

# ECP-1 Project

**Section ID:** 34270 Embedded Controller Programming

*Connect an 8x2 Keypad to the  
8031-SDK*

Wolfgang Paulus

6606 Daylily Dr  
Carlsbad, CA 92009

mailto: wolf@paulus.com

Student ID:

## Table of Contents

<b>Abstract</b> .....	<b>3</b>
<b>Hardware Design</b> .....	<b>3</b>
<b>Block Diagram</b> .....	<b>4</b>
<b>Software Design</b> .....	<b>4</b>
<i>main()</i> .....	4
<i>writeChar()</i> .....	4
<i>getBit()</i> .....	5
<i>getKeypadEvent()</i> .....	5
<i>ISR()</i> .....	5
<b>Software Flow Diagram</b> .....	<b>6</b>
<b>Analysis and Conclusion</b> .....	<b>6</b>
Testing and special results .....	7
Conclusion .....	7
<b>Part List</b> .....	<b>8</b>
<b>Code Listing 1: Test Keypad and Interrupt Line</b> .....	<b>8</b>
<b>Listing 2: Test data transfer from Encoder IC to Port Pin P1.0</b> .....	<b>9</b>
<b>Listing 3: Final (Interrupt driven) Version of the Software</b> .....	<b>11</b>

## Abstract

In this student project a 16-key keypad is getting connected to the HTE-8031SDK. While there are many ways to accomplish this task, the here presented combined hardware/software approach makes use of an EDE1144 Keypad Encoder IC.

Every time a key is getting pressed on the keypad, ASCII-encoded information is sent to a terminal or LCD connected to the SDK's internal serial port. To avoid constant port polling, an external interrupt notifies about key-press events.

While the whole program easily fits on two pages, still most of the 8051 relevant software techniques learned in the ECP-1 class have been applied. The software written for this project uses an Interrupt Service Routine to handle keypad events, a timer is used to and poll the state of a port pin, and another timer is used in combination with the internal serial port to send data to a connected terminal. A lookup table residing in external memory is used to convert the data received from the keypad-encoder into ASCII and terminal control character data.

Using a keypad encoder requires only one port to get the serialized keypad data into the microcontroller, compared to 8, if a decoder were not present. Moreover, the decoder IC makes it very easy to make use of the MC's external interrupts. On the other hand, the keypad encoder costs about \$7 when bought in single units, which makes it more expensive than the microcontroller used for this project.

## Hardware Design

An EDE1144 keypad encoder IC interfaces a 16-key (2 rows 8 columns) keypad to the SDK's microcontroller. The keypad encoder IC is used to reduce the number of required I/O pins and to handle contact debouncing.

Connecting a 16-key keypad directly to the microcontroller would require at least 8 port pins on the microcontroller. The use of the hardware encoder on the other hand requires only a single port.

The microcontroller receives serial data from the EDE1144 by connecting the EDE's XMIT line to the MC's pin P1.0. Moreover, the EDE's Data-Valid signal (pin 17) is connected to the MC's External Interrupt pin P3.2.

While the use of an interrupt line is not required, it certainly makes the software run more efficiently. Alternatively, the software could poll pin P1.0 constantly for the arrival of a new startbit (Listing 2 shows an implementation of this approach).

Because the keypad encoder needs to send the data serialized through one line, a crystal oscillator has to be connected to the encoder IC. Resistors have to be present to prevent a short circuit (during a scan) in case more than one key is being pressed. 4.7K-Ohm resistors should prevent inputs from floating or oscillating.

