ZBasic

Application Note

# AN-203 I/O Expansion Techniques

## Introduction

On more complex projects it is often the case that more I/O lines are needed than the number that are available on the chosen processor. In this situation, you might first try to redesign the application to require fewer I/O lines. For example, you may be able to use a given output for multiple purposes. If you still need additional I/O lines, there are several techniques that you might employ, separately or in combination, to solve the problem.

Although the examples given in this application note are based on the ZX-24, the principles can be equally well applied to the other ZX processors.

## Using a Multiplexer for Multiple Inputs

If your application has a number of signals that only need to be connected to the processor when it is time to read the input, you may benefit from utilizing an external digital multiplexer. These devices allow 1 of N input signals to be selected for passing on to the processor, where N is a power of two. Examples of such a device are the 74HC151 (single 8 to 1), the 74HC153 (dual 4 to 1) and the 74HC157 (quad 2 to 1). Figure 1 shows how a '151 can be connected to a ZX-24.

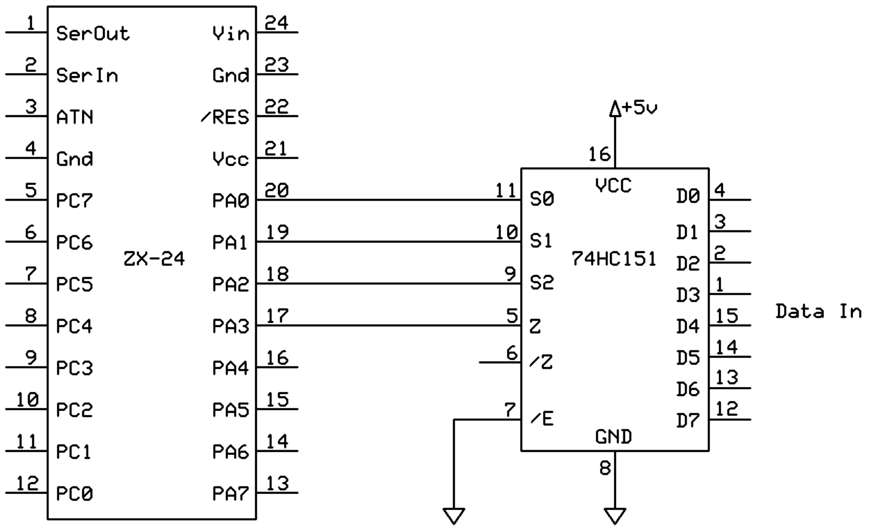

**Figure 1 Multiplexing 8 Digital Inputs**

In this example, using the multiplexer allows up to eight inputs to be read while only using 4 I/O lines of the processor. The I/O pins 18-20 are used to select the desired input and pin 17 is used to read the state of the selected input.

An example of the code required to read one of the 8 inputs is given below. Note that the example code may not work well in a multi-tasking environment depending on how the ports are used. The reason is because the first line in the GetInput() function reads the value of Register.PortA, modifies it and writes it back. In a multi-tasking environment, it is possible for a task switch to occur between the time when the value of Register.PortA is read and when it is written back. If another task gets control between those actions and that task modifies the value of Register.PortA, the value written back when the Main() task resumes will be incorrect. This problem can be avoided by either using a semaphore to control access to Register.PortA or by locking the Main() task before the operation and unlocking it afterward. It should be pointed out, however, that if a task is awaiting an external interrupt, locking the task is not sufficient to avoid this problem because an external interrupt will cause the current task to be suspended even if it is locked.

```
Private Const dataPin as Byte = 17
Private Const selMask as Byte = &H07

Dim b as Byte

Sub Main()
   ' initialize the I/O pins so that pins 18-20 are outputs
   Register.DDRA = Register.DDRA Or selMask

   ' read data from input channel 3
   b = GetInput(3)
End Sub

Function GetInput(ByVal channelNum as Byte) As Byte
   ' output the channel select code to the multiplexer
   channelNum = channelNum And selMask
   Register.PortA = (Register.PortA And Not selMask) Or channelNum

   ' read the selected input
   GetInput = GetPin(dataPin)
End Function
```

This technique can be extended even further by using stages of multiplexers. For example, the outputs of two 74HC151 devices could be multiplexed through a 74HC157. This would provide 16 inputs while using only 5 I/O pins. Note that the S2 to S0 inputs of both '151 multiplexers can be driven by the same output pins. Only one additional output pin would be required to feed to the '157 to select which "bank" of inputs is to be read.

