



AVR240: 4 x 4 Keypad – Wake up on Keypress

Features

- 16 Key Pushbutton Pad in 4 x 4 Matrix
- Very Low Power Consumption
- AVR in Sleep Mode and Wakes Up on Keypress
- Minimum External Components
- ESD Protection Included if Necessary
- Efficient Code
- Complete Program Included for AT90S1200
- Suitable for any AVR MCU

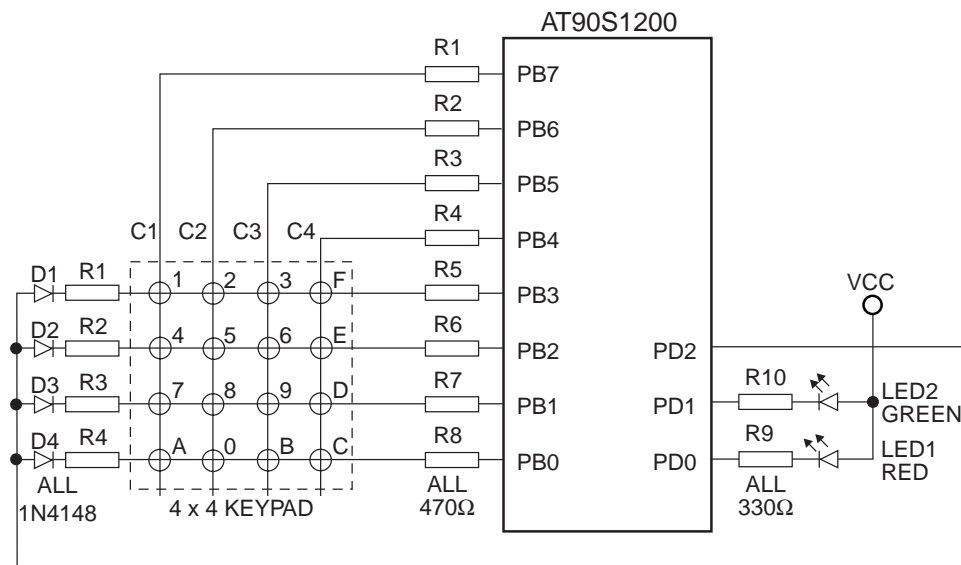
Introduction

This application note describes a simple interface to a 4 x 4 keypad designed for low power battery operation. The AVR spends most of its time in power down mode, waking up when a key is pressed to instigate a simple test program that flashes one of two LEDs according to the key pressed. If 0 (zero) is pressed the RED LED flashes 10 times. All other keys flash the GREEN LED the number of times marked on the key (e.g. if “C” is pressed the GREEN LED flashes twelve times).

4 x 4 Keypad – Wake up on Keypress

Application Note

Figure 1. Keypad and LED connections



Theory of Operation

The keypad columns are connected to the high nibble of port B. The keypad rows are connected to the low nibble. Resistors R1 to R8 (this is shown in Figure 1) serve to limit input current to a safe level in the event of ESD from the

keypad. They can be omitted in most applications.

In the steady state condition the high nibble is configured as outputs and are in the low state. The low nibble is configured as inputs and has the internal pull-ups enabled, removing the need for



external pull-up resistors. After initialization the AVR is put to sleep. When a key is pressed one of the diodes D1-D4 pull down the external interrupt line PD2, which also has internal pull-ups enabled. This wakes up the AVR and causes it to run the interrupt service routine which scans the keypad and calculates which key is pressed.

It then returns to the main program and drives the LEDs according to the key pressed, putting the AVR back to sleep when it has finished.

Resistors R9 and R10 are the traditional current limit resistors for the LEDs and can be any suitable value for the supply rail. This application note was tested using 330 ohms on a 5v supply. The LEDs are driven in current sink mode ("0" = ON) and provide about 10 mA of forward current with the values specified.

