
BASIC-Tiger® PIO

Gunther Zielosko, Heiko Grimm

1. Basics

You've probably already come across the problem of constructing a device using BASIC-Tiger® which, however, does not provide enough I/O pins. So you need an expansion module, and there are several in the Wilke Technology assortment. Many of them have additional outputs, inputs and even combined inputs and outputs in one module. Yet – those modules are appointed concerning the I/O pin's usage in contrast to the “real” BASIC-Tiger® I/Os. Both commands IN/OUT and DIR_PIN/DIR_PORT allow using the port connections variably. So what about providing extended ports (EPorts) using a Xilinx CoolRunner® module. Those ports can be used more or less arbitrarily as input or output (either as pins, tetrads or ports). In the following application note we will implement such a project, creating 2 new ports to begin with. If a bigger CoolRunner® module is chosen, considerably more I/O ports can be provided. This module, however, has a very complex package which means it is not a suitable object for a handicraft laboratory. It requires professional wiring of the printed circuit board, thus creating the need for an industrial solution.

2. Implementing a „Small“ Model Using XCR3064XL-10 PC44

As mentioned before the big CoolRunner® modules are very hard to handle. Therefore we suggest a model solution for 2 ports with 8 bits each. Due to the small module's restricted register and logic capacity limitations concerning programming had to be accepted. Bigger modules allow implementing 8 ports with 8 bits each which are freely programmable. Now we have one port at our disposal, the pins of which can be programmed by bit and one port, the pins of which can be programmed by tetrad (i.e. bits 0...3 and 4...7 together) as input or output. This is common with standard PIO modules as well and does not pose a major restriction. Addressing is carried out with the commands XIN and XOUT similar to the BASIC-Tiger®'s EPort system which we already used in earlier CoolRunner® experiments.

2.1. Hardware

Connecting the XCR3064XL to BASIC-Tiger® is quite simple. We just need the 8 pins of port 6 (common data lines for expansion modules) as well as three signals ACLK, DCLK and INE. This time these signals are not common pins L33, L34 and L35 but pins L80, L81 and L82. This is necessary because of possible conflicts with e.g. the keyboard. Figure 1 shows the XCR3064XL pin assignment and figure 2 shows its connection to BASIC-Tiger®.