## Block Diagram

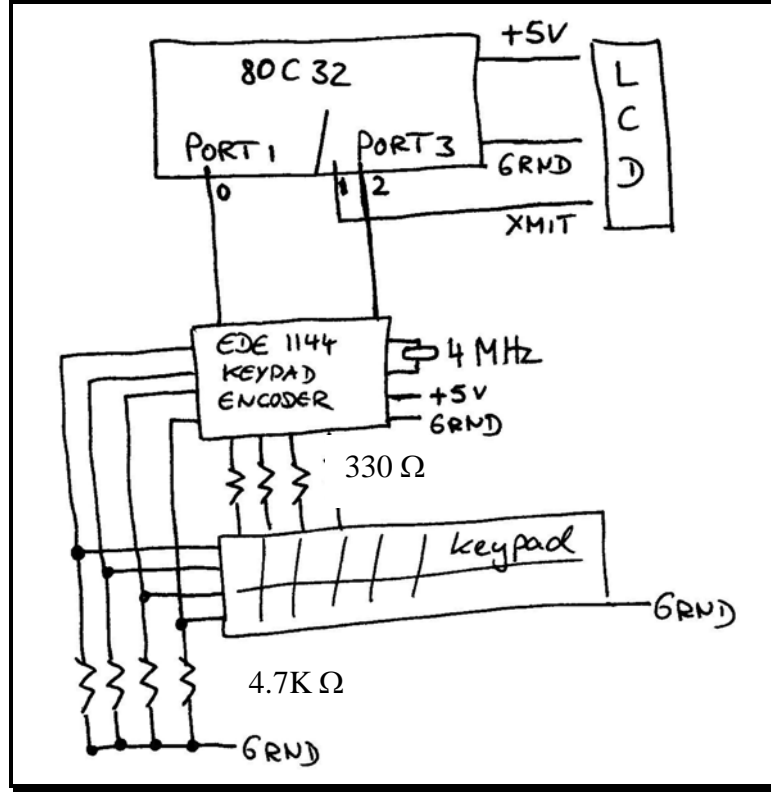


Figure 1 – Hardware Block Diagram

## Software Design

Using a keypad encoder IC substantially reduces the amount of work that needs to be done through software. Contact debouncing problems are sufficiently handled by the hardware and the software only needs to care about one or two port pins. Therefore, the program written for this project is fairly small but still applies most of the 8051 relevant software techniques like port polling, internal serial port processing, external memory access, and timers.

### **main()**

To allow a *main()* program to perform efficiently without constant worrying about probable key-presses, an interrupt service routine got deployed to handle keypad events. The main program sends ASCII encoded dots to the terminal by calling the *writeChar()* subroutine. Only when the down-arrow key is pressed will the program stop and exits back to the SDK-Monitor.

### **writeChar()**

The *writeChar()* subroutine sends the current content of the A register through the MC's internal serial port to. The routine waits for the XMIT to clear before sending the next byte. (Check the inline code documentation for details about the timer value calculations.)

**getBit()**

The *getBit()* subroutine is a timer controlled port listener, copying a port pin P1.0 into the *PSW's Carry* bit.

**getKeypadEvent()**

The *getKeypadEvent()* subroutine implements a manual serial reader. After a startbit is received, a timer is programmed and started in mode 2 (auto-reload).

8 bits are captured (using the above mentioned *getBit()* routine) and shifted into the A register. Before returning it to the caller, the value in the Accumulator is converted into ASCII data using a lookup table stored in external (code-) memory.

Keypad Label	EDE1144 Output	Lookup Table	Comments
1	0x30	#'1'	
2	0x31	#'2'	
3	0x32	#'3'	
↑	0x33	#'U'	<i>Display 'U' (up)</i>
4	0x34	#'4'	
5	0x35	#'5'	
6	0x36	#'6'	
↓	0x37	#'D'	<i>Display 'D' &amp; Exit main()</i>
7	0x38	#'7'	
8	0x39	#'8'	
9	0x41	#'9'	
2 <sup>nd</sup>	0x42	0x01	<i>LCD-Cursor Home</i>
Clear	0x43	0x0C	<i>LCD-Clear Display</i>
0	0x44	#'0'	
Help	0x45	#'?'	
Enter	0x46	0x0D	<i>LCD-Carriage Return</i>

Figure 2- Keypad / Encoder-Output /ASCII lookup table

**ISR()**

The interrupt service routine bound to the ISR vector for the external interrupt 0 pushes all registers that might be modified during the execution of the ISR on to the stack. It then calls the *getKeypadEvent()* subroutine to capture the data sent by the keypad encoder IC and finally writes out the converted to the terminal. Before the interrupted program continues, the previously stacked register values are restored.

