Design Note #047

AUTHOR: SUDHIR KAMAT

KEYWORDS: 4021, 4094, SPI, PORT EXPANTION

The Pin Adder.
Uses Seven Pins to Add 64 or more

Introduction

This might sound a like a rather strange name for a project but this is just what the circuit does "Adds I/O". The AVR series are already very brutal with respect to speed and performance but for me (and I suppose there are a lot of folks around there) having additional I/O is always welcome. The Parallel method using 74HC573 latches needs a dedicated latch enable per 573 and uses the eight pins of the bus. Cascading is thus difficult.

The I^2C port expander PCF8574 is good but I like to keep things simple and fast. This circuit allows huge cascading capability without the limits of addressing and I^2C routines. PCB layouts are much simpler and straightforward with this method. The SPI is used from the main MCU in the Master mode and this works right up to $F_{clk}/4$ (2 Mbps with 8 MHz XTAL) without any problem whatsoever. Remember to use some further buffer drivers for the LD, SCK, STR, OE, and LE signals if driving between different PCBs or a large number of chips.

I have designed an 80 input (10 x 4021) and 40 Relay output (5 x 4094) Logic Controller for an amusement park game and it works beautifully. In fact I even multiplexed the SPI to also read from a console keypad and drive display leds and several seven segment displays. This designs note is aimed at the AVR series having an SPI interface. But with a little ingenuity a software SPI can be written and then even an ATtiny can be made very powerful with a handful of Shift Registers.

Serial Input Driver

The Serial Inputs are handled by a CMOS 4021 chip. This is a very versatile Serial/Parallel In – Serial Out chip. A low to high strobe (LD) on the P/S Input (Pin 9) serves as the load signal to place the data at the parallel inputs into the Shift Register.

Every subsequent clock pulse transfers the data from the register to the output. When devices are cascaded the subsequent pulses after the first chip has serially output its data are accepted from the Serial Input (as P/S is now low) thus data from the subsequent chips shift into each other and to the MISO pin. The last chip should have its serial in grounded so that no noise creeps in.

Actually this is not very critical because you have already got your required data by then. P/S is therefore strobed once (LD) and the data from the required number of 4021 chips shifted in. The same routine can be repeated after a suitable debounce delay. I use a dedicated debounce byte per 4021 (each bit serves as a debounce indicator) but there are a lot of debounce options. Note the inputs of the 4021 are Idle high and Active Low. They can be further opto isolated for safety and noise surpression.

SPCR Set Up Routine

The SPI Interrupt Enable bit must be set depending on whether a polling sequence is being used or an interrupt. I prefer the polling sequence though. The SPI is enabled by the setting of the SPE bit. Disabling this after the routine can stop the SPI operation if necessary. The actual inputs and the bits of the data byte are determined by the DORD (Data ORDer) option. The MSTR bit sets either Master or Slave operation. Master is the mode employed here. The CPOL bit sets the Idle clock state. This is Low in case of the 4021 & 4094 and must bet set accordingly. The CPHA bit sets the clock phase which is 0 for the 4021 and 4094. Bits SPR1 and SPR0 set the SPI Data rate. The speed can be changed to suit requirements but I have used $F_{clk}/4$ without any problem.

Serial Input Routine

To read a byte you must write a dummy byte to the SPDR. This byte appears on the MOSI pin (That’s why the next section uses a STR for the 4094 chips, to ignore this data.) so make sure nothing else is getting triggered. You can poll the SPCR for the SPI Transfer Complete Flag or use an interrupt based routine. Either way once the SPI Transfer is complete the data from the SPDR must be read and stored to a register or ram. I always use an array in RAM and put the whole process in a loop depending on the number of bytes to be read. That’s about it. The bytes from your inputs are in Ram and you can proceed to solve your debouncing problems etc.

```
.Cseg
;**************************READSPI ************************************
.equ      Table_input =0x60             ; Ram address to store input data
.def      data =r1                      ; Data Register
.def      temp =r16                     ; General Register
.def      temp2 =r17                    ; Counter for number of bytes
Readspi:  ldi r30,low(Table_input)      ; Load Z-pointer with ram address
ldi       r31,high(Table_input)
ldi       temp2,4                       ; Number of bytes to input counter
ldi       temp,0b01110000               ; No SPI Interrupt, SPI enabled,
                                        ; Data order LSB first
out       spcr,temp                     ; Master select, Clock polarity
                                        ; normally Low,
                                        ; Clock Phase 0, SPI clock rate
                                        ; Fclk/4
clr       temp                          ; Dummy byte to be written to SPDR
sbi       portb,0                       ; Strobe LD High to Load 4021s
nop
cbi       portb,0                       ; Restore LD
readloop: out spdr,temp
nop
poll:     sbis spsr,7                   ; wait for transfer to complete
rjmp      poll
done:     in data, spdr                 ; read data from SPDR
st        z+,data                       ; Store in Ram and increment ram
                                        ; address
dec       temp2                         ; Decrement byte counter
```

```
brne     readloop                      ; If not done go to readloop else
                                       ; whatever
whatever: rjmp whatever
;*******************; Remember to Disable SPI if necessary when done
```

