

## **Xilinx Design Tools: Foundation Series Version 3.1i**

The Field Programmable Gate Array (FPGA) design in this project will be undertaken using the Xilinx Foundation CAD package. This is a complex package which allows you to undertake schematic design of a circuit, simulate it, and then configure it so that it can be downloaded to a FPGA device. The Foundation software package contains some features that are not used in this project; therefore we attempt to provide you with enough information so that you can use the package sensibly, but not be overwhelmed by a mass of unnecessary detail.

You should work through this document at a computer having Xilinx Foundation installed, with a copy of the blue *XACT Libraries Guide*, which gives the actual schematic components you will use.

### **1. INTRODUCTION TO THE DESIGN FLOW, AND FOUNDATION 3.1i**

The design process adopted for this project, using Foundation 3.1i and leading to a configured FPGA, consists typically of the following steps:

1. Producing a circuit design; Foundation offers Schematic Capture, VHDL design, and State Machine design; we will use the first of these;
2. Functional Simulation of the design;
3. Implementing the design; this is the process of converting the electronic circuit design of stage 1 into configuration data which will be downloaded to the FPGA;
4. Downloading the design to the FPGA in the target hardware system and checking the design interfaces correctly with external hardware (eg. ADC, DAC & memory).

Stage 4 is then followed by system test. Both stages 2 and 4 are likely initially to lead to circuit design revisions.

## **Logging onto Foundation**

The instructions which follow assume that you are using one of the computers in the EIETL Lab. If you are working elsewhere the start-up procedure and location of files may be slightly different. To log on

1. Switch on the computer and monitor and allow the machine to boot up in Windows NT.
2. Login using the user name and password given to you by your project supervisor. If the "Domain" field in the login box is not already set to "EIETL2", drop down the list and select "EIETL2".

## **Logging Off**

Make sure you do this at the end of a design session, otherwise anyone that enters the Inglis Electrical lab has access to your files. Exit Project Manager, then click

**Start -> shut down -> Close all programs and log on as different user -> OK**

Note: Each person has their own logon. Only one person (PC) should be logged on using the same logon at once otherwise Foundation *Will* screw up your files. If you wish to transfer files between logons you can copy them to a floppy disk and then back again.

There is one computer allocated per pair though if there are spares you may use one each..

## **On-Line Help**

While this document will get you started on using Foundation, the on-line Help facility should be used to add detail. There are also on-line documents, distinct from the Help facility, available for background reading. To find these, click

**Start -> Programs -> Xilinx Foundation Series -> Online Document Viewer.**

Useful references, accessed through **Xilinx Books CD**, are **Foundation Series Quick Start Guide**, and **Foundation Series User Guide**. Please *do not* attempt to print these, apart from the occasional pages. You will block the printer for other users, and run the risk of giving yourself information overload. (Note however that Chapter 6 of the Quick Start Guide is included as an Appendix to this document).

## Project Manager

The Foundation Series uses the Project Manager to play a coordinating role in controlling the design process. You will be working on one project though this will contain more than one schematic. Schematics that are in the current project will be combined into the final FPGA.

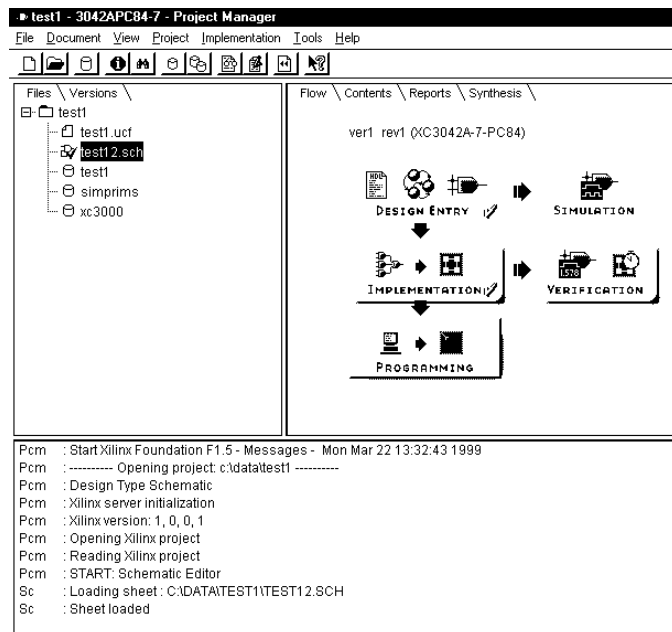
Having logged on, click on the Xilinx icon to launch the Foundation Series Project Manager. Click **Create New Project->OK**. You will now be asked to specify some project details. Make the project name unique by including your group number, it is this name that identifies your printout. Set Project Type to **Foundation series V3.1i**. Change the Xilinx family shown in the dialog box to XC3000A, the part to 3042APC84, and the speed to 7. This is the device type normally fitted on the Data Logger Project CDCC (Configurable Data Conversion Card).