The ISR makes the most currently pressed key available to other parts of the program by storing the key code value in the R0 register.

### Software Flow Diagram

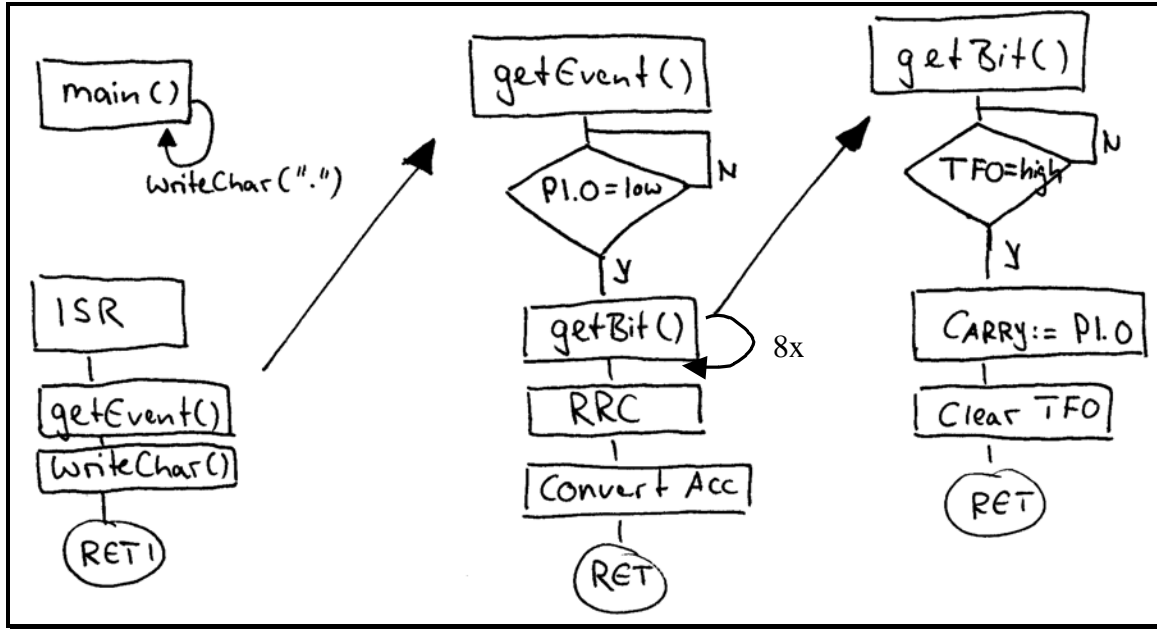


Figure 3- Software Flowchart

### Analysis and Conclusion

After the hardware was assembled, a 1<sup>st</sup> version of the software was written to test the interrupt handling. *Listing 1* shows this early version of the software. The program responds to an external interrupt, by simply sending “\*” to the connected terminal. The encoder’s data line was not polled at this stage.

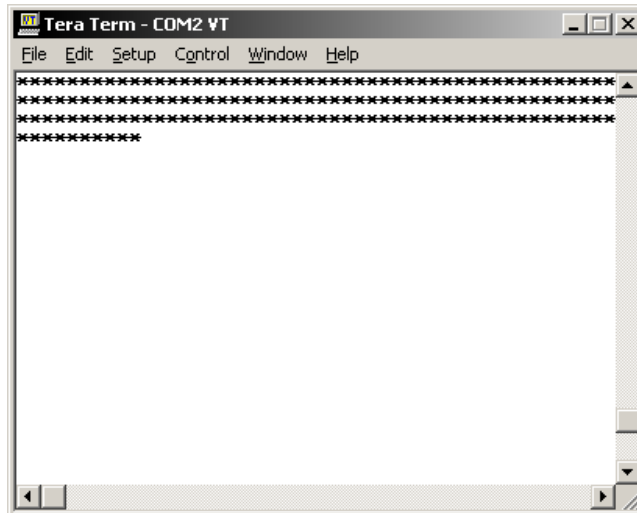


Figure 4- Output with early version of the software

The next step of the software didn’t use interrupts but constantly polled port pin P1.0 for a startbit. When a startbit was received, a timer got started and a data byte was read from the port pin. *Listing 2* shows this version of the program.