The same multiplexing principle can be applied to analog inputs however an analog multiplexer must be used. The CD4066, 74HC4066 and MAX4634 are examples of a device that would work for this purpose.

## *Using a Decoder/Demultiplexer for Multiple Outputs*

A decoder is a device that outputs a "select" signal on 1 of N outputs based on the state of the selection inputs. A demultiplexer is a device that routes an input signal to 1 of N outputs based on the state of the selection inputs. The similarities between these two descriptions suggests why devices are often labeled decoder/demultiplexer – depending on how you connect it, it can be used to perform either operation.

The 74HC138 and 74HC139 are two examples of a decoder/demultiplexer. The former is a single 1 of 8 device while the latter is a dual 1 of 4 device. The circuit in Figure 2 shows how the 74HC138 can be connected to a ZX-24 to provide 8 select or strobe signals using only 4 processor outputs. Pins 18 to 20 provide the 1 of N selection code that determines which of the 8 outputs of the '138 will go low when the device is properly enabled. The '138 has three enable inputs: two active low and one active high. In this example, we've connected one of the active low enable inputs to pin 17 of the ZX-24 with two remaining enable inputs hard wired to their active states. When pin 17 of the ZX-24 goes low, one of the eight '138 outputs will go low depending on the A2-A0 inputs.

The outputs of the '138 could be used as an active low device select signal to an external circuit that enables the device to perform its function. If a device requires an active high select signal, the signal will need to be inverted before using it. The outputs of the '138 could also be used with edge-sensitive devices like flip-flops, shift registers or counters. When used this way the signal is commonly called a strobe or clock signal.

Note the pullup resistor used on the ZX output that drives the enable input. This resistor ensures that there will be no false outputs from the decoder during processor resets.