The Project Manager window is broken into three sub windows.

The left hand window shows the files that are in the current project.

The right hand window shows the parts that make up the Foundation software, we will be using Design Entry, Simulation, and Implementation.

The lower window logs the commands that are run when you ask Foundation to perform an action. These can mostly be ignored, but do tell you if a problem has occurred (eg. in implementing your design).



It is possible to create, copy and delete projects by using the commands in the File menu, and transfer files to and from a project using the Document menu. Note that removing a file from a project does not delete it, you can add it again later by using the Add command in the Document menu.

There are several ways to use Foundation Projects in the Data Logger project. The thing to bear in mind is that when Foundation comes to implement the design, it attempts to configure the whole of the specified project. You can create more than one project if you wish or just add and remove schematics from the project (Document...Remove).

Your projects will be saved on H: drive, on the server. **Do not** change this drive letter or you will not be able to use your files when working on another PC.

## 2. SCHEMATIC DESIGN

Start the schematic editor from the Project Manager by clicking the AND gate symbol on the Design Entry button.

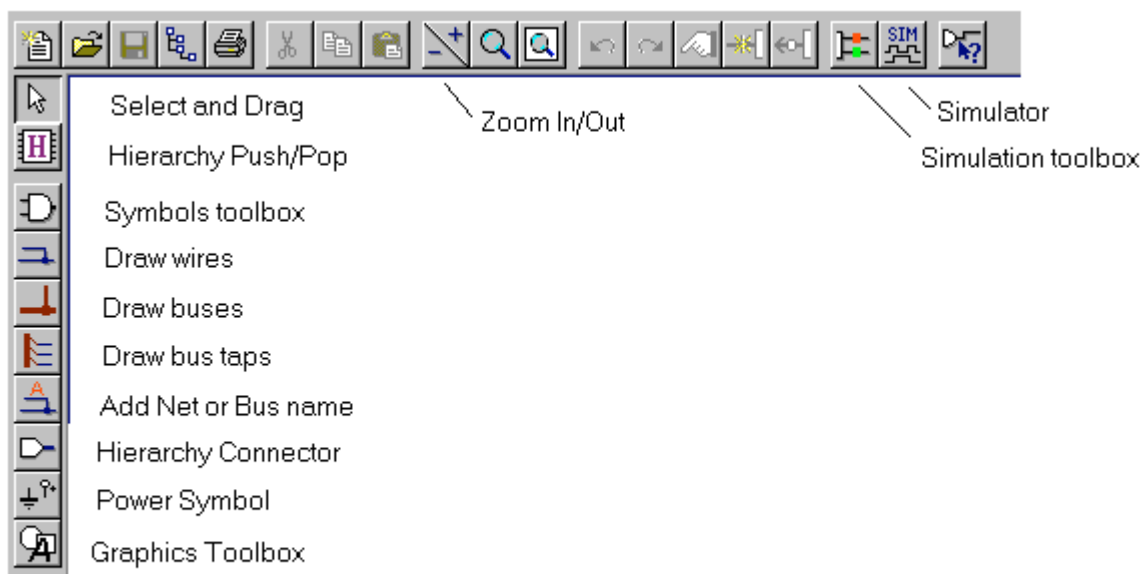


### Page Setup


Click **File -> Page Setup** and set your page size to A3 or A2. This should be appropriate for the scale of design you are likely to be doing. Most schematics fit into Landscape setting best.

### Familiarisation

The diagram below shows the Schematic Capture toolbars. The vertical bar allows the user to enter the main schematic capture operating modes; exit from these is achieved by the Esc key, or by returning to "Select and Drag". While in any mode a right click on the mouse generally brings up a menu showing options relevant to the mode.