## 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)
- 16x2 LCD Crystalfontz <http://www.crystalfontz.com>

## Code Listing 1: Test Keypad and Interrupt Line

```

-----
ECP-1 Project
Connect a 8x2 keyad to the 8032-SDK
05/01/2002

Wolf Paulus (wolf@paulus.com)
6606 Daylily Drive
Carlsbad, CA-92009
-----
Test keypad with external interrupts
-----
CODE_BASE equ 4000h
BAUD_CONST equ 0FDh
-----
; On the HTE 8031SKD user programs incl. interrupt vectors,
; must be relocated and assembled starting at 4000h.
org CODE_BASE ; sets base address for program start
JMP main ; jumps around ISR to start main prg.
-----
org CODE_BASE + 3h ; ISR vector for External Interrupt 0
AJMP ISR_IE0 ; jumps to the ISR
-----
main:
SETB IE.7 ; Enable Interrupts
SETB IE.0 ; Enable External Interrupt 0
MOV SCON, #50h ; Sets Serial Port into Mode 1 and enables Receive bit
; SCON bits = 0101 0000 = 50h
Mov TMOD, #22h ; sets mode for Timer1 to mode2 (auto-reload)
MOV TH1, #BAUD_CONST ; sets the delay initial and any following
MOV TL1, #BAUD_CONST ; sets the delay
SETB TI ; allow the initial output
SETB TR1 ; start Timer0 now
loop:
NOP ; loop endlessly
JMP loop
-----
writeChar ( Acc )
; This subroutine sends the character stored in ACC out P3.1
-----
writeChar:
JNB TI, writeChar ; wait until last SBUF has been send
MOV SBUF, A ; put the char to be sent into SBUF
CLR TI ; and indicate that the charater is ready for transmission
RET
-----
; This Interrupt Service Routine handles External Interrupts 0,
; triggered when: EA = IE.7 is set to 1 (Enable Interrupts)
; EX0 = IE.0 is set to 1 (Enable External Interrupt0)
; Port pin 3.2 is pulled low
-----
ISR_IE0:
PUSH ACC ; store current ACC on stack
MOV A, #'*' ; write out some character, to show we got here
CALL writeChar;
POP ACC ; restore ACC
RETI ; continue interrupted program
end