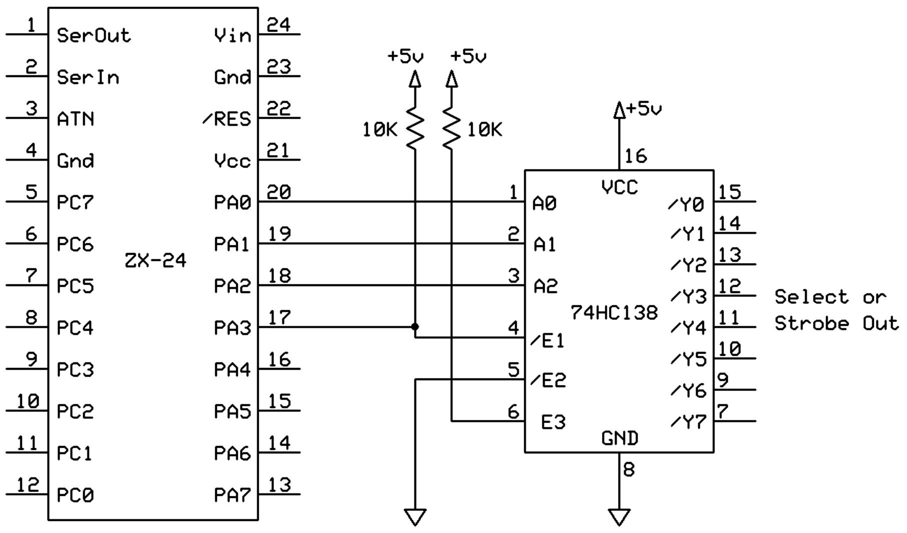

**Figure 2: Using a Decoder/Demultiplexer**

The code below illustrates how to drive an external decoder. As with the multiplexer example given earlier, this code may not work well in a multi-tasking environment depending on how the ports are used. Be sure to review the discussion of the multiplexer code to make certain that you understand the limitations.

```
Private Const enblPin as Byte = 17
Private Const selMask as Byte = &H07

Dim b as Byte

Sub Main()
   ' initialize the I/O pins so that pins 17-20 are outputs
   ' and pin 17 is high
   Call PutPin(enblPin, zxOutputHigh)
   Register.DDRA = Register.DDRA Or selMask

   ' send a strobe or select signal to channel 3
   Call StrobeOutput(3)
End Sub

Sub StrobeOutput(ByVal channelNum as Byte)
   ' output the channel select code to the decoder
   channelNum = channelNum And selMask
   Register.PortA = (Register.PortA And Not selMask) Or channelNum

   ' strobe the selected channel by sending the decoder a low-going pulse
   Call PulseOut(enblPin, 1, 0)
End Sub
```

## *Using a Shift Register for Multiple Outputs*

A shift register can be used to free up some I/O lines by sending the states of several output lines to the register using the synchronous serial interface. An example of the circuitry for this technique is shown in Figure 3.

The ZBasic ShiftOut() can be used to transmit data to the external shift register. The example circuit employs a 74HC595 that contains an 8-bit serial shift register and an 8-bit latch. This arrangement allows the outputs to remain in a steady state while a new set of bits is shifted into position using pins 14 (shift data) and 11 (shift clock). Once the bits have been shifted out, a rising edge applied to pin 12 (strobe) clocks the data into the output latch. A simple example of the ZBasic code required to implement this logic is given below.

```
Private Const dataPin as Byte = 20
Private Const clockPin as Byte = 19
Private Const strobePin as Byte = 18

Sub Main()
   ' initialize the I/O pins
   Call PutPin(dataPin, zxOutputLow)
   Call PutPin(clockPin, zxOutputLow)
   Call PutPin(strobePin, zxOutputLow)

   ' output some data to the shift register
   Call OutputData(&H55)
End Sub

Sub OutputData(ByVal dataVal as Byte)
   ' send 8 bits of data to the shift register
   Call ShiftOut(dataPin, clockPin, 8, dataVal)

   ' strobe the output latch to transfer the data to the register's outputs
   Call PulseOut(strobePin, 1, 1)
End Sub
```