Implementation

The firmware consists of three sections, the reset routine, the test program and the interrupt service routine sets up the ports, sleep mode, power saving and the interrupts. The test program flashes the LED on wake-up and the interrupt service routine responds to the keypress.

Reset Routine

The flowchart for the Reset Routine is shown in Figure 2. On reset the ports are initialized with their starting directions. These are fixed on port D, with all bits as outputs except PD2 which must be an input for the external interrupt. This bit has its pull-up enabled by setting bit 2 of Port D. The unused bits are configured as outputs to avoid noise pickup or excessive power consumption which could otherwise occur if left floating. Port B starts with the high nibble as outputs sending out zeroes, and the low nibble set as inputs with the pull-ups enabled.

Since we are using a minimum of external components, we must ensure that internal pull-ups are turned on for all those bits set up as inputs. This is achieved by configuring the Data Direction Register with "1"s for outputs, "0"s for inputs, and then writing "1"s to the input bits in the PORT register. The inputs can then be read or tested from the PIN register. This program looks for "0"s and uses the SBIS instruction to skip over the keypress action if not a "0".

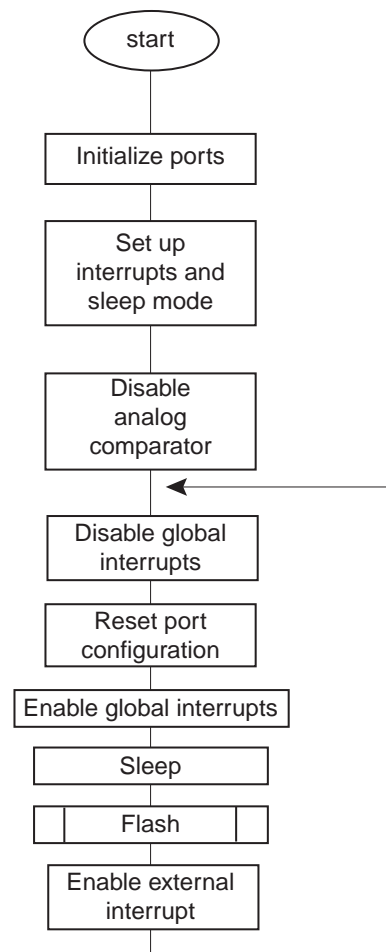
Power down mode is selected by setting the SE and SM bits of the MCUCR. At the same time the external interrupt configured by writing "0"s into the ISC00/01 bits. This will set the external interrupt INTO to trigger on a LOW level. When using "power down" mode the AVR can only be woken up by LOW LEVEL trigger.

Power consumption is reduced further by turning off the analog comparator. This is done by setting the ACD bit in the ACSR register. This must be done with care, otherwise

an unwanted interrupt can be generated. This program disables global interrupts until the program is ready to be interrupted, solving this problem. If you wish to use the analog comparator this code can be removed, but you will need to change ports for the keypad since port B is used for this.

The AVR then enters sleep mode. This is placed in the main loop to ensure that it goes back to sleep after it has finished its interrupt function and carried out the "flash" test routine. When the AVR wakes up after a keypress, the "flash" routine is called after the interrupt routine is finished. When the "flash" routine is done, the external interrupt is enabled, so that another interrupt can occur.

Figure 2. Flowchart for "flash" function



Flash Test Function

The flow chart is shown in Figure 3.