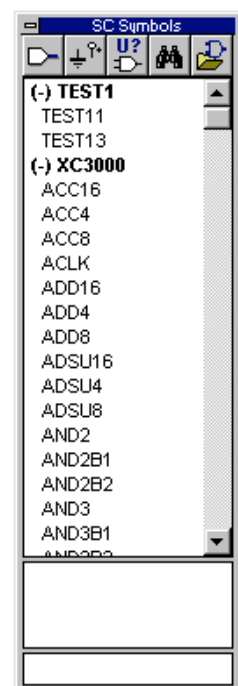


### Adding Components

Click on the Symbols Toolbox icon.  This brings up the Symbols

library window. The library contains all the schematic symbols which are listed in the blue libraries book. Have a look at the Device Summary on the inside front cover of the book, and familiarise yourself with the naming convention used. To add a component either select the item from the list or type its name in the dialog box at the bottom of the window.

Ground and  $V_{cc}$  connections are treated as components, and should be called from the library (**GND** and **VCC**). You can also add when placing a wire by right clicking to bring up a menu and selecting **Add PWR** (this has both power and ground on its pull-down menu; beware however that it allows you to put down a ground symbol labeled Vcc, if you don't enter the name correctly).



Components can be rotated by selecting them (see below), and entering **Ctrl R/L/M** (for right, left, mirror).

Repeated components are placed by clicking on the desired original component, and placing the one which is then displayed.

### Connecting Components - Wires and Buses

Foundation tends to use the terminology *Wire* for a single connection, and *Net* to describe a set of connected wires forming a single electrical node, but is not always consistent.

#### Drawing Wires

Click on the **Draw Wires**  icon .

Click on the starting point, and then on the end point; the wire will automatically be routed the best way. If you wish the wire to follow a particular path click at every corner you wish it to take. If you wish to leave a wire trailing at one end you can either start at the trailing end and finish at a component, or start where you like and right click and select **End Net**.


#### Labelling Wires

In some cases wire labelling is optional, in other situations it carries an essential function. To label wires double click on the wire and enter the name in the dialog box which appears. The name will be added about in the middle of the wire, you can later move this to any point on the wire.

Labelling is essential:

- if wires connect to a bus (see below);
- when connecting wires not otherwise joined; once a wire in a schematic is labelled, it is automatically connected to all other wires carrying the same label (called the *connect by name* facility), using this can greatly simplify the appearance of a schematic design;
- to identify nodes which are monitored in the simulation process.

#### Drawing Buses

Buses are groups of wires and are used in a similar way to wires. Wires can be connected into buses, and by adding appropriate labels, the connection is fully defined. To draw a bus, click on the **Draw Buses** icon,  and proceed as with drawing wires. With buses you are quite likely to want to leave one end floating, as you take off the wires one by one. To do this, right click on your furry friend. Among other things you will be offered **Add Bus Terminal** and **Add Bus End**. Choose the latter for self-contained schematics. The former acts as a *Hierarchy Connector*, discussed below under **Grouping a Circuit into a Macro Symbol**.

#### **Labelling Buses**

Buses are labelled using the same procedure as wires. Always label busses starting with the msb first ie A[10:0] not A[0:10] Example legal bus label formats are

|                |                                   |
|----------------|-----------------------------------|
| A[7:0]         | 8-bit bus, A7 (msb) to A0 (lsb)   |
| Q[7:0],SET,CLK | 10-bit bus, Q7 to Q0, SET and CLK |

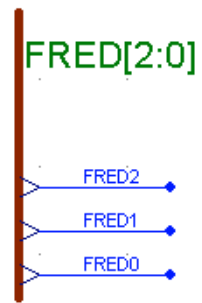
The dialog box offers *Simple Bus* (default) or *Complex Bus* format, which you must select. In the list above the first two are examples of simple bus, and the second of complex bus.

You **must** label all busses otherwise foundation will merge them all together.

### Tapping a Wire from a Bus

To connect a wire to a bus use the Draw Wire icon and add a label to the wire (don't bother with the bus taps icon).

Note that you are not allowed to connect bus lines together. This becomes a problem if you want to connect several to a common point, eg ground. In this case you will have to take each one individually through a buffer.



