.4. “The big Loop” in main().....	16
3.6. SCREEN LOG .....	17
<b>4. TESTING .....</b>	<b>18</b>
4.1. TEST METHODOLOGY.....	18
4.2. TEST PROCEDURES (AFTER INTEGRATION PHASE) .....	18
<b>5. RESULTS.....</b>	<b>18</b>
<b>6. PROGRESS AND PROBLEMS.....</b>	<b>19</b>
6.1. TIME ESTIMATES     ESTIMATE D / USED .....	19
<b>7. PART LIST:.....</b>	<b>19</b>
<b>8. REFERENCES:.....</b>	<b>20</b>
<b>9. SOURCE LISTINGS.....</b>	<b>21</b>
<b>10. PRESENTATION SLIDES .....</b>	<b>22</b>



## 1. Project Description

This project demonstrates how to interface a low-resolution color camera with the 8031SDK to build a simple motion detection system. The basic task of this system is to let the camera take a picture, download the image data and compare it with a previously taken *shot*. If the two images differ considerably, execute an alert function, in any case repeat.

An inexpensive low-resolution CMOS camera, a 16 key keypad, and three LEDs are connected to the 8031SDK board. The LEDs inform about the current system status, the keypad is used to allow simple user interaction like start, stop, etc., and the CMOS camera is needed to capture raw image data.

An EDE1144 Keypad Encoder IC connects a 16-key keypad to the 8031SDK. Every time a key is pressed on the keypad, an **external interrupt** notifies about key-press events and ASCII-encoded information is read by an interrupt service routine.

The 8031's internal serial port is connected to the camera's RS-232 serial interface. The SDK controls the camera through this connection and also downloads raw image data into the SDK's SRAM.

The camera does not have any image processing capabilities and after a photo has been taken, about 1.2 Kbytes (for a *thumbnail* sized image) or about 20 Kbytes (for a full sized image) uncompressed, raw image sensor data, is downloaded into the SDK's SRAM memory. Since the camera's serial interface only supports a single transfer rate of 57,600 baud, optimized assembler routines are used to allow the SDK (running at about 12 MHz) to receive data and immediately transfer it into external memory.

The system's *motion detection* capabilities are based on quickly grabbing and comparing images. If consecutively captured images differ considerably, and alert function displays a message and blinks an LED. A software watchdog protects the system from the potential risk of an image grabbing process failure that could be triggered by insufficient illumination or by letting run out the 9V battery that powers the camera.

If image grabbing or transfer fail, the compare function doesn't get to run and an alarm could not be triggered. To cover this case, a **timer interrupt** based watchdog is used to call the alarm function if not being reset continually.

### Background Task and Interrupt-Service-Routines (pseudo code)

```
1. Let the camera take a picture
2. Download a "thumbnail" version (40 × 31) of the image into the SDK.
3. Compare the thumbnail with a previously taken one
4. If the difference is greater than a threshold {
    Execute an alarm function (to blink an LED i.e.)
}
```

```
Start, Pause or Stop the program based on Keypad inputs
```

```
Execute an alarm function, if the "watchdog countdown" reaches 0, (watchdog was not fed).
```



## 2. Hardware Design

### 2.1. CMOS Camera (VV6301 Sensor)

The camera hardware consists of the following major components:

- STMicroelectronics VV6301 CMS Sensor
- TEMIC 51 X3702/B (Intel MSC51 compatible) Micro Controller
- Winbond 128 Kbytes CMOS SRAM
- 2 TI 97E1L1M High Speed Counter

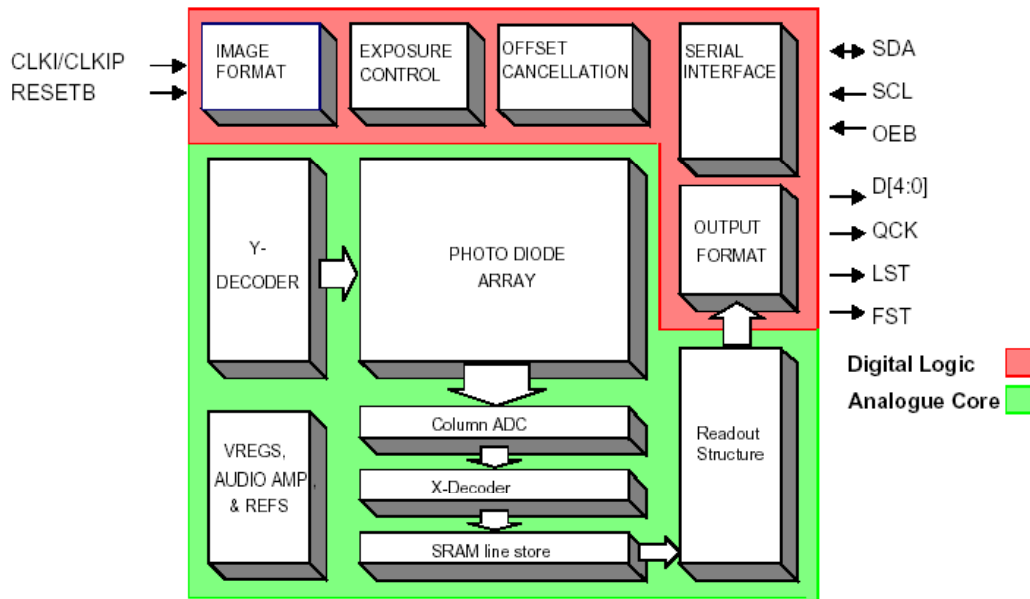


Figure 1: Image Sensor Block Diagram (Source [4])

The Camera's serial port is connected to the SDK's internal serial port.