This function can be replaced by your own application to be executed out of "power down" mode. It serves to demonstrate that the key scan routine is working correctly. The value of the key pressed is taken from the "key" variable

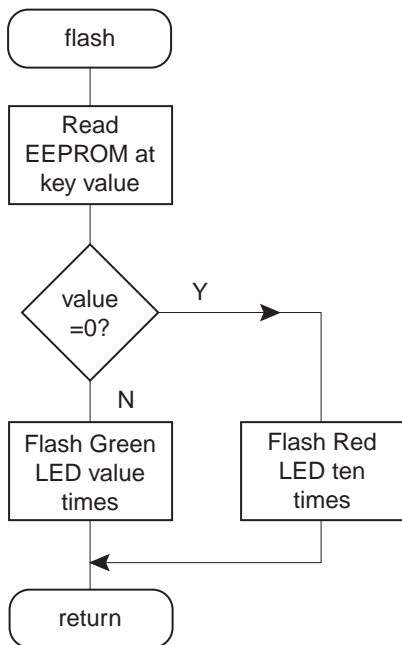
and used as a pointer to access a 16 byte look-up table stored in EEPROM. The look-up table contains the number of the key pressed.

The table has been used for two reasons, it makes the program much shorter, and it allows easy extension to provide full ASCII coding for the key press. For the larger AVRs it would make sense to store this table in program memory and access it using the LPM instruction.

The key value derived from the EEPROM is then used as a countdown variable inside an ON/OFF loop for the LED outputs. If the value is "0" the RED LED is flashed 10 times. If the value is non-zero the GREEN LED is flashed that number of times. e.g Three times for the "3" key, fifteen times for the "F" key etc., The AVR then repeats the loop and falls asleep.

The LED flashing routine is easily modified for your own application, replacing the "flash" function by your routine. The main consideration is the timing. Because the test program spends some time flashing the LEDs, no extra debounce arrangements are necessary. If your code is very fast you might need to put a short delay in to allow time for contact bounce. Wake up from sleep mode typically takes 16 ms or so, although this is being reduced on the newer devices. This also provides some debouncing.

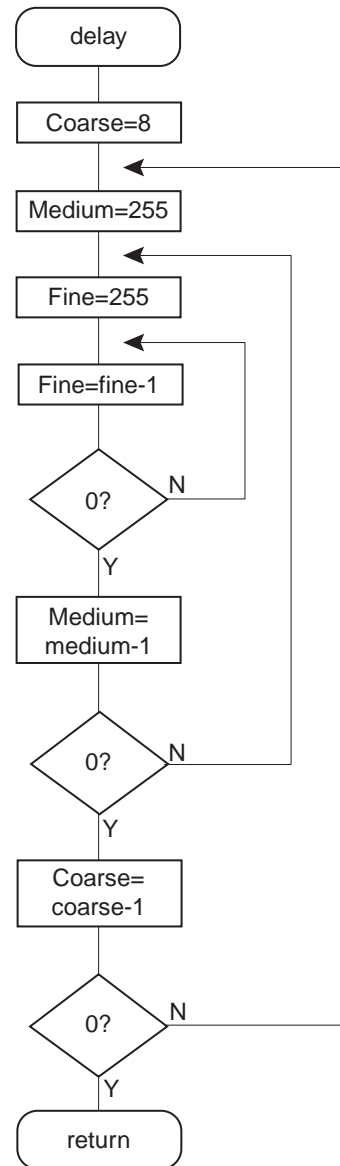
Figure 3. Flowchart for Reset and Main Routine



Long Time Delay Subroutine (delay)

To see the LEDs flash requires a delay of about 0.25 second. This is achieved using a conventional FOR loop to keep the timer/counter free for other work. To achieve 0.25 second with a 4 MHz clock requires three nested loops. Three local variables contained in registers "fine", "medium" and "coarse" are used for the loop. The fine and medium counters run the maximum of 255 times with the coarse counter set to 8, giving just over 0.25 second delay. The flowchart is shown in Figure 4.

Figure 4. Flowchart for Delay Subroutine



Interrupt Service Routine

On entry the status register is preserved to avoid corrupting any work the main program was doing. In this application it may be left out for optimisation if you wish. The flowchart is shown in Figure 5.

The key row is first detected by testing each row input in turn looking for "0". A base number 0, 4, 8 or 12 is then assigned to the variable "key". The ports are then reinitialized with port B I/O swapped over so that the key rows are tested.