### Adding Attributes

In Foundation attributes can be used to define certain characteristics of circuit elements, for example how an element is configured in the FPGA for optimum performance. In simple and/or low-speed designs their use is not so important however. In this project you will probably only need to use one type of attribute, **LOC**.

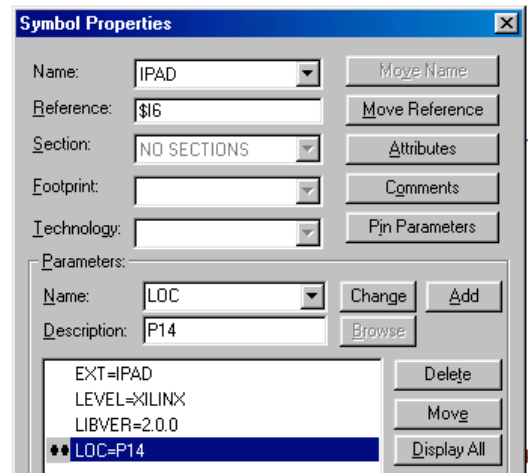
## The Location (LOC) Attribute

Connections into and out of the FPGA are made through special pads (eg **IPAD**, **OPAD** in the libraries book), and must be buffered (eg with **IBUF**, **OBUF**). For IPADs and OPADs the FPGA pin number to which they are connected must be specified; this is added as a **LOC** attribute.

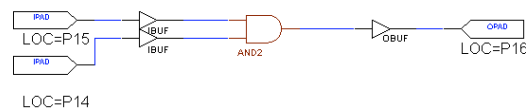
To add a Location to an IPAD or OPAD:

- double click the IPAD or OPAD,
- select LOC from the pull-down menu at **Name**,
- enter  $P_n$  in the Description (where  $n$  is the pin number), and click on **Add**.

$LOC=P_n$  should appear in the dialog box below, There should be two bullets to the left of this. If an item is shown with two bullets it will be displayed in full, if it is shown with one bullet it will be displayed in a shortened form, if there are no bullets then the item will not be displayed. To change the number of bullets double click on the item in the dialog box.



As with labels, attributes can be moved to any preferred position on the schematic. Below is an example of an AND gate that is connected to input and output pins.



## Editing the Title Block

A Title Block is automatically placed bottom right of the sheet. To edit this go to **File -> Table Setup**, and enter appropriate titles.

## Completing the Schematic Design: Design Rule Checking and Netlists


When you have completed the first iteration of schematic design, you will be going on to simulation or implementation. In either case Foundation will convert the schematic into a *netlist*. This is a listing of connections between the schematic components, and is given an **.alb** extension. In Foundation the netlist is normally created automatically at appropriate moments (ie as you enter simulation or implementation). The netlist can only be made if your design obeys certain design rules, so it is a useful check. To make use of this checking process, you can create a netlist whenever you want, by invoking **Options -> Create Netlist**. A more exhaustive design rule check is provided by the *Integrity Test*, invoked through **Options -> Integrity Test**. This claims to detect naming errors, bus errors, and hanging wires. *Neither of these tests is foolproof however, and of course neither guarantees a working circuit.*

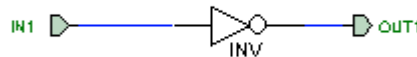
***Proceed to Section 4, Implementing Your Design, when working through this document for the first time.***

## Grouping a Circuit into a Macro Symbol

The design you are developing will inevitably be modular. The overall design ("top level design") will be prepared by the pair working together, and then each individual will contribute sub-sections of that design. These sub-sections can (should) be developed as independent blocks, which can be converted into symbols by Foundation, and then entered on the top-level design. This approach is fundamental to a Foundation-based design, in that many of the symbols that you call up from the library are actually created from logic primitives.

### Hierarchy Connectors

To create a Macro Symbol from a schematic, first add Hierarchy Connectors to all connections which will be inputs and outputs to the symbol. Do this by clicking the icon on the left hand toolbar.  In the dialog box which follows give the connection a name, and be sure to define it correctly as Input/Output etc (horrible problems with simulation or implementation can arise later with small errors made here). If the hierarchy connector is connected to a wire which is already labelled, then give it the same name. A very simple example appears below.