The Camera-SDK serial connection speed is fixed at to 57,600 baud.



## 2.2. 16-Key Keypad

The following components are used to connect the keypad:

- EDE1144 Keypad Encoder IC (Jameco #171969) <http://www.elabinc.com/>
- Resistor Net 16PIN 330 OHM (Jameco #108572) <http://www.jameco.com>
- Resistor Net 10PIN 4.7K OHM (Jameco #24660)
- Crystal Oscillator 4MHz 50ppm 20pF (Jameco #14592)
- 2x8 Keypad (Jameco #196171)

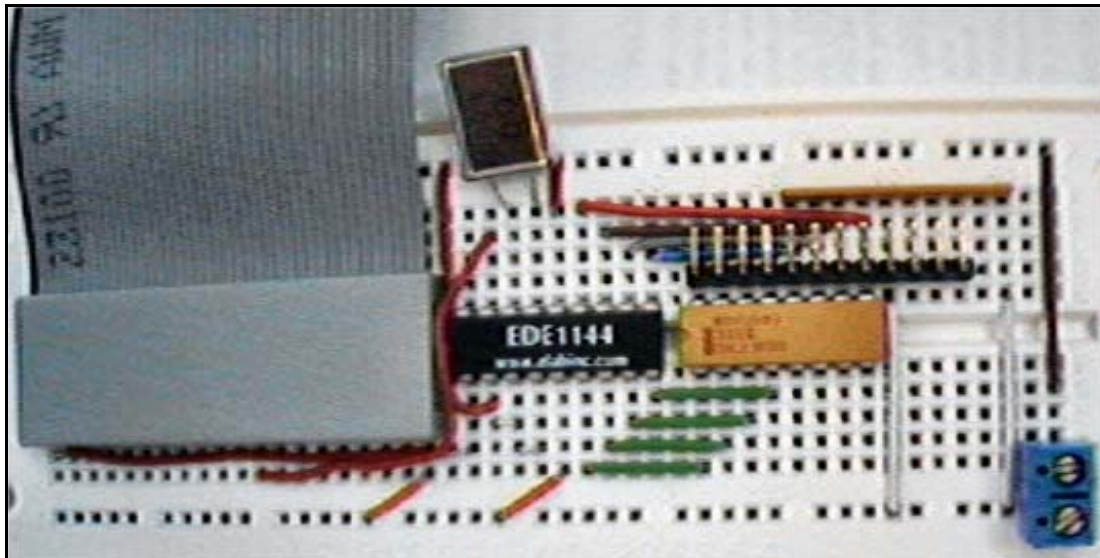


Figure 2: Keypad Connector, Encoder IC, Oscillator, and Resistors

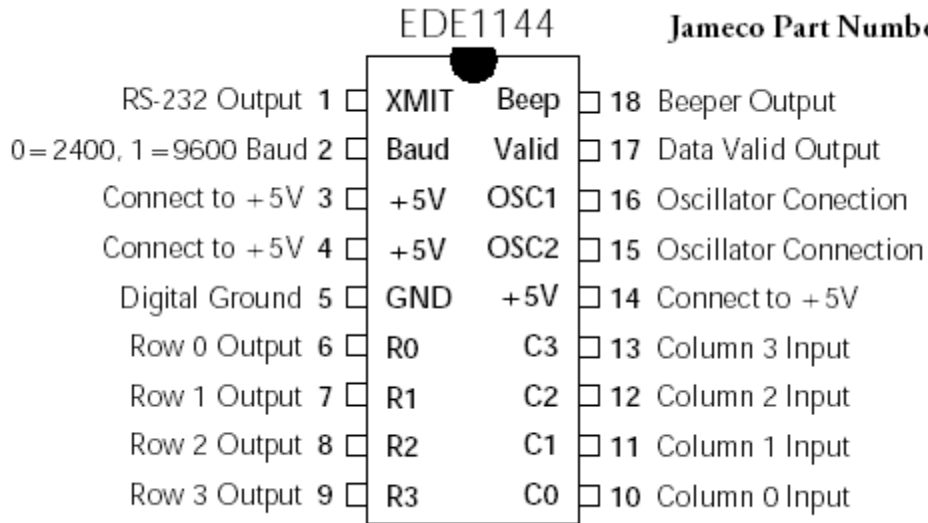


Figure 3: Keypad Encoder PINS (Source [6])