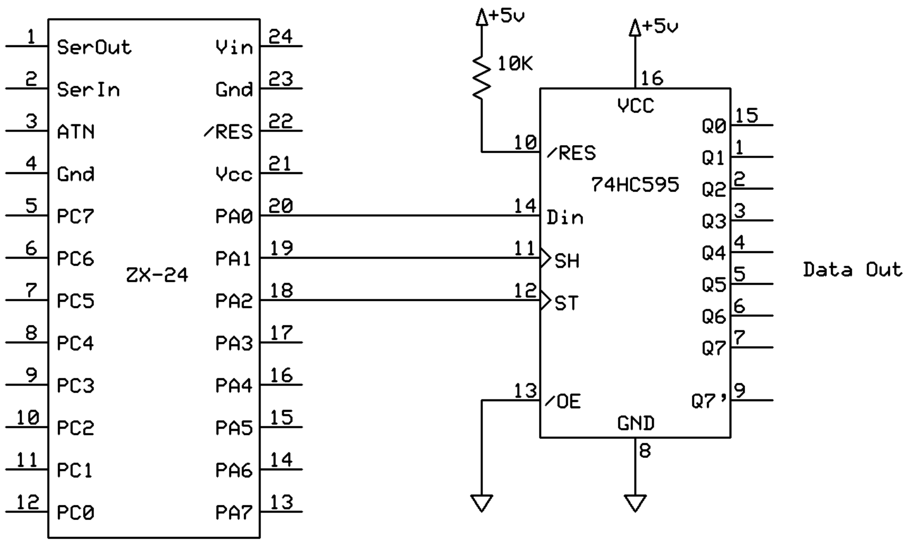

**Figure 3: Using a Shift Register for Multiple Outputs**

This technique can be extended by connecting together several shift registers either in "parallel" or in "series". In parallel mode all of the shift registers would be fed the same data and clock signals but each one would have its own output latch strobe signal. (A decoder/demultiplexer like the 74HC138 can be used to efficiently produce a large number of such unique strobe signals.) In series mode, the shift clock inputs of all the registers would be connected together. The data out from the ZX would be connected to the data in of the first shift register, the serial data out of the first shift register would be connected to the data in of the second shift register, etc. It is important to note that in the case of the '595 shift register, the serial data out signal (pin 9) is inverted. This can be addressed by adding an inverter between the stages or by inverting the data destined for the second register (and other even-numbered registers) before shifting it out.

Note that this technique can be combined with the input multiplexing technique described above, using some of the same outputs for multiple purposes, e.g. to select the input to be read and to shift data out to the shift register. Of course, the output latch strobe signal for the shift register would need to remain separate to avoid unwanted strobing.

Depending on the requirements of the application, a shift register with no output latch (e.g. the 74HC164) might also be used. The difference is that as the data is being shifted out the outputs of this type of shift register will "ripple" to their final state. If the data is being presented in parallel to an external device like an LCD this difference may be insignificant since those devices usually have there own "strobe" input to direct them to accept the data.

## *Using a Shift Register for Multiple Inputs*

A shift register can also be used to free up some I/O lines by sampling input data in parallel and then shifting it in serially. An example circuit is shown in Figure 4. The shift register shown has an input to control the loading of input data into the internal register (pin 1, /PL). To read the 8 data inputs you set the /PL input low and then high again. The logic level that was present at the P7 input of the shift register when the load signal goes high appears immediately on the Q7 output of the shift register. Then, on each subsequent rising edge of the clock input (pin 1, CP1 or pin 2, CP2) the logic level of the next lower input will appear at the Q7 output. The DS input (pin 10) is the data that is shifted into the internal register as it is clocked. If the Q7 output is connected to DS (instead of grounding DS as is shown), the register will maintain its data value through a cycle of 8 clock signals. This capability may or may not be useful in your application.