Serial Output Driver

The Serial Output Driver is just the opposite of the Input Driver described above. The chip utilised is a simple CMOS 4094. This has a Serial Input (pin2) a Strobe Input (pin1) a SCK Input (pin3) and an Output Enable (pin15). The strobe selects whether the Serial Data is for this chip or not (therefore it ignores the clock pulses fed to the 4021 s in the input driver). An important observation is that the 4021 loads all the Parallel data in one shot. In comparison the 4094 serially shifts one input bit at a time to the output on each clock pulse. This means there is a finite time (depending on the number of 4094 s) where the outputs are changing depending on the data. The output enable pin can effectively tri-state the outputs but this can cause a temporary undefined state to the output devices. This is ok when driving LEDs and seven segment displays but avoidable when driving relays etc. The solution is therefore implemented in the form of an additional 8-bit latch (74HC573) which can be latched (LE) after all the data bytes have been serially output and are settled and stable. Photo-triac drivers such as the MOC3021, MOC3041 (Zero-Crossing) can also be driven using an open collector NPN transistor or a low drive current FET such as the BSS170.

Serial Output Routine

The SPCR set up routine (as described above) must be followed first before data is written to the SPDR. The data to be output must be in a defined register(s) or in Ram.

An output array is what I use in Ram. Once data has been written to the SPDR the SPCR can then be polled for the SPI Transfer Complete Flag or an interrupt based routine can be used to determine the Transfer Complete. Subsequent bytes can be read and transferred using a simple loop with a register or RAM increment. While driving seven segments please note the lack of series resistors. The voltage drop of the segment LED + the two 4148 diodes serves to establish the brightness and equally distribute all current.

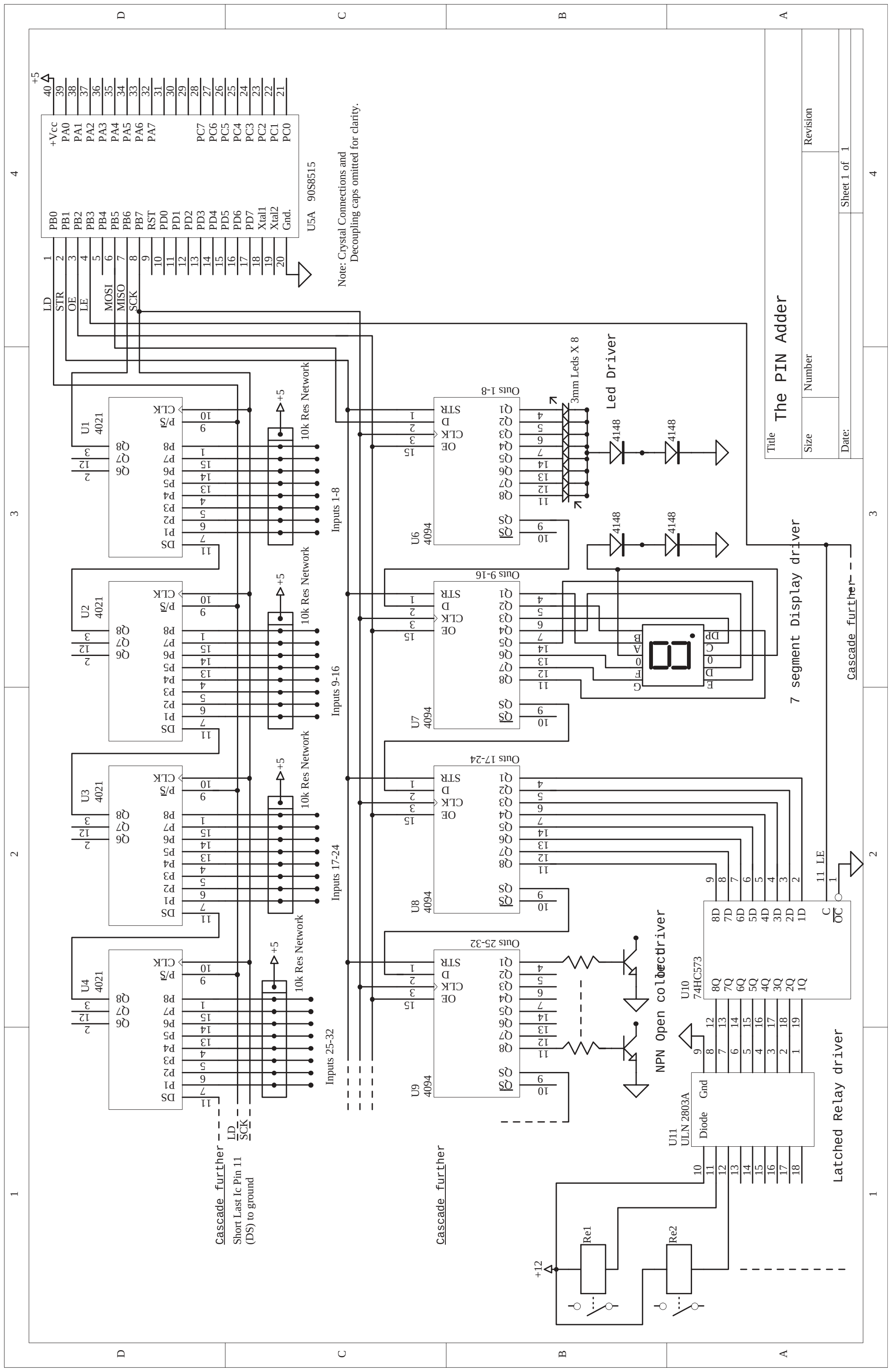
Use good quality displays and the brightness will never be a problem. The output of the 4094 does saturate but this is no problem and I have been using this for years without any problem whatsoever. The same applies for driving LEDs. The beauty of driving seven segment displays is the layout on the pcb is very simple because you can route any of the segments to any of the 4094 outputs. A simple lookup table will then have to be created to display the corresponding data on the display. In fact I have even used different pcb artwork for different seven segment displays adjacent to each other. This has allowed me to use a single side pcb and all I had to do was create separate lookup tables for each display.