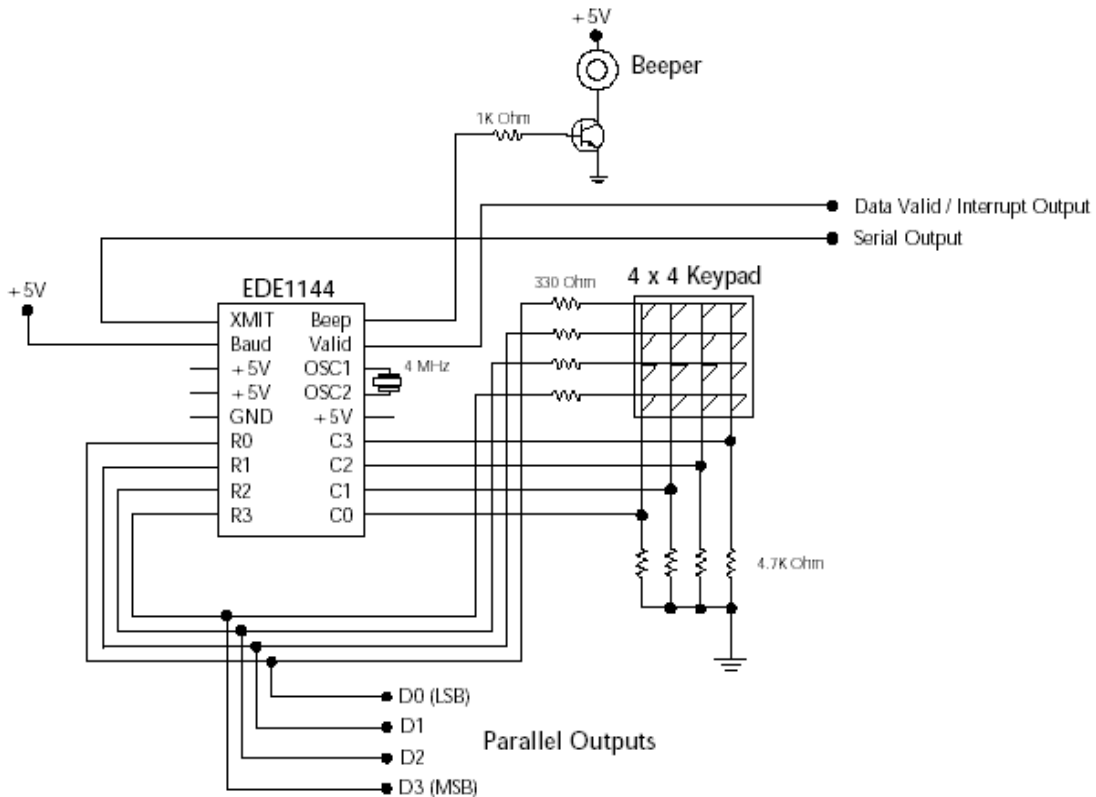


Figure 4: Schematic of keypad connection (Source [6])



### 2.2.1. Key, ASCII Character, Behavior Mappings

Keypad Label	EDE1144 Output	Lookup Table	Comments
1	0x30 (48d)	'1'	
2	0x31 (49d)	'2'	
3	0x32 (50d)	'3'	
↑	0x33 (51d)	'U'	Enable Watchdog
4	0x34 (52d)	'4'	
5	0x35 (53d)	'5'	
6	0x36 (54d)	'6'	
↓	0x37 (55d)	'D'	Disable Watchdog
7	0x38 (56d)	'7'	
8	0x39 (57d)	'8'	
9	0x41 (65d)	'9'	
2 <sup>nd</sup>	0x42 (66d)	'S'	
Clear	0x43 (67d)	'C'	Pause
0	0x44 (68d)	'0'	Exit
Help	0x45 (69d)	'?'	Help
Enter	0x46 (70d)	'E'	Start / Continue

Figure 5: Keypad / Encoder-Output /ASCII lookup table

### 2.3. Hardware Block Diagram

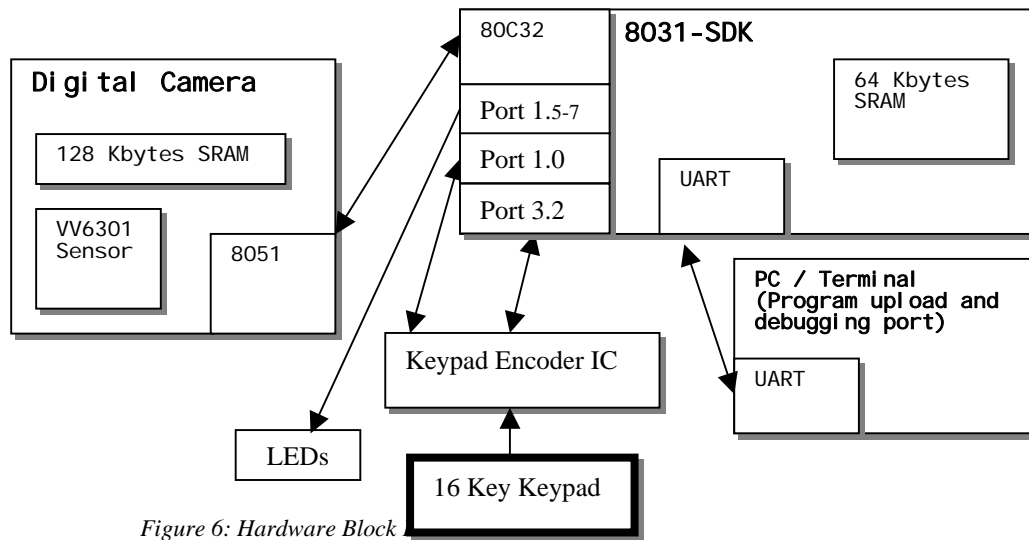


Figure 6: Hardware Block

The Camera's serial port is connected to the SDK's internal serial port, while the SDK's external UART connects to the PC / Terminal.