```

## Listing 2: Test data transfer from Encoder IC to Port Pin P1.0

```

-----
ECP-1 Project
Connect a 8x2 keypad to the 8032-SDK
05/01/2002

Wolf Paulus (wolf@paulus.com)
6606 Daylily Drive
Carlsbad, CA-92009
-----
Test keypad with port polling
-----
CODE_BASE      equ 4000h      ; HTE-8032SDK Code Base Address
BAUD_CONST     equ 0FDh      ; speed for internal serial port
BIT_BANG_SPEED equ 0A0h      ; port poll speed
-----
; On the HTE 8031SKD user programs incl. interrupt vectors,
; must be relocated and assembled starting at 4000h.

org CODE_BASE      ; sets base address for program start
JMP Main           ; jumps around ISR to start main prg.

main:

MOV  SCON, #50h    ; Sets Serial Port into Mode 1 and enables Receive bit
                    ; SCON bits = 0101 0000 = 50h
MOV  TMOD, #22h    ; sets mode for Timer1 to mode2 (auto-reload)
MOV  TH1, #BAUD_CONST ; sets the delay initial and any following
MOV  TL1, #BAUD_CONST ; sets the delay
SETB TI           ; allow the initial output
SETB TR1          ; start Timer1 now
SETB P1.0         ; init. Port Pin P1.0 to high
loop:
MOV  A, #'.'      ; just to do something
CALL WriteChar    ; simulating a signal
CALL GetKeypadEvent ; coming in on the interrupt line
                    ; convert the received character
SUBB A, #30h      ; use lookup table in code mem.
MOV  DPTR, #ExLookup
MOVC A, @A+DPTR

CALL WriteChar    ; send encoded ASCII char. to display
JMP  loop         ; loop endlessly
-----
; getKeyPressed( Acc )
; This subroutine receives a keypressed event from the keyboard
; encoder connected to port pin P.1.0
-----
GetBit:
JNB  TFO, GetBit ; wait for Timer Flag
MOV  C, P1.0     ; read port pin P1.0 and
CLR  TFO         ; write data into Carry bit
ret

-----
; GetKeypadEvent( Acc )
; This subroutine receives a keypressed event from the keyboard
; encoder connected to port pin P.1.0
-----
GetKeypadEvent:
JB  P1.0, GetKeypadEvent
MOV  TH0, #BIT_BANG_SPEED ; sync with the senders clock
MOV  TLO, #BIT_BANG_SPEED ; reset timer value
SETB TRO         ; start timer

CALL GetBit ; 1
RRC  A
CALL GetBit ; 2
RRC  A
CALL GetBit ; 3
RRC  A
CALL GetBit ; 4
RRC  A

```

## Connect an 8x2 Keypad to the HTE-8031SDK

```
CALL GetBit ;5
RRC A
CALL GetBit ;6
RRC A
CALL GetBit ;7
RRC A
CALL GetBit ;8
RRC A

CLR TRO
RET

;-----
; WriteChar ( Acc )
; This subroutine sends the character stored in ACC out P3.1
;-----
WriteChar:
JNB TI,writeChar      ; wait until last SBUF has been send
MOV SBUF,A            ; put the char to be sent into SBUF
CLR TI                ; and indicate that the charater is ready for transmission
RET

;-----
; This Interrupt Service Routine handles External Interrupts 0,
; triggered when: EA = IE.7 is set to 1 (Enable Interrupts)
;                EX0 = IE.0 is set to 1 (Enable External Interrupt0)
;                Port pin 3.2 is pulled low
;-----
ISR_IE0:
PUSH ACC
CALL GetKeypadEvent
CALL writeChar
POP ACC
RETI

;-----
; Lookup Table
;-----
ExLookup:
db '123U456D78*****9SC0HE'
END
```

### Listing 3: Final (Interrupt driven) Version of the Software

```

-----
ECP-1 Project
Connect a 8x2 keypad to the 8032-SDK
05/01/2002

Wolf Paulus (wolf@paulus.com)
6606 Daylily Drive
Carlsbad, CA-92009
-----
CODE_BASE      equ 4000h      ; HTE-8032SDK Code Base Address
BAUD_CONST     equ 0FDh      ; speed for internal serial port
BIT_BANG_SPEED equ 0A0h      ; port poll speed
EXIT_ADDR      equ 79Dh      ; SDK-Monitor code address
-----
; On the HTE 8031SKD user programs incl. interrupt vectors,
; must be relocated and assembled starting at 4000h.
org CODE_BASE      ; sets base address for program start
JMP main           ; jumps around ISR to start main prg.
-----
org CODE_BASE + 3h ; ISR vector for External Interrupt 0
AJMP ISR_IE0       ; jumps to the ISR
-----
main:
MOV SP, #7Fh      ; moves stack to upper intrnl memory
MOV SCON, #50h    ; Sets Serial Port into Mode 1 and enables Receive bit
                  ; SCON bits = 0101 0000 = 50h
MOV TMOD, #22h    ; sets mode for Timer1 to mode2 (auto-reload)
MOV TH1, #BAUD_CONST ; sets the delay initial and any following
MOV TL1, #BAUD_CONST ; sets the delay
SETB TI           ; allow the initial output
SETB TR1          ; start Timer1 now
SETB P1.0         ; initializes the port that is getting polled
SETB IT0          ; programs INTO port for transition interrupt mode
SETB EA           ; enables Interrupts
SETB EX0          ; enable External Interrupt 0
MOV R0, #0        ; R0 will contain the last pressed key code

loop:
NOP
DJNZ R1, loop    ; pause
DJNZ R2, loop    ; pause even more ..
MOV A, #'.'      ; put some character into ACC
CALL writeChar   ; write something out to show processing
CJNE R0, #'D', loop ; loop until down arrow key is pressed

CLR TI           ; Clean up
CLR TR1          ; .
CLR P1.0         ; ..
CLR IT0          ; ...
CLR EA           ; ....
CLR EX0          ; .....
CALL EXIT_ADDR   ; exit to monitor
-----
writeChar ( Acc )
; This subroutine sends the character stored in ACC out port pin P3.1
-----
writeChar:
JNB TI, writeChar ; waits for XMIT buffer to clear
CLR TI            ; clears TI Bit prior to character is sent
MOV SBUF, A       ; sends the character, now that the buffer is clear
RET
-----
GetBit( Carry )
; Reads timer synchronized state of port pint P1.0
; Returns the state of port pin P1.0 in the carry flag
-----
GetBit:
JNB TFO, GetBit  ; waits for Timer Flag
MOV C, P1.0      ; assigns Carry the value of P1.0
CLR TFO          ; enables the next transmission
RET