Rememberto check on the MSB/LSB first bit as this can cause some confusion.

Notes:
1. The code does not include the Strobing of the LE (Latch Enable of the 74HC573) for the Latched relay outputs. Remember to keep this pin low normally and strobe it high and back low ---after all the data has been shifted to the various 4094s.
2. On power up a Display clear routine sending 0b00000000 to all 4094s should be usedto clear the initial states of the 4094s.

```
.Cseg
;**************************WRITESPI ***********************************
.equ        Table_output =0x60          ;Ram address of data to be output
.def        temp =r16                   ;General and data register
.def        temp2 =r17                  ;Counter for number of bytes
Writespi:   ldi r30,low(Table_output) ;Load Z pointer with ram address
ldi         r31,high(Table_output)
ldi         temp2,4                     ;Number of bytes to output counter
ldi         temp,0b01110000             ;No SPI Interrupt, SPI enabled, Data
                                        ;order LSB first
out         spcr,temp                   ;Master select, Clock polarity
                                        ;normally Low, Clock Phase 0 , SPI
                                        ;clock rate Fclk/4
cbi         portb,2                     ;Tristate 4094 outputs ( use if
                                        ;necessary )
sbi         portb,1                     ;Enable the 4094s STR pin as data is
                                        ;routed to them
writeloop:  ld temp,z+                  ;Read data from Ram and increment Ram
                                        ;address
out         spdr,temp                   ;Write data to SPDR
nop
poll:       sbis spsr,7                 ;Wait for transfer to complete
rjmp        poll
done:       dec temp2                   ;Decrement byte counter
brne        writeloop                   ;If not done go to readloop else exitspi
exitspi:    sbi portb,2                 ;Enable 4094 outputs ( use if necessary)
cbi         portb,1                     ;Disable the STR of the 4094s
whatever:   rjmp whatever               ;do whatever next
;********************************
;Remember to Disable SPI if necessary when done
```

Figure 1. The Pin Adder


The PIN Adder
Led Driver
7 segment Display driver
NPN Open collector driver
Latched Relay driver
Cascade further
Note: Crystal Connections and Decoupling caps omitted for clarity.
U5A 90S8515
Inputs 1-8
Inputs 9-16
Inputs 17-24
Inputs 25-32
10k Res Network
3mm Leds X 8
U10 74HC573
U11 ULN 2803A
Short Last Ic Pin 11 (DS) to ground
Re1
Re2
+12
Sheet 1 of 1