The Camera-SDK serial connection speed is set to 57,600 baud, the SDK-PC serial connection speed is set to 9,600 – 38,400 baud



### 3. Software Design

#### 3.1 Software Moduls Overview

To support re-use and better maintainability of the software, the following modularization was chosen:

- **Main.c** contains the main loop
- **CamDriver.c** contains the camera specific functions, allowing control of the camera.
- **ImgProc.c** contains the implementation of image comparison algorithms.
- **Watchdog.c** contains the implementation of the timer-based software watchdog.
- **Serialin57.asm, Serialout57.asm**, high-speed serial driver for the SDK's internal serial port.
- **Keypad.c** contains keypad specific functions

All modules are accompanied by \*.h files, containing function prototypes.

#### 3.2. System Resources Table

Resource	Comment	Function / Program	Module
Interrupt0	External Interrupt0	keypad_isr()	KeypadDriver.C
Interrupt1	Timer-0 Interrupt	----	----
Interrupt2	External Interrupt1	---- (8031 Monitor)	---
Interrupt3	Timer-1 Interrupt	----	----
Interrupt4	Serial Interrupt	----	----
Interrupt5	Timer-2 Interrupt	watchdog_isr()	Watchdog.C
Timer 0	Timer 0	sendserial (), readserial()	serialin57.asm, serialout57.asm
Timer 1	Timer 1	<ul style="list-style-type: none"> <li>• keypad_isr()</li> <li>• readserial()</li> <li>• alarm()</li> </ul>	<ul style="list-style-type: none"> <li>• KeypadDriver.C</li> <li>• serialin57.asm</li> <li>• main.c</li> </ul>
Timer 2	Timer 2	watchdog_isr()	Watchdog.C



