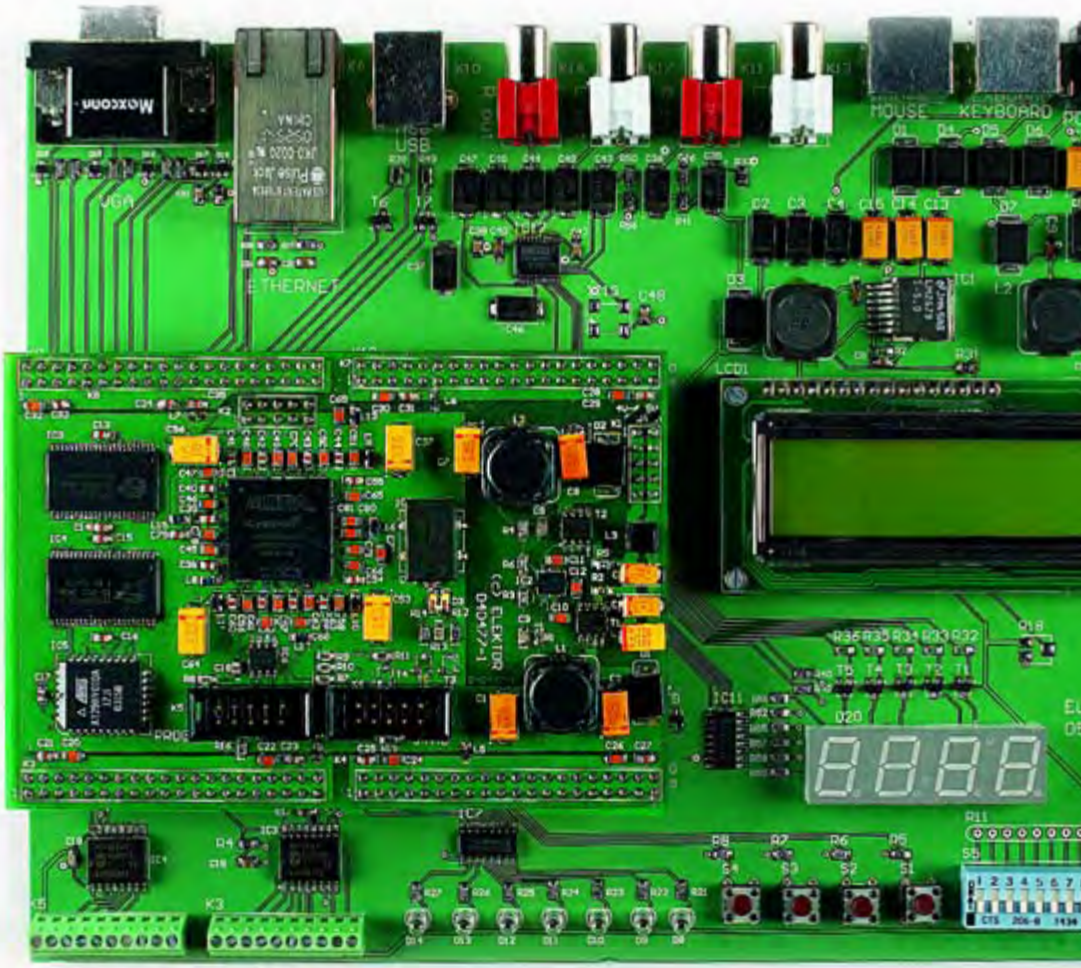
FPGA Course (5)

Paul Goossens



Every embedded system uses a system bus to transport data between the various components. This also applies to systems implemented in an FPGA. However, a different sort of bus system is commonly used in FPGAs. This month's article introduces a bus system that is often used in FPGAs.

A typical bus system in a 'normal' microprocessor circuit consists of a data bus, an address bus and several control signals, such as RD/~~WR~~. The peripheral ICs put their data on the bus when requested to do so. During the rest of the time, their inputs are in a high-impedance state to give other ICs a chance to put data on the bus. These data ports are tristate ports, which means they can be set to a high-impedance state.