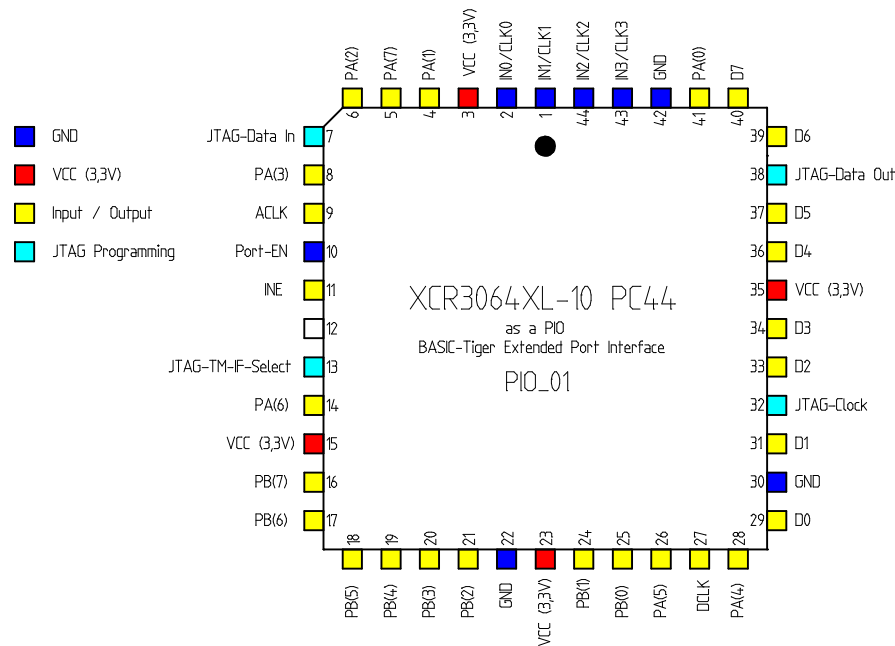


Fig. 1 PIO terminal assignment model

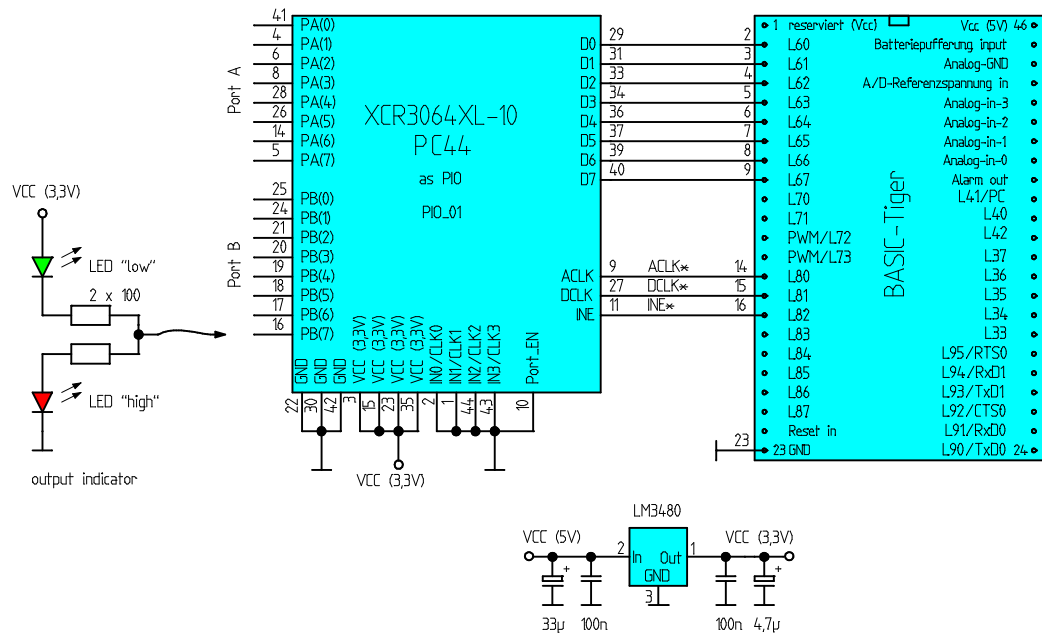


Fig. 2 Connection to BASIC-Tiger[®] using a simple logic indicator for outputs

Keep in mind that all VCC pins of CoolRunner[®] have to be connected to 3.3 V and that all this connections require a 100 nF blocking capacitor against ground soldered near to the pin.

Also consider that the CoolRunner[®] system's outputs can only deliver 3.3 V. If they are only loaded weak this should not be a problem for common logic versions such as LS or HCT.

For monitoring the finished module's outputs we present a simple indicator on the left of the schematic, which is also suitable for other logic outputs. When there is no logic signal (i.e. pin is set as input or high-resistant), both diodes light up. When logic "1" (high) is outputted, only the red diode lights up, in the case of logic "0" (low) only the green diode does. Since the LED's are only driven by the output in this case you can automatically estimate its driving capacity. As you can see the CoolRunner® delivers quite something...

2.2. Addressing

Before we start programming the BASIC-Tiger® just some comments concerning the PIO module addressing. We have two ports which must have their own address each. In contrast to previous BASIC-Tiger® system expansion modules we want to address both inputs and outputs with the same address. The BASIC-Tiger® EPort system is not designed for such a case. It has one address space for extended outputs and another one for extended inputs. This is why we use the relatively new commands XOUT and XIN which do not have this restriction. Besides our in- and outputs will not collide with the (unfortunately!) incomplete coded addresses of keyboard, DIL switches and the extended ports. Safe operation also requires independence from the three standard control signals ACLK (L33), DCLK (L34) and INE (L35). The command XSETUP allows an almost arbitrary usage of ordinary BASIC-Tiger® I/O lines for this purpose. We made

L80 an ACLK* signal,

L81 a DCLK* signal and

L82 an INE* signal

(* stands for further independent existence of the original signals for keyboard etc. Therefore we have extra signals for our PIO which are not jammed by other accessing attempts)

We need two more addresses to determine the data direction of both our new PIO ports. Although we only have to address two ports, we defined the ports' addresses and the belonging direction registers with a distance of 8 addresses, in order to keep the option of further developments using more I/O ports. The address space looks like shown in table 1. Obviously there are no differences between physical and logical addresses as it was common with EPorts. Addresses are created merely from port 6 data. An offset does not exist any more.

After Power on reset all pins of ports A and B are inputs. All outputs have to be programmed in a way that first of all direction is set by a XOUT command. Then with further XOUT or XIN commands the outputs and inputs are carried out. This also allows reading "back" the output pins' states using XIN.

Physical address	Register address in the XCR3064XL PC44 (PIO_01)	Function	Remarks
08h	08h	address port A	OUT or IN
09h	09h	address port B	OUT or IN
0Ah	0Ah		reserved
0Bh	0Bh		reserved
0Ch	0Ch		reserved
0Dh	0Dh		reserved
0Eh	0Eh		reserved
0Fh	0Fh		reserved
10h	10h	direction port A	1 input, 0 output by bit*
11h	11h	direction port B	1 input, 0 output by tetrad**
12h	12h		reserved
13h	13h		reserved
14h	14h		reserved
15h	15h		reserved
16h	16h		reserved
17h	17h		reserved

- * 10100101 Bits 0, 2, 5 and 7 are inputs
 Bits 1, 3, 4 and 6 are outputs
- ** XXXXXX01 Bits 0, 1, 2 and 3 (lower tetrad) are inputs
 Bits 4, 5, 6 and 7 (upper tetrad) are outputs

Table 1 PIO addressing

2.3. BASIC-Tiger® Programs PIO_01.TIG and PIO_02.TIG

BASIC-Tiger® program PIO_01.TIG demonstrates new possibilities using the 16 additional optional I/O pins. With the required hardware present we can kick off at once.