### 3.3. Software Flow Diagram

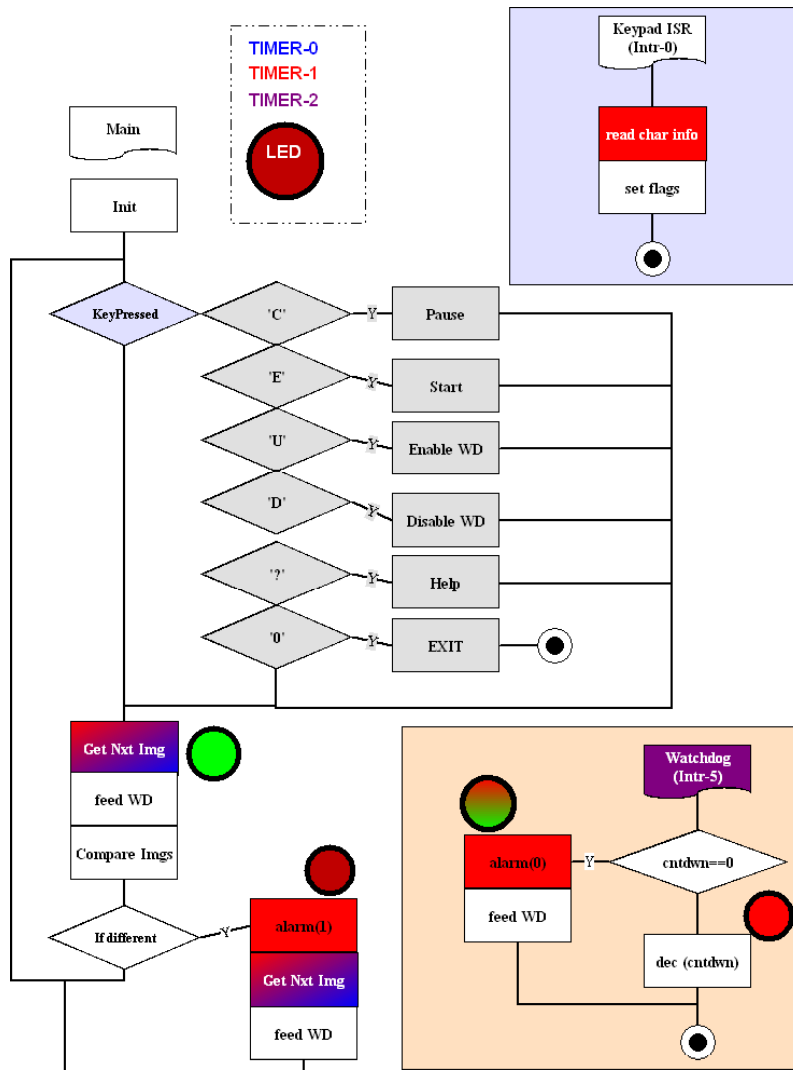


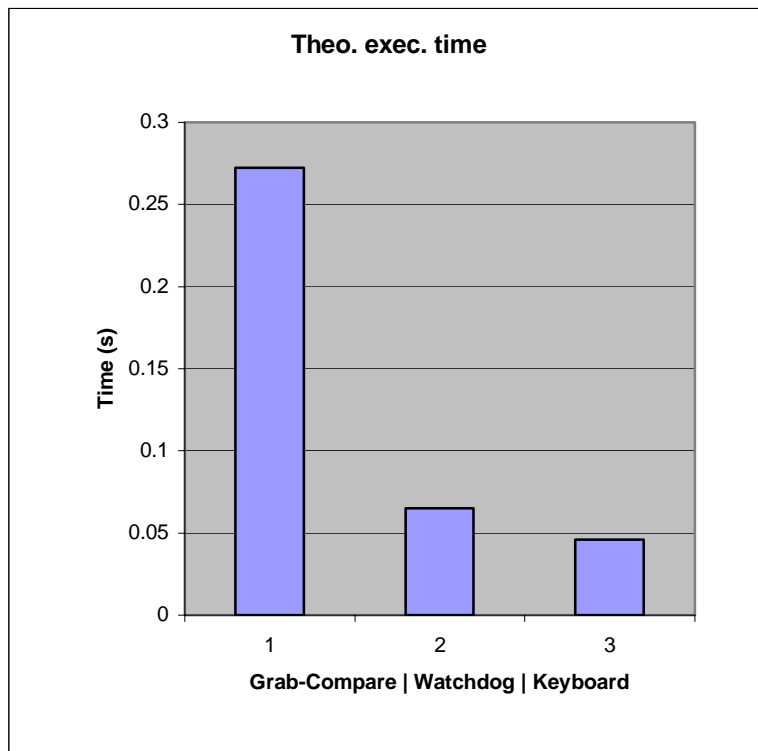
Figure 7: Software Flowchart



### 3.3.1 Timing

**Timer-2 has to be disabled during the data download**

Grab-Compare Freq.	sec	cycles	Watchdog	sec	cycles	Keypad	sec	cycles
<b>Get Nxt Img</b>						<b>Event</b>		
sendSerial(4)	0.0008	800						
readSerial(4)	0.0008	800						
sendSerial(4)	0.0008	800						
readSerial(1240)	0.2	198400						
	0.2024	200800	Triggered	0.065	<b>65536</b>			
<b>feed WD</b>						Pause	0.045	45000
<b>Cmp Img</b>	0.07	70000				Bit bang	0.0009	900
Theoretical limit	0.2724	270800		0.065	<b>65536</b>		0.0459	45900
Actual behavior	0.6							



This diagram shows that the time needed to grab and download an image is much longer than the time between interrupts triggering the Watchdog. The download of a complete image only works if uninterrupted. There is no sync process (handshaking etc.) available and the camera continues sending data even if the SDK stops receiving. Therefore, the Watchdog timer needs to be stopped during the download process.



### 3.4. Data structures

Besides standard C data types, only the **ImgPTR** type is used. This type is declared in `camdriver.h` and used to identify pointers to images.

```
typedef unsigned char volatile xdata* ImgPTR;
```

The program's `main()` function defines the `rawimg` variable, reserving `IMGSIZE+HDRSIZE` number of bytes.

```
unsigned char xdata rawimg[ IMGSIZE + HDRSIZE ];
```

While this area is used to store two thumbnail size images, it is setup this way to also accommodate one full size image if needed.

```
ImgPTR buffer[] = {&rawimg, &rawimg + TNLSIZE + HDRSIZE};
```

This array stores two pointers. Both pointers point to memory locations in `xdata` and both locations are able to store a thumbnail size image (including a header) without corrupting another.

### 3.5. Essential Code Explained

#### 3.5.1. KeypadDriver.C

The keypad driver implementation file contains the `initKeypad()` function as well as the ISR responding to the External Interrupt 0, (Interrupt 0).

```
void initKeypad (void);
```

The `initKeypad` function sets the Interrupt-0 latch high and sets it into transition (edge sensitive) mode. The Keypad-Encoder IC, for 50mS pulls the pin low; therefore the edge sensitive mode has to be used.

```
static void keypad_isr ( void ) interrupt 0 using 1
```

The ISR doesn't use any other shared variables but in the header file declared. Moreover, no other functions are getting called in the ISR. It is therefore safe to let the ISR run with its own set of R-registers (using 1).

After an interrupt is captured by this ISR, Timer-1 is used to *bit-bang* Port1-Pin-0 at 9600 baud, until the data byte, encoding the key that was pressed is received.

The `keypad_isr` function sets the Timer-1 into auto-reload mode, using a reload value of **0xA0**.

The Encoder-IC sends data at a baud rate of 9600 baud. Therefore, Timer-1 needs to be setup to overflow every 1/9600 sec. Since the SDK increments the timer register every 12/11059200 sec, the reload value is getting computed like this:



$$(256 - x) * 12 / 11059200 = 1 / 9600 \Leftrightarrow$$

$$256 - x = 11059200 / (12 * 9600) \Leftrightarrow$$

$$256 - x = 96$$

$$160 = x = 0xA0$$

Care has to be taken, however, since other functions also use Timer-1. Three globally available bit variables are used to flag the recent execution of this ISR:

```
keyEvent1 = keyEvent2 = keyEvent3 = 1;
```

*keyEvent1* is used inside the *main()* function to find out if a new user input occurred.