```

## Connect an 8x2 Keypad to the HTE-8031SDK

```

-----
; GetKeypadEvent( Acc )
; This subroutine receives a keypressed event from the keyboard
; encoder connected to port pin P.1.0
; Returns the code of the pressed key in ACC
-----
GetKeypadEvent:
JB P1.0, GetKeypadEvent ; wait for the startbit, indicating
; the next transmission
MOV TH0, #BIT_BANG_SPEED ; sync. with the senders clock
MOV TLO, #BIT_BANG_SPEED ; reset timer value
SETB TR0 ; start timer

CALL GetBit ; get the 1st bit in the carry bit
RRC A ; rotate the carry bit into A
CALL GetBit ; 2.
RRC A
CALL GetBit ; 3.
RRC A
CALL GetBit ; ... for all 8 bits
RRC A
CALL GetBit ; 5.
RRC A
CALL GetBit ; 6.
RRC A
CALL GetBit ; 7.
RRC A
CALL GetBit ; 8.
RRC A

SUBB A, #30h ; undo conversion done in chip
MOV DPTR, #ExLookup ; lookup key in lookup table
MOVC A, @A+DPTR ; assign looked up value
CLR TR0 ; stop timer, since data collection is done
RET

-----
; This Interrupt Service Routine handles External Interrupts 0,
; triggered when: EA = IE.7 is set to 1 (Enable Interrupts)
; EX0 = IE.0 is set to 1 (Enable External Interrupt0)
; Port pin 3.2 is pulled low
; Modifies R0 to contain key code of the pressed key
-----
ISR_IE0:

PUSH ACC ; Push everything the ISR might mess up
PUSH PSW ; therefore, A, PSW, and DPTR are
PUSH DPH ; getting stored on the stack
PUSH DPL

CALL GetKeypadEvent ; read and decoded key-press event
CALL WriteChar ; write out the key-label
MOV R0, A ; makes key available to main program

POP DPL ; restore all previously stored registers
POP DPH ; before continuing the interrupted program
POP PSW
POP ACC
CLR IE0
RETI

-----
; Lookup Table
-----
ExLookup: ; defines a loopup table in program mem. space
db '123U456D78*****9' ; the hardware encoder adds 0x30 to keys 0..9
db 01h ; and 0x37 to keys 10..15.
db 0Ch ; the resulting gap is filled with 7 stars which
db '0?' ; should never be requested from the lookup table
db 0Dh ; some values were chosen to accomodate a specific
end ; LCD (i.e. 0Ch will clear the LCD's screen)
-----

```