Different

In many FPGAs, it is not possible to put internal signals in a high-impedance state. In addition, a mistake in the design can cause short-circuits or data corruption on the bus. Tristate ports are thus not used for the system bus in such systems. Another factor is that the peripheral electronics often runs at a different clock speed than the processor. This makes handshaking necessary to ensure correct data transport.

Several standard system busses have been developed to avoid problems of this sort in FPGA designs. This instalment focuses on a system bus called the 'Wishbone bus', which is used quite often. It is used a lot at *www.opencores.com*, a handy site where you can download free designs and subdesigns.

Minimum system

A minimum Wishbone bus with a single master and a single slave is shown in **Figure 1.** The dual data bus is clearly visible. Each bus is unidirectional – one bus carries data from the master to the slave, while the other bus carries data in the opposite direction.

The STB (strobe), CYC (cycle) and ACK (acknowledge) signals provide handshaking for each data transmission. The slave can only respond to the Wishbone signals if the STB_I and CYC_I signals are both high. The master sets WE (write enable) high to indicate that it wants to write data to the slave. If this signal is low, it means that the master wants to read data from the slave.

When the slave has finished processing the data, it signals this by setting the ACK signal high. The master pulls the STB signal low in response. The slave must then return its ACK output to the low state.

This handshake protocol makes it possible to connect a slow slave device to

Part 5: bus systems and interconnections

a much faster master, since the slave can set its ACK signal high sometime later. This gives a relatively slow slave enough time to process the data. **Figure 2** shows a read operation of this sort, in which the slave needs two extra clock cycles to complete the transaction.

Example

We have prepared a simple example in *ex13*. Here the 8051 microcontroller has a master interface for the Wishbone bus. The bus is connected to a simple slave device that enables the master to drive eight outputs. The slave interface causes the ACK signal to appear with a delay of 10 clock pulses. This makes it possible to display the handshake process using the logic analyser built into Quartus.

The processor used here (T8052) uses the Wishbone bus for all transactions with XRAM memory in the region starting at address 0x1000. The only extension unit on this Wishbone bus is an 8-bit output named *wish_output*. This extension also has an internal address decoder. This is normally placed in a separate bit of hardware, but for this simple example we placed it in the core instead.
Seven of the eight outputs are connected to the LEDs on the extension board. The software causes the LEDs to light up sequentially for a 'running-light' effect.

Internal

Processing the Wishbone signals is fairly simple. The *sel* signal detects whether the address on the system bus matches the address of the extension (0x8000).
The code starting on line 63 causes the outputs to go high after a reset. When a valid address appears (*sel* = '1') while a valid write cycle is in progress (STB = '1', CYC = '1' and WE = '1'), the data present at the DAT_I input is stored in the output register by the rising edge of the clock signal.
Generating the ACK signal is somewhat more complicated in this case because it has to be delayed. The COUNT signal keeps track of how many clock pulses have occurred since the last write operation to this core. When the value of this counter reaches 10, ACK_OK goes high. This signal indicates that an ACK signal can be generated now.
The ACK output signal is finally defined in line 101. This core also generates an ACK if an invalid address is placed on the bus (*sel* = '0'). This is designed to prevent the processor from hanging if the software accidentally uses an incorrect address.
Note that the ACK signal is an asynchronous signal. In other words, it is not generated using a flip-flop. This is one of the requirements of the Wish-