*keyEvent2* is used inside *serialin57.asm* to stop the current downloading since it was interrupted and data was definitely lost.

*keyEvent3* is used inside the *alarm()* function, which uses Timer-1 to blink LEDs.

**For debugging and demonstration purposes, an LED is switch on while this ISR executes.**

```
static void keypad_isr (void) interrupt 0 using 1 {
    unsigned char i, buf = 0;
    DBG_LED = 1; // switch LED on while in ISR

    keyPressed = ERR_KEY; // reasonable value if things go wrong from here
    keyEvent1 = keyEvent2 = keyEvent3 = 1; // to notify others that this ISR recently ran
    kbdta = 1; // set keyboard data receiver latch high

    ET1 = TR1 = TF1 = 0; // Timer-1: Disable Interrupt, Stop, clear flag
    TMOD&= 0x0F; // clear Timer-1 mode bits
    TMOD|= 0x10; // set Timer-1 into 16-bit mode
    TH1 = TL1 = 0; // set to 0, works now like an 16-bit auto reload timer
    TR1 = 1; // start the timeout timer
    while (kbdta && !TF1); // wait for startbit or timeout
    TR1 = 0; // stop Timer-1
    if (!TF1) { // Startbit was received
        TMOD&= 0x0F; // clear Timer-1 mode bits
        TMOD|= 0x20; // set Timer-1 into auto re-load mode
        TH1 = TL1 = 0xA0; // = 256 - (11.059MHz / (12 * 9600 BAUD))
        TR1 = 1; // start the timer and start bit banging
        i = 8;
        while (0<i--) { // loop 8 times to receive a full byte
            while (!TF1); // wait until its time to look for the next bit
            buf>>=1; // shift receiving bits into the buffer ..
            if (kbdta) { // .. starting left
                buf|=0x80; // set current bit.
            }
            TF1= 0; // clear sticky bit
        }
        if ((47<buf) && (buf<71)) { // check for valid data range
            keyPressed=keys[buf-48]; // lookup return value.
        }
    }
    TR1 = TF1 = 0; // stop Timer-1 and clear Timer-1 flag
    DBG_LED = 0; // switch LED off again
}
```



### 3.5.2. Watchdog.C

Watchdog.c implements the ISR responding to the Timer-2 Interrupt, (Interrupt 5) an initialization function and the feedWatchdog() function.

```
void enableWatchdog(unsigned char s);
```

This function initializes for the use of interrupt 5 and also programs Timer-2. Timer 2 is used as a 16-bit auto-reload timer. The reload value is set to 0, which generates an overflow about every 1/14 sec.

Timer calculation:

$$(65536 * 12) / (11.0592 * 1e6) = 0.07111 \text{ and } 1 / 0.07111 = 14.063$$

Therefore, the provided parameter value is multiplied by 14, to set the proper countdown value. Caused by the chosen variable type (unsigned char), the watchdog cannot be setup to wait longer than 18 sec.: (256 > 14\*s)

```
void feedWatchdog( void );
```

This function simply resets Timer-2 and the internal countdown variable.

```
void feedWatchdog( void ) {  
    if ( _testbit_( ET2 ) ) { // disable Intr and feed the dog, in case it is awake.  
        TR2 = 0; // stop Timer-2  
        TH2 = RCAP2H; // reset the Timer-2  
        TL2 = RCAP2L;  
        cntDwn= rstVal; // reset countdown  
        ET2 = 1; // enable Timer-2 Interrupt again  
        TR2 = 1; // re-start Timer-2  
    }  
}
```

```
static void watchdog_isr( void ) interrupt 5;
```

This ISR calls the *disableWatchdog()* function, which is publicly available, and also the externally defined *alarm()* function. Therefore, this ISR uses the default R-Register set.

The watchdog ISR only calls the alarm function if the module-static countdown counter reaches 0. However, **for debugging and demonstration purposes, an LED is toggled every time the ISR executes.**

To identify the watchdog to this function, *alarm()* is call with id=0. After the *alarm()* function was executed, the watchdog resets itself by calling the *feedWatchdog()* function

```
static void watchdog_isr( void ) interrupt 5 {  
    if (TF2) { // check if triggered by timer overflow  
        TF2= 0; // clear flag  
        led( cntDwn&1 );  
        if ( 0 == cntDwn-- ) { // countdown, only trigger alarm when 0 is hit  
            alarm( 0 ); // execute external alarm routine;  
            feedWatchdog(); // reset the countdown and the timer  
        }  
    } else {  
        EXF2= 0; // should never have happened, clear T2.Ext. flag anyway  
    }  
}
```



### 3.5.3. Camera – Driver: camdriver.c

#### 3.5.3.1. Protocol

All communications with the camera has to be initiated by the connected 80C32 controller, i.e., the controller is always the master and the camera always the slave.

All messages, both to and from the controller, are sent as packets starting with STX and terminating with ETX. [1]

**A command from the controller has the following format:**

***STX Command Data 1 Data 2 ... Data NETX***

**STX** Standard ASCII character (0x02).

**Command** A defined command byte, typically an **uppercase** character.

**Data n** Data for specific command. All commands have at least 1 byte of data (which is set zero if not used), some have more, the actually number is implicitly defined by the command byte.