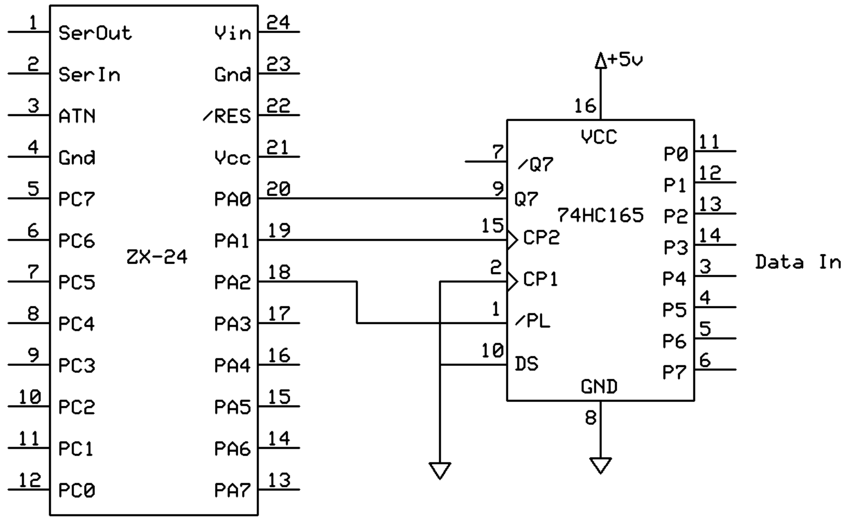

**Figure 4: Using a Shift Register for Inputs**

The sample code below shows how to read data in from the external shift register. Note, particularly, that the call to ShiftInEx() uses a 'flags' value that causes the data input to be sampled before the clock signal is generated. This is necessary since the shift register is already presenting the MSB of the data before the first clock transition occurs.

```
Private Const dataPin as Byte = 20
Private Const clockPin as Byte = 19
Private Const latchPin as Byte = 19

Dim b as Byte

Sub Main()
   ' initialize the I/O pins
   Call PutPin(dataPin, zxInputTriState)
   Call PutPin(clockPin, zxOutputLow)
   Call PutPin(latchPin, zxOutputHigh)

   ' read data from the shift register
   b = GetInput()
End Sub
```

```
Function GetInput() As Byte
   ' load data into the shift register
   Call PulseOut(latchPin, 1, 0)

   ' Shift in the latched data, the value of fourth parameter
   ' causes the data pin to be sampled before the clock edge.
   GetInput = CByte(ShiftInEx(dataPin, clockPin, 8, &H02))
End Function
```

## *Using an I/O Expander Chip*

The Philips PCF8574A is an I2C device that connects to the processor using only 2 I/O lines and provides 8 input/output lines. The device is also produced by Texas Instruments. Both manufacturers also produce the PCF8574 device; the only difference between the A and non-A part is the hard-wired portion of I2C device address. An alternate device, the Microchip MCP23016, offers 16 I/O lines and has more features at the expense of being slightly more complicated to use.

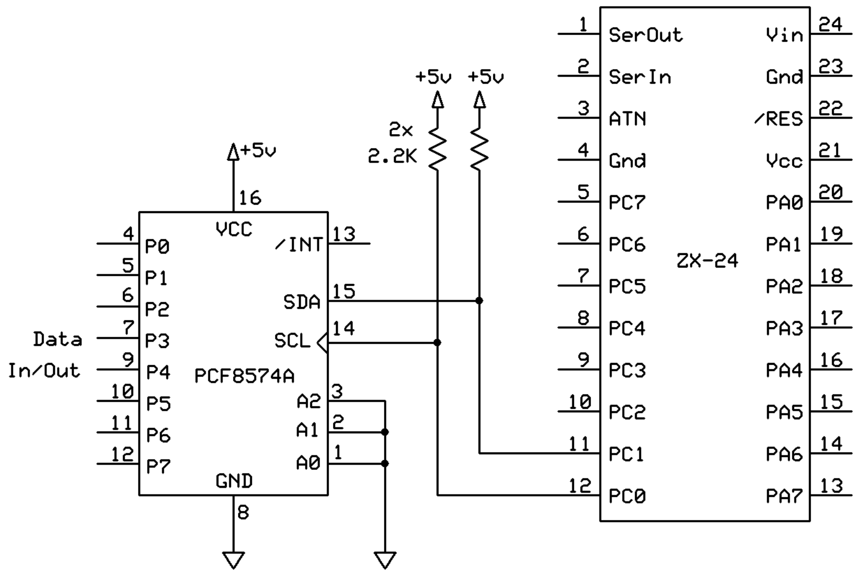

**Figure 5: Using a PCF8574A I/O Expander**

The schematic in Figure 5 shows how this device might be connected to the ZX-24. One advantage of using this device over the techniques described previously is that it provides quasi-bidirectional pins, meaning that the added pins can be used for input or output. Moreover, it is easily expanded by just adding more devices, each wired with its own unique combination of A2-A0 address inputs.

The example code below illustrates how data can be sent out and data can be read in using a single command. Pins intended to be inputs to the PCF8574A should always be written as logic one bits. The example assumes that the most significant 4 bits of the expander are used as inputs and the least significant 4 bits are used as outputs.

```
Private Const i2cChan as Byte = 0
Private Const i2cSpeed as Integer = 66
Private Const i2cAddr as Byte = &H70
Dim stat as Integer
Dim b as Byte
```

```
Sub Main()
   Call OpenI2C(i2cChan, 0, 0, i2cSpeed)
   b = &Hf5

   ' write the low nibble, read back the high nibble
   stat = I2CCmd(i2cChan, i2cAddr, 1, b, 1, b)
End Sub
```

An interesting aspect of the PCF8574A is that it can generate an interrupt signal when the inputs to the device change. This can be useful in many situations where the processor needs to be alerted to a pin state change. The \INT output of the PCF8574A can be fed to one of the ZX processor's interrupt inputs: INT0, INT1 or INT2. The circuit shown in Figure 6 depicts the /INT output connected to the INT0 input of the ZX-24.