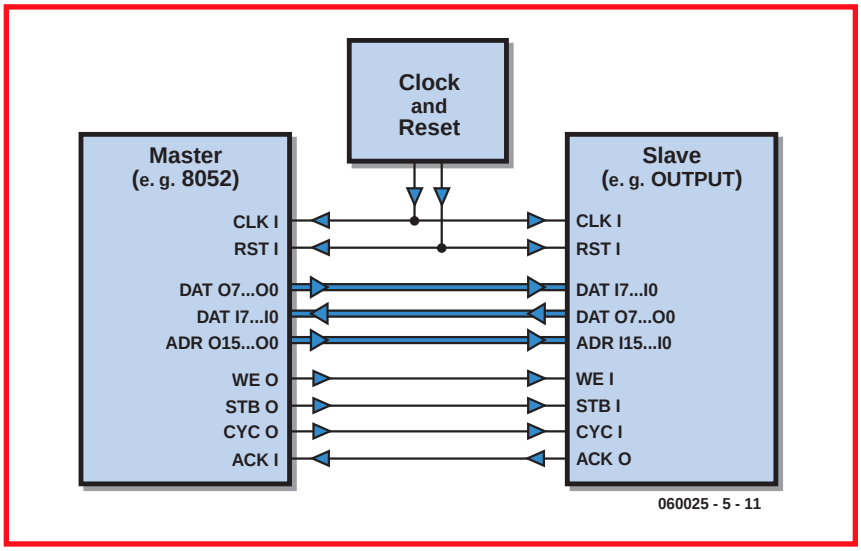


Figure 1. A minimum Wishbone bus with a single master and a single slave.

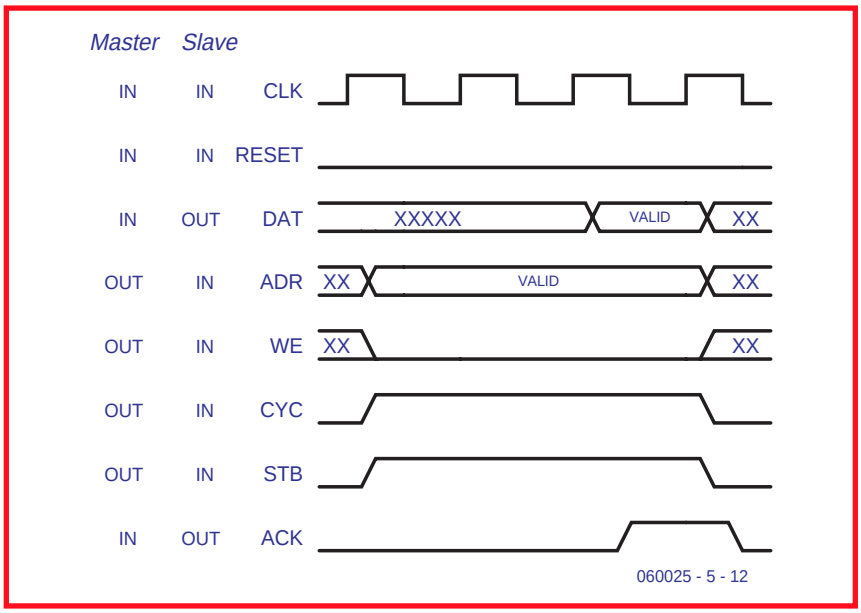


Figure 2. Here you can see that slave needs two extra clock cycles to complete the transaction.