**ETX** Standard ASCII character (0x03).

On receipt of a command from the controller, the camera always returns an acknowledgment. This will be either **ACK (0x06)** or **NAK (0x15)**. An ACK is returned if the command was successfully received and recognized (it does not indicate whether it was executed successfully, that is done by a response packet). A NAK indicates the command was not received successfully or was unrecognized. If no ACK or NAK is received in response to a command, the communication are assumed to be faulty or the camera disconnected.

**The response has the following format:**

***STX Response Data 1 Data 2 ... Data NETX***

**STX** Standard ASCII character (0x02).

**Response** A defined response byte, usually a **lowercase** character.

**Data n** Data for specific response. All responses have at least 1 byte of data (which is set zero if not used), some have more, the actually number is implicitly defined by the response byte.

**ETX** Standard ASCII character (0x03).

#### 3.5.3.1.1. Command: Reset Image Counter:

***STX CB\_RESET\_COUNTER (A) Index ETX***

The required image index must be in range 0 to MAX\_IMAGES or expect erratic results.



Response:

```
STX  RB_RESET_COUNTER (a)  Error Code  ETX
```

Error Code is always zero. If the camera is busy the RB\_CAMERA\_BUSY response is returned with the appropriate busy code.

### 3.5.3.1.2. Command: Grab an Image

```
STX  CB_GRAB_IMAGE (G)  Control  ETX
```

The lower nibble of *Control* specifies the self-timer delay (0-15). A delay of zero means the grab is to be executed immediately. The upper nibble controls the response to bad lighting conditions. Bits 4-6 specify the number of retries if the exposure is bad, ie, 0, 16, 32 ...112. Bit 7 is reserved with the intention of implementing a “grab anyway” option if needed.

Response:

```
STX  RB_GRAB_IMAGE (g)  Error Code  ETX
```

Error Code will always be set to zero. Use the grab result command to find out the status of the grab.

### 3.5.3.1.3. Command: Download Raw Image Sensor Data into the SDK

```
STX  CB_UPLOAD_IMAGE (U)  0  ETX
```

The argument is always zero.

Response:

```
STX  RB_UPLOAD_IMAGE (u)  N1 N2 N3 N4 D1 D2 ... Dn ETX
```

The complete image is returned as a stream of data bytes, including black lines, visible lines and status bytes. The total number of data bytes may be determined from the leading 4 bytes, N1 - N4, as follows:

$$N_{tot} = N1 \times (N2 + N3) + N4$$

**N1** is the number of columns in the image (164)

**N2** is the number of black lines (2)

**N3** is the number of visible lines (124)

**N4** is the number of status bytes (16)

$N_{tot}$  is the number of data bytes D1 - Dn only, it does not include the 6 leading header bytes or the final ETX. There is no explicit error reporting within the response message but the image status bytes may contain useful information.



### 3.5.4. "The big Loop" in main()

```
while( running ) {
    if ( _testbit_ ( keyEvent1 ) ) {
        switch (keyPressed) {
            case 'C' : // Stop
                disableWatchdog();
                active = 0;
                print( MSG_INACTIVE );
                break;

            case 'E' : // Enter Motion Detection
                print( MSG_ACTIVE );
                enableWatchdog( WD_TIMEOUT );
                getNextThumbnail( buffer );
                getNextThumbnail( buffer );
                active = 1;
                break;

            case 'U' : // Enable WDog
                enableWatchdog( WD_TIMEOUT );
                break;

            case 'D' : // Disable WDog
                disableWatchdog();
                break;

            case '?' : // Help
                help();
                break;

            case '0' : // Exit
                disableWatchdog();
                print( MSG_EXIT );
                running = active = 0;
                break;

            default : // Handle unused keys
                print("_");
                break;
        } // end switch
    } // end_if keyEvent

    if (active) { // get the next image
        if (getNextThumbnail( buffer )) {
            feedWatchdog(); // feed the dog, comparing takes a while
            if (cmpImages(buffer[0], buffer[1], TNLSIZE + HDRSIZE)) {
                alarm( 1 ); // call alarm if images are different
                getNextThumbnail( buffer );
                feedWatchdog();
            }
        }
    }
} // end_while running
```





## 4. Testing

### 4.1. Test Methodology

Before the whole system was tested, each component (ImageProc, Watchdog, Keypad, and CamDriver with serial57) was tested independently. main() routines were written to test only the functionality provided by a single module. (As an example, see main() functions ....). In a 2<sup>nd</sup> step, modules were integrated in pairs, testing them working together. This test was performed for the ImageProcessor - CamDriver pair as well as for the two ISRs, Watchdog and KeypadDriver working together.

Only after those test were successfully completed, all components were put together.

In a 3<sup>rd</sup> step, interrupts were forced to occur when the system was executing crucial routines.