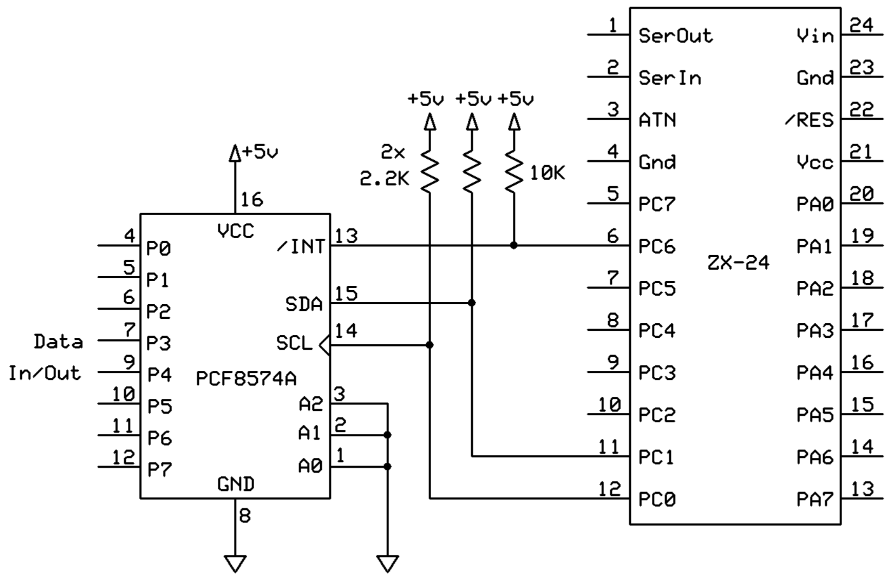

**Figure 6: Using the /INT Output of the I/O Expander**

The sample code below illustrates how this capability might be used. The task Task1 waits for a falling edge on INT0. When the falling edge occurs, indicating that the port value changed, the I/O expander port value is read. This action both retrieves the current port value and resets the /INT output of the PCF8574A, preparing it for the next input change.

A note of caution is in order. The datasheet indicates that, due to the way that the interrupt output is reset by read and write operations on the PCF8574A, there is a possibility that an input change may be missed. Consult the datasheet for a complete explanation.

```
Private Const i2cChan as Byte = 0
Private Const i2cSpeed as Integer = 66
Private Const i2cAddr as Byte = &H70
Dim stat as Integer
Dim b as Byte
Dim ts1(1 to 60) as Byte

Sub Main()
   Call OpenI2C(i2cChan, 0, 0, i2cSpeed)

   ' Write a value to the port expander, note that 1's must
   ' be written for pins that will be used as inputs.
   b = &Hf5  ' value chosen for example purposes only
   stat = I2CCmd(i2cChan, i2cAddr, 1, b, 0, 0)
   ' invoke the task that waits for a pin change
   CallTask "Task1", ts1

   ' add code here for other activities while awaiting the pin change
   Do
   Loop
End Sub

Sub Task1()
   Do
       ' await a falling edge on INT0
       Call WaitForInterrupt(zxPinFallingEdge, 0)

       ' read out the port value, resetting the interrupt
       stat = I2CCmd(i2cChan, i2cAddr, 0, 0, 1, b)

       ' do something here with the value read
       ' for this example, we just display the hexadecimal value
       Debug.Print CStrHex(b)
   Loop
End Sub
```

## Author

Don Kinzer is the founder and CEO of Elba Corporation. He has many years experience working with microprocessors, microcontrollers and general purpose computers. Don can be contacted via email at *dkinzer@zbasic.net*.

**e-mail: support@zbasic.net** **Web Site: http://www.zbasic.net**