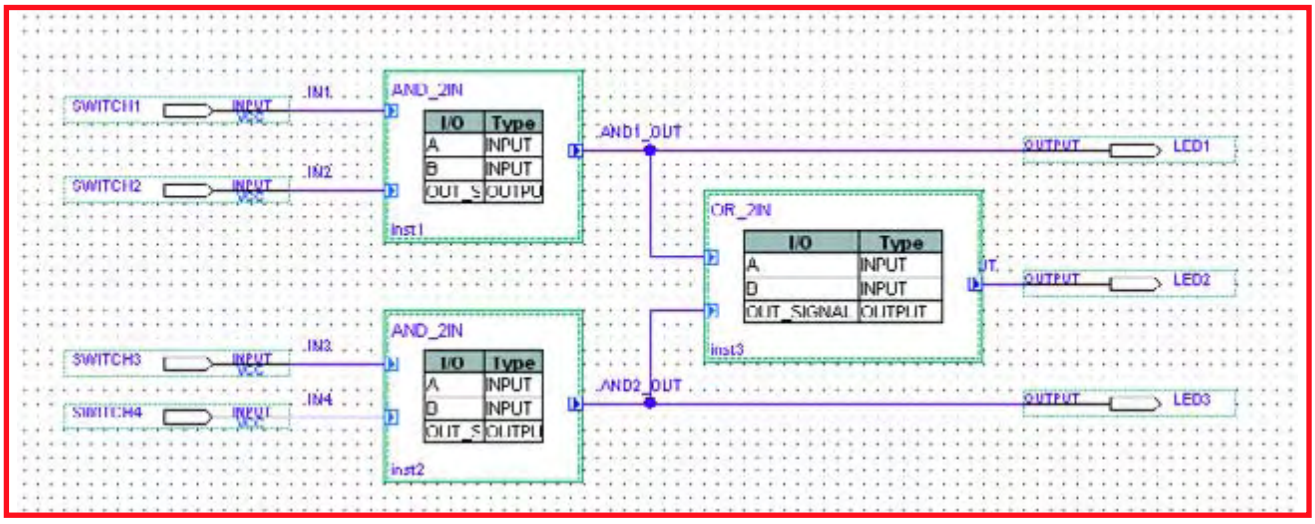


Figure 3. A simple design consisting of two VHDL files and a graphic file.

bone specification. ACK must go low in response to setting STB or CYC low.

Experiment

In the software, we send the same value to the output 20,000 times. This slows the running light down to the point that you can observe the effect visually.
ACK is delayed by 10 clock cycles in *wish_output*. If you increase this delay, the running light will also slow down. You can easily make this experiment yourself. Simply change line 87 of *wish_output.vhdl* to the following line:

IF (COUNT=200) THEN

Recompile the project and load it into the FPGA. Now the running light will run quite a bit slower than before. This proves that a slow slave on the Wishbone bus causes the master to run slower. This slowdown only occurs during read and write operations with the slave. All other instructions in the microcontroller are executed at full speed.

Multiple slaves

In practice, microcontroller circuits usually have more slaves than just a single I/O slave. All these slaves must communicate with the microcontroller via the same bus. This makes it necessary to add another piece of hardware that uses the address to determine which slave is being addressed.
In *ex14* the microcontroller is connected to two slaves. They are nearly the same as the slave used in the previous example. The address input has been omitted because there is only one write register and one read register. The slave also has eight inputs.

The job of the address decoder (*wishbone_decoder*) is to pass the signals to one of the two slaves depending on the address. We use two signals for this purpose (S1_SEL and S2_SEL), which go high when the right address appears on the Wishbone bus. The corresponding code for S1_SEL is:

```
S1_SEL<='1' WHEN ADR_I=x"8000"
ELSE '0';
```

In this case address 0x8000 was selected for slave 1.
A master–slave transaction can only occur when the CYC and STB signals are both high. It's easy to generate these signals now for slave 1:

```
S1_STB_O <= STB_I AND S1_SEL;
S1_CYC_O <= CYC_I AND S1_SEL;
```

The above lines of code ensure that the STB and CYC signals for slave 1 do not go high unless the slave is addressed. Finally, the data bus from the master to the slave has to be modified. If slave 1 is addressed, data must be sent from slave 1 to the master, and of course the same applies to slave 2. This is provided by the following line of code:

```
DAT_O_MASTER <= S1_DAT_I WHEN
(S1_SEL='1') ELSE
S2_DAT_I WHEN (S2_SEL='1')
ELSE x"00";
```

The same considerations apply to the ACK signal. It is passed on to the master in a similar manner.

Versatile

The handshake protocol makes the Wishbone bus very versatile. Besides the features already described, the bus can be extended with other signals such as an error signal, it can be configured so several masters can drive a single bus, and so on. If you want to know more about this, you can download the bus specification from the Opencores website.
There are also several other SoC busses. Most of them also use a handshake protocol, which makes it easy to implement a bridge between different bus systems.