Hierarchy connectors for a bus are selected by clicking the above icon while in bus draw mode, *or* by right clicking on the rodent while laying down the bus, and selecting **Add Bus Terminal**.

### Creating and Editing the Symbol

To create the symbol select **Hierarchy -> Create Macro Symbol From Current Sheet**. Enter a descriptive name for the symbol, and leave other settings as default. Select Yes when asked if you wish to overwrite the schematic file. Once saved this symbol can be used in another schematic, by selecting it from the component library.

If you wish to edit the appearance of the symbol, eg its size, position of pins etc, place it on the sheet, left click on it and enter **Symbol Editor** via the **Options** menu (*or* right click on it again when selected, and choose **Symbol Editor** from the pop-up menu). Use the on-line Help for details on symbol editing.

To move between hierarchical levels, select **Hierarchy -> Hierarchy Push**, or **Hierarchy -> Hierarchy Pop** as appropriate, *or* use the Hierarchy button on the left toolbar. You can modify the schematic at any level. If you **Push** into a symbol from the top level and then modify it, you will be asked whether you wish to over-write the design as you **Pop** out of it. It is also interesting to **Push** into some of the larger Foundation symbols (eg multiplexers or counters).

### Creating a Top-Level Design with Empty Symbols

It is an apparent contradiction that you create symbols before their internal circuit has been designed, but you need to develop a top level design showing those symbols as a very first design stage. In fact, as symbols can be edited without difficulty, you can create them empty apart from their input/output connections, and enter them on the top level design. You can then add detail later. To create an empty symbol simply lay down hierarchy connections, with suitable names, on an empty sheet, and convert the sheet to a symbol. Then make up your top-level design using these symbols.



*When you draw your top-level design in Foundation, it is strongly recommended (for ease of later debugging) that you put all FPGA input/outputs at this level, rather than hiding them within macro symbols.*

### **3. FUNCTIONAL SIMULATION**

Functional simulation uses the circuit as you have designed it in your schematic, and does not take account of Xilinx characteristics. A copy of Chapter 6 of the Foundation Series Quick Start Guide from the Xilinx documentation is provided as an appendix to this document to guide you in using the Functional Simulator

#### **Some Additional Notes**

1. To start with, you are advised to define signals using "Probes" in the schematic capture window, and to add stimuli with "custom formulae" (yes, this does make sense when you read the guide!).
2. You can move directly between Schematic Capture and the Simulator using the buttons on the top toolbars, without returning to Project Manager. If you change the schematic however you should force a new netlist to be written, (from schematic editor Tools...Update simulation). You don't need to do this if you only change the probe selection.
3. When identifying signals as stimuli in large designs, do so with Probe at the top design level, and as far "upstream" as possible. This should ensure that the signal gets to all the places it's meant to. If you place the stimulus within a macro symbol it may not get to points "upstream", due for example to the presence of hierarchy connectors.
4. If it appears signals are just not getting through, then check and double check your schematic labelling. Many simulation malfunctions arise from labelling errors.
5. Reset simulation time to 0 with the "Power on" toolbar button, and clear old waveforms with the "Delete Waveforms" button.
6. When you simulate a schematic you simulate everything that is included in the project. Use Document..Add in Project Manager to remove any schematics that you do not want to simulate.
7. See appendix 2 for more details

#### **Timing Simulation**

After the design has been routed in the FPGA it is possible to perform a timing simulation, which includes timing delays based on the FPGA characteristics. This is called Verification in Xilinx. For the purposes of this project we do not do a timing simulation.

#### 4. IMPLEMENTING YOUR DESIGN

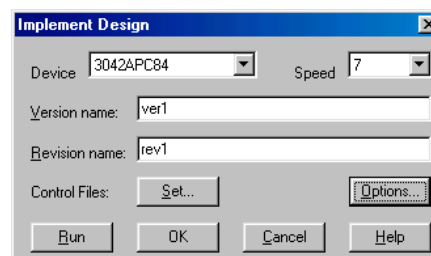
Once your design has been verified in terms of logic functionality, it is ready for design implementation. This is the process of converting the logic design into configuration data for the FPGA.

To implement the design click on the Implementation icon in Project Manager.