First of all let's take a look at commands of the XOUT/XIN kind, which were added to Tiger-Basic version 5.0, but unfortunately were not annotated in detail. These commands take less time, they can deal with whole groups of in- and outputs and are very versatile.

First we will need commands XOUT, XIN and XSETUP. Starting from the back, XSETUP allows choosing different pin combinations for controlling external modules. You can use either the original pins for external modules or new ones, in the case of bad coding and disturbing accesses of already present components. In our demo program we will use the following:

FLAG = XSETUP (6, 8, 0, 1, 2, 4, 0)

Port 6 is data bus	as with keyboard, display, expansion modules etc.
Port 8 is control bus	port 3 with previous components!
L80 is ALCL	L33 with previous components
L81 is DCLK	L34 with previous components
L82 is INE	L35 with previous components
Port 4 is control bus 2	has to be entered, but is not required here
L40 is CE pin	has to be entered, but is not required here

Before using XSETUP all required pins of the **new** control bus have to be defined as output using DIR_PIN or DIR_PORT commands. But that's not everything: The control signals' idle levels also have to be defined before – using an ordinary OUT command you set bits as they are supposed to be during idle state. This is different from signals ACLK, DCLK and INE of the same name which cannot be set arbitrarily! It allows reacting flexibly to all different sorts of hardware, which is actually quite handy. It is not handy though if nobody tells you...Our PIO component is programmed as follows:

ACLK*	high active	(idle level low)
DCLK*	high active	(idle level low)
INE*	low active	(idle level high)

With this done, we will continue programming. We need three kinds of commands, whose features are described in the following section.

Data direction definition (instead of DIR_PORT respectively DIR_PIN):

Basically is: 1 = input
 0 = output

Example:

XOUT (10h, 10100101b) at the first 8-bit port (08h)
 Bit 0 becomes input
 Bit 1 output
 Bit 2 input
 Bit 3 output
 Bit 4 output
 Bit 5 input
 Bit 6 output
 Bit 7 input

Please note!

- **Direction command always has an address that is 8 higher than the belonging port.**

- The underlying PIO's second port's direction is switched by tetrad. The two low-order bits (bit 0 = low-order tetrad, bit 1 = high-order tetrad) serve for this.

Data output (as ordinary XOUT):

Example:

XOUT (08h, 10011101b) Data byte 10011101 is outputted at the first port 08h. Only pins previously defined as output will give out information.

Data input (as ordinary XIN):

Example:

E = XIN (08h) Port 08h is read into variable E

Here you have the following options:

The whole port was defined as input before (XOUT 10h, 11111111b)

Data of all port 08h pins' are read just as the information is fed from outside

Only single bits were defined as input (XOUT 10h, 10011101b)

Data from pins defined as input (1) of port 08h are read just as the information is fed from outside

Data from pins defined as output (0) are read as set by a previous OUT command at these outputs (Option of reading back a previous OUT command!)

So there is nothing to bar a BASIC-Tiger® program anymore. The small program PIO_01.TIG demonstrates some possibilities of optional ports. It is mostly self-explaining. Figure 3 shows a screenshot from the running program.



Fig. 3 PIO_01.TIG running

Mask M is shown at the top (at addresses 10h and 11h for setting the pins' directions). Below we see data word D which has just been outputted (at addresses 08h and 09h), then data O which are really outputted at the pins set as output and at the bottom values I which are read at

the pins defined as input. Here the left group is the PIO channel A and the right group is PIO channel B. As you can see choice of direction takes place by pin at port A and by tetrad at port B. Choice of direction M is fixed in this program, output data are counted up slowly.

The second program PIO_02.TIG does exactly the same apart from the bottom line which shows all bits of both ports. This demonstrates the option of reading back previous output data by the input byte again.

3. Outlook

We made a small Xilinx CoolRunner® module a universal PIO module for 2 ports with 8 bits each. But the presented system is much more universal. The experiment's objective is to provide PIO modules (with e.g. 8 ports and 8 bits each) which fit into any Wilke Technology BASIC-Tiger® expansion module system.

Such components will need bigger CoolRunner® modules and, above all, a complex circuit board as well as a matching case.

Besides universal addressing in the available EPort system address space would be desirable. The EPort system's current addressing is incomplete due to the limited possibilities of discrete address generation with 1 out of 8 decoders 74HC138, and therefore is a source of error. This problem does not exist with the CoolRunner® system which allows complete address decoding. Conflicts with the Plug-and-Play-Lab's EPort addressing can be avoided with the suitable choice of address. Also the use of common control lines ACLK, DCLK and INE is limited for the new modules.

It would be interesting to have new commands for choice of direction (such as DIR_PORT and DIR_PIN) for the new expansion modules analogue to the BASIC-Tiger® ports (possibly XDIR_PORT and XDIR_PIN). This would enable us to use the same addresses for choice of direction as for the actual OUT or IN commands.

We added file PIO_01.JED to this application note for your experiments which can be used by CoolRunner® experts for "burning" the small PIO right away. For a more complex version we will have to wait for a new Wilke Technology development...

Let's wish ourselves success!