### 4.2. Test Procedures (after integration phase)

- 1.) Perform reasonable movements in front of the camera and check result. (LED should go on)
- 2.) Unplug the camera from the system at various states, (LED should come on after some time has passed)
- 3.) Re-connect camera (LED should go off)
- 4.) Again, perform movements in front of the camera (LED should go on)
- 5.) Remove illumination but don't move (LED should still go on)
- 6.) Make key entries while in capture and loading routines
- 7.) Make key entries while an alarm is performed

## 5. Results

Grabbing an image and transferring it into the SDK needs some time, no matter how fast and optimized the serialin57.asm code may be. Grabbing, downloading and comparing takes close to 1 second and therefore, the motion detection system is not able to detect very fast motions, (i.e. in and out of the camera's view port in less than a second.) On the other hand, the system is not able to recognize incredibly slow motions either. This is caused by the fact that the pixel values of the same pixel location in two consecutive exposures are not equal and may differ up to **10%**, which is well in the expected range of a sub 20\$ camera. Therefore a slight difference has to be considered as NO motion. While at pixel level, a change of **10%** or more qualifies for a motion, looking at the whole image, only **3%** of all pixels have to be considered different to trigger a motion alert. These thresholds are obviously arbitrary and have to be adjusted based on the environment the system is used in.

Extensive tests have shown that reasonably fast movements in front of the camera are getting detected consistently.

Unplugging and re-connecting the camera works well – the system recovers reasonably well after reconnecting the camera.

Still after the camera is turned on, the first 2 or three exposures seem to differ considerably, almost as if the camera needs some kind of warm up phase. This behavior leads to an alarm being issued, soon after the camera is turned on again.



Since the camera is not equipped with an IR filter, illumination is a factor and has to be considered for usefulness of this implementation of a Motion-Detection-System. However, this system has the potential to be used for Motion-Logging, which standard MDS aren't capable of. After a motion has been detected based on thumbnail size images, the camera still hold the full image information, which could be download decoded and send to a connected file mass data storage. That this is very much possible was shown during my RTP-II project, which performed a BAYER-Decoding, UUEncoding of full sized (20Kbyte) images.

Still, illumination requirements and the low grabbing rate of about 58 image per minute are definitely drawbacks.

## 6. Progress and Problems

The implementation of the image comparer function actually went much faster than I had anticipated. However, the underlying assembler code, responsible of receiving the raw image data and transferring it into XDATA needed some improvement. I really wanted the receiver to have some kind of timeout that would make it return and recover gracefully, if the camera would suddenly fail. While I used a simple timer for this job, it was still difficult, because the hard 57,600-baud requirement does not leave enough processing cycles to care about timeouts.

The implementation of the interrupt functions was about as bad as expected. The one thing that turned this project into *hard work* was the usage of Timer-1 almost everywhere. Timer-1 is used to protect all bit-banging routines from endlessly waiting, caused by data producer error. It is also used to make LED's blink etc.

Finally, after introducing a flag for every Timer-1 user, things fell into place. The keypad ISR, which is a high priority uninterruptible function (and also a Timer-1 user), sets all these flags to notify every other Timer-1 user of a potential Timer-1 corruption.

6.1. Time estimates	Estimated	/	Used
Implement test image comparer function	2 h		3 h
Find suitable thresholds (also testing)	1 h		1 h
Implement and test Keypad-Driver	2 h		3 h
Implement and test Watchdog	4 h		2 h
Integrate all modules	6 h		5 h
Test System, final adjustments	3 h		2 h
Write draft version of the Project Report	4 h		8 h
Write final version of the Project Report (incl. PP slides)	4 h		6 h
<i>(I guess, I spent too much time on the documentation ...)</i>	26 h		30h

## 7. Part List:

- EDE1144 Keypad Encoder IC (Jameco #171969) <http://www.elabinc.com/>
- Resistor Net 16PIN 330 OHM (Jameco #108572) <http://www.jameco.com>
- Resistor Net 10PIN 4.7K OHM (Jameco #24660)
- Crystal Oscillator 4MHz 50ppm 20pF (Jameco #14592)
- 2x8 Keypad (Jameco #196171)
- VV6301 based CMOS Camera (i.e. NickClick )  
<http://shop.store.yahoo.com/direct2u-software/2172.html> \$13.99 (7/22/2002)



## 8. References:

- [1] "Company Communication" "EVK LoRes Serial Communications Protocol", Vision, 1998, RS232 command protocol specs, Jan. 1999, <http://webcam.sourceforge.net/barbie/scicomms.doc>
- [2] Jean-Pierre Dubé, "Java Tip 60: Saving bitmap files in Java", Java World Magazine. Sep 01 1998 (online version)
- [3] "Low Resolution Digital CMOS Image Sensor, VISION VV6300", VSLI Vision Limited 1998
- [4] "Low Resolution Digital CMOS Image Sensor, VV5301 & VV6301", STMicroelectronics, May 2001, <http://www.vvl.co.uk/>, [http://www.vvl.co.uk/products/image\\_sensors/500/7264197A.PDF](http://www.vvl.co.uk/products/image_sensors/500/7264197A.PDF)
- [5] "A Study of Spatial Color Interpolation Algorithms for Single-Detector Digital Cameras", Winter, 1999, Ting Chen, Information System Laboratory, Department of Electrical Engineering, Stanford University <http://ise0.stanford.edu/~tingchen/main.htm>
- [6] "EDE1144 Keypad Encoder IC", E-Lab Digital Engineering, Inc. Carefree Industrial Park, 1600 N. State Rte. 291 Hwy. Ste. 330 Independence, MO 64052-0436 <http://www.elabinc.com/ede1144.pdf>



## 9. Source Listings



## 10. Presentation Slides