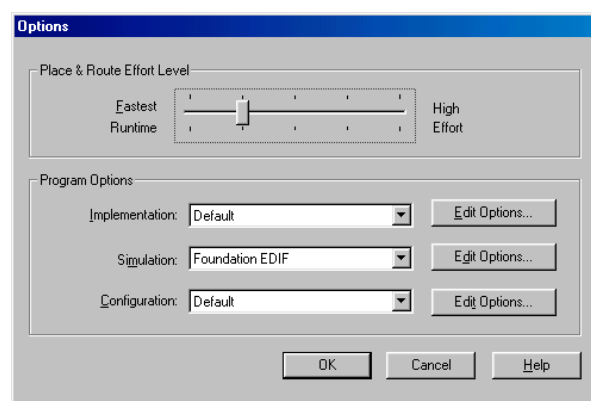


Foundation may ask you if you wish to update the netlist, click **Yes**.

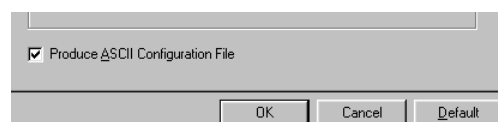
An Implement Design dialog box appears, most people will be using a 3042APC84 speed 7, check this corresponds with the chip that is on your CDCC FPGA board. If you do not need the old .rpt file then tick the **Overwrite current version** check box to save disc space.



Within this dialog box click on **Options**, and in the new dialog box click on **Edit Options** opposite the **Configuration** option. Another dialog box appears, which controls the configuration parameters of a device.

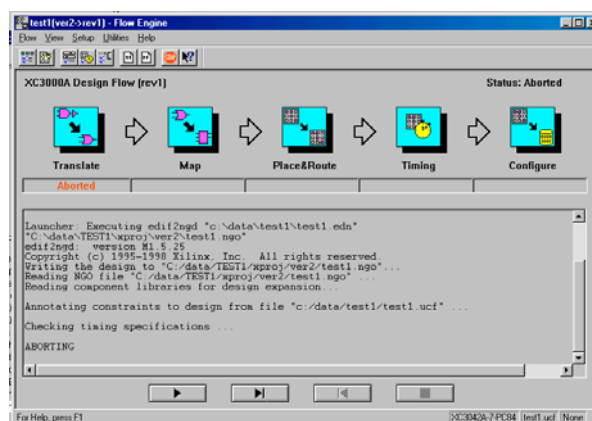


Make sure the **Produce ASCII Configuration File** check box is ticked. This ensures the .rpt file (raw bits) that you will download to the Xilinx chip will be produced. You should only need to do this the first time you run.



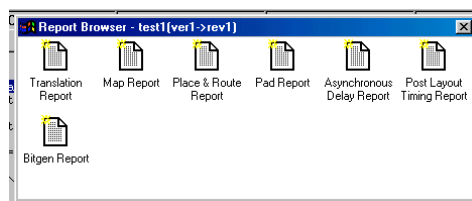
Click **OK** -> **OK** -> **Run** to clear all three dialog boxes and start implementation.

A **Flow Engine** window will appear and Xilinx will then implement your design, going through the stages of Translate, Map, Place & Route, Timing & configure. *This can take up to 1/2 hour for a large design.* Text reporting progress will appear in the bottom half of the screen. Check **Reports...Implementation log file** – if design has compiled correctly then the last line will start xcopy logger.rbt (where logger is the name of your project).



If you have any errors then your design cannot be implemented. If you have warnings your design has been implemented, but there is something that may need your attention. Some warnings can be ignored but you should always check these.

To get information about the implementation (eg for errors, warnings or CLB count) you can view reports by selecting the **Report** tag in Project Manager. Clicking on the **Implementation Report Files** will bring up the screen opposite. Check reports for the following information:



**Map Report:** for removed logic; don't worry when you see some logic removed, this is typically from Xilinx macro symbols that you have not fully utilised. Worry if everything seems to have been removed!

**Place and Route Report:** for the number of CLBs used.

**Pad Report:** for the I/O pins used.

For a successfully completed implementation there is considerable repetition between these reports. For an implementation which failed due to errors, the later reports may be missing. The message containing the error can usually be found in the last report present.

**Note** When you implement a design you implement every schematic that is included in the project. Use Document..Add in Project Manager to remove any schematics that