Hierarchic VHDL

Up to now we have used graphic representations in our course to interconnect various blocks. However, you can also describe a complete design in Quartus using only VHDL.
We have prepared two examples to show how this can be done in VHDL. The first example (*ex15*) is a simple circuit consisting of two VHDL files and a graphic file. The graphic file is the 'top-level entity', which means it is the highest level in the hierarchy. The purpose of this file is to couple the subdesigns to each other and link the signals to the outside world (in other words, the FPGA pins). This is the method we have used up to now in all the examples. **Figure 3** shows this in schematic form.

The second example (*ex16*) contains the same design, but here the top-level document has been replaced by a VHDL file. The first statement in the *ex16.vhdl* file (see inset) is a standard ENTITY declaration. The inputs and outputs of this entity are ultimately connected to the pins of the FPGA, since this is our top-level document.
The input and output signals of the AND_2IN subdesign are described in lines 13-19. The signal names in this description must be the same as the names used in the AND_2IN.VHDL file. The same information must be provided for the OR_2IN subdesign.
Next we declare the signals used in this design. The signal names are the same as the names already used in example 15. In that example, these signals were drawn and labelled. In VHDL, this corresponds to signals of type STD_LOGIC.
A component with the name *inst1* is instanced in line 38. This reference is comparable to the designation 'IC1' or the like in a normal schematic diagram. The type of component to be placed here is described after the colon (:). In this case it is the component AND_2IN.
Finally, the inputs and outputs of this component are connected to signals starting with line 41.
If you compare the two examples, the principle involved will quickly become apparent.

Compatible

The advantage of describing a design entirely in VHDL is that the resulting source code is compatible with other CAD programs. Such a design can thus be used with the software of a different FPGA manufacturer without too much trouble. It's even possible to produce a real ASIC using exactly the same source code.

Another advantage is that it is often faster to modify a VHDL file than to make the corresponding changes in a graphic design, especially if there are a lot of signals between the various subdesigns.

(060025-5)

Web links

Opencores homepage:
www.opencores.org

Wishbone specification:
www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf

Listing ex16.vhdl

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ex16 IS
PORT
  (
    SWITCH1, SWITCH2, SWITCH3, SWITCH4 : IN STD_LOGIC;
    LED1, LED2, LED3 : OUT STD_LOGIC
  );
END ex16;

ARCHITECTURE arch OF ex16 IS
  COMPONENT AND_2IN
  PORT
    (
      A,B : IN STD_LOGIC;
      OUT_SIGNAL : OUT STD_LOGIC
    );
  END COMPONENT;

  COMPONENT OR_2IN
  PORT
    (
      A,B : IN STD_LOGIC;
      OUT_SIGNAL : OUT STD_LOGIC
    );
  END COMPONENT;

  SIGNAL IN1,IN2,IN3,IN4 : STD_LOGIC;
  SIGNAL AND1_OUT,AND2_OUT, OR_OUT : STD_LOGIC;

BEGIN
  IN1 <= SWITCH1;
  IN2 <= SWITCH2;
  IN3 <= SWITCH3;
  IN4 <= SWITCH4;

  inst1 : AND_2IN
  PORT MAP
  (
    A => IN1,
    B => IN2,
    OUT_SIGNAL => AND1_OUT
  );

  inst2 : AND_2IN
  PORT MAP
  (
    A => IN3,
    B => IN4,
    OUT_SIGNAL => AND2_OUT
  );

  inst3 : OR_2IN
  PORT MAP
  (
    A => AND1_OUT,
    B => AND2_OUT,
    OUT_SIGNAL => OR_OUT
  );

  LED1 <= AND1_OUT;
  LED2 <= OR_OUT;
  LED3 <= AND2_OUT;

END;
```