A short time delay "settle" is used to allow the pins time to discharge. This takes the form of a conventional time waste loop using a FOR loop arrangement.

The key column is then detected and a number assigned in a temporary variable "temp" of 0, 1, 2 or 3. The final key-press is then computed by adding "key" and "temp", placing the result in "key" ready for use by the "flash" function. This method is easier to code than the conventional single bit scan in this application.

The port B configuration is the swapped back prior to restoring the status register. This saves using the settling delay again.

At the end, the external interrupt is disabled. This is done to avoid the interrupt routine being triggered again immediately upon exit.

Short Time Delay Subroutine

This short delay is required when changing the port B I/O configuration to allow time for the pin values to settle. The routine uses the global scratch register "temp" as a single loop counter for the FOR loop, set at maximum 255 passes. This provides a delay of 0.129 ms at 4 MHz. This value could be shortened by experimentation if time is of the essence or the pins are set high prior to reconfiguration to speed things up. This might remove the need for this delay completely.

Figure 5. Flowchart for Interrupt Service Routine

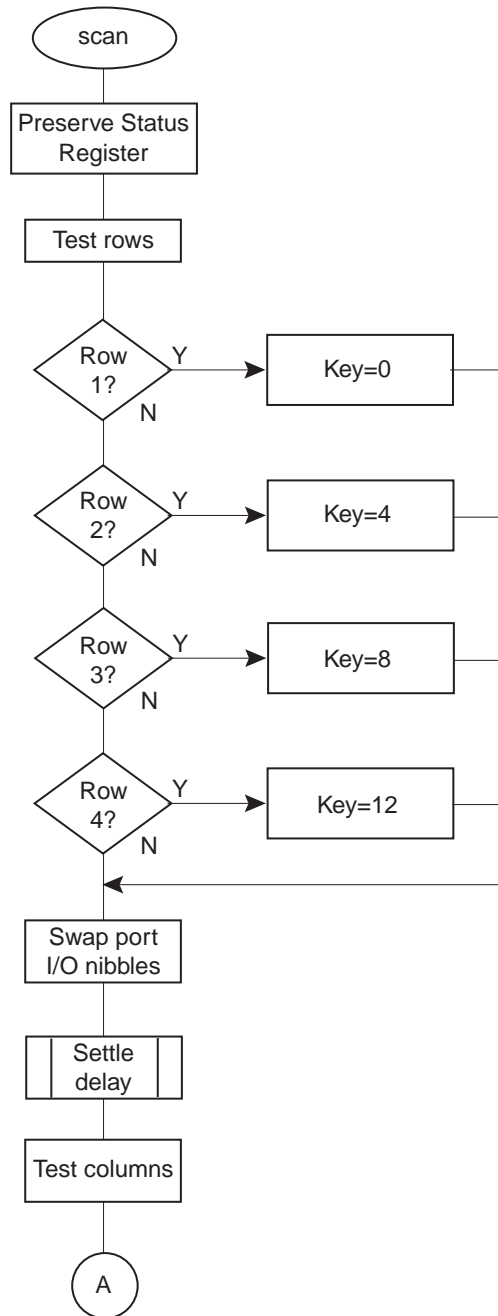
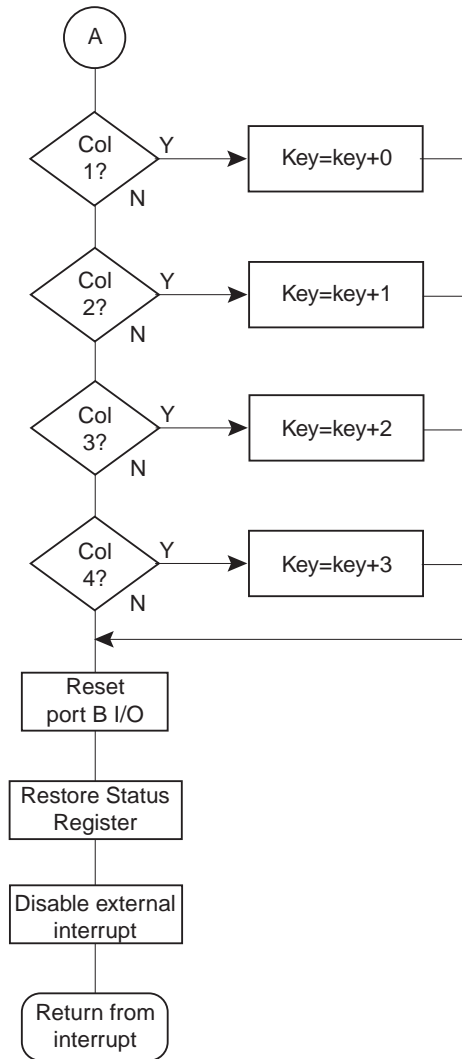


Figure 6. Flowchart for Interrupt Service Routine Continued



Resources

Table 1. Main CPU and Memory Usage

Function	Code Size	Cycles	Register Usage	Interrupt	Description
Main	24 words	19 cycles	R16	-	Initialization
Flash	20 words	-	R16	-	Example program
Scan	31 words	47 typical	R16, R17, R21	INT0	Scans 4x4 keypad
Delay	10 words	1,000,000	R18, R19, R20	-	0.25 second delay used in example program only
Settle	4 words	516 cycles	R16	.	Pin settling time delay used in scan
Total	87 words	-	R16, R17, R18, R19, R20, R21	-	

Table 2. Peripheral Usage

Peripheral	Description	Interrupts
External Interrupt 0 (INT0)	Key pressed wake up signal	External Interrupt 0 (Low Level triggered)
16 bytes EEPROM	Key to value mapping	-
8 I/O pins	4 x 4 keypad connections	-
2 I/O pins	Flashing LEDs for example only	-

```

;**** A P P L I C A T I O N   N O T E   A V R 240 *****
;*
;* Title:4x4 keypad, wake-up on keypress
;* Version:1.1
;* Last Updated:98.05.30
;* Target:All AVR Devices
;*
;* Support E-mail:avr@atmel.com
;*
;* DESCRIPTION
;* This Application note scans a 4 x 4 keypad and uses sleep mode
;* causing the AVR to wake up on keypress. The design uses a minimum of
;* external components. Included is a test program that wakes up the AVR
;* and performs a scan when a key is pressed and flashes one of two LEDs
;* the number of the key pressed. The external interrupt line is used for
;* wake-up. The example runs on the AT90S1200 but can be any AVR with suitable
;* changes in vectors, EEPROM and stack pointer. The timing assumes a 4 MHz clock.
;* A look up table is used in EEPROM to enable the same structure to be used
;* with more advanced programs e.g ASCII output to displays.
;*****

;**** Register used by all programs
;****Global variable used by all routines

.deftemp=r16;general scratch space

;Port B pins
.equ ROW1 =3; keypad input rows
.equ ROW2 =2
.equ ROW3 =1
.equ ROW4 =0
.equ COL1 =7; keypad output columns
.equ COL2 =6
.equ COL3 =5
.equ COL4 =4

;Port D pins
.equ GREEN =0;green LED
.equ RED =1;red LED
.equ INT =2;interrupt input

.include "1200def.inc"

;**** Registers used by interrupt service routine

.def key =r17;key pointer for EEPROM
.def status =r21;preserve sreg here

```

```

;***** Registers used by delay subroutine
;***** as local variables

.def    fine    =r18;loop delay counters
.def    medium  =r19
.def    coarse  =r20

;*****Look up table for key conversion*****
.eseg                                ;EEPROM segment
.org 0

    .db    1,2,3,15,4,5,6,14,7,8,9,13,10,0,11,12
;*****Source code*****
.cseg                                ;CODE segment
.org 0

    rjmp  reset                    ;Reset handler
    rjmp  scan                     ;interrupt service routine
    reti                             ;unused timer interrupt
    reti                             ;unused analogue interrupt

;*** Reset handler *****
reset:

    ldi  temp,0xFB                ;initialize port D as O/I
    out  DDRD,temp                ;all OUT except PD2 ext.int.
    ldi  temp,0x30                ;turn on sleep mode and power
    out  MCUCR,temp               ;down plus interrupt on low level.
    ldi  temp,0x40                ;enable external interrupts
    out  GIMSK,temp
    sbi  ACSR,ACD                 ;shut down comparator to save power
main:  cli                        ;disable global interrupts
    ldi  temp,0xF0                ;initialize port B as I/O
    out  DDRB,temp                ; 4 OUT  4 IN
    ldi  temp,0x0F                ;key columns all low and
    out  PORTB,temp               ;active pull ups on rows enabled
    ldi  temp,0x07                ;enable pull up on PD2 and
    out  PORTD,temp               ;turn off LEDs
    sei                             ;enable global interrupts ready
    sleep                          ;fall asleep
    rcall flash                    ;flash LEDs for example usage
    ldi  temp, 0x40
    out  GIMSK, temp              ;enable external interrupt
    rjmp main                      ;go back to sleep after keyscan

;*****Interrupt service routine*****
scan:

    in  status,SREG                ;preserve status register
    sbis PINB,ROW1                 ;find row of keypress
    ldi  key,0                      ;and set ROW pointer
    sbis PINB,ROW2

```



```

ldi key,4
sbis PINB,ROW3
ldi key,8
sbis PINB,ROW4
ldi key,12
ldi temp,0x0F                ;change port B I/O to
out DDRB,temp                ;find column press
ldi temp,0xF0                ;enable pull ups and
out PORTB,temp                ;write 0s to rows
rcall settle                  ;allow time for port to settle
sbis PINB,COL1                ;find column of keypress
ldi temp,0                    ;and set COL pointer
sbis PINB,COL2
ldi temp,1
sbis PINB,COL3
ldi temp,2
sbis PINB,COL4
ldi temp,3
add key,temp                  ;merge ROW and COL for pointer
ldi temp,0xF0                ;reinitialize port B as I/O
out DDRB,temp                ; 4 OUT 4 IN
ldi temp,0x0F                ;key columns all low and
out PORTB,temp                ;active pull ups on rows enabled
out SREG,status               ;restore status register
ldi temp,0x00
out GIMSK,temp                ;disable external interrupt
                                ;have to do this, because we're
                                ;using a level-triggered interrupt
                                ;go back to main for example program

reti

;***Example test program to flash LEDs using key press data*****

flash:        out EEAR,key                ;address EEPROM
              sbi EECR,EERE                ;strobe EEPROM
              in temp,EEDR                ;set number of flashes
              tst temp                    ;is it zero?
              breq zero                    ;do RED LED

green_flash:
              cbi PORTD,GREEN                ;flash green LED 'temp' times
              rcall delay
              sbi PORTD,GREEN
              rcall delay
              dec temp
              brne green_flash

exit:         ret
zero:         ldi temp,10
flash_again: cbi PORTD,RED                ;flash red LED ten times
              rcall delay
              sbi PORTD,RED
              rcall delay

```



```
    dec temp
    brne flash_again
    rjmp exit

;****Time Delay Subroutine for LED flash*****
delay:
    ldi coarse,8                ;triple nested FOR loop
cagain:  ldi medium,255          ;giving about 1/2 second
magain:  ldi fine,255           ;delay on 4 MHz clock
fagain:  dec fine
        brne fagain
        dec medium
        brne magain
        dec coarse
        brne cagain
        ret

;***Settling time delay for port to stabilise*****
settle:
    ldi temp,255
tagain:  dec temp
        brne tagain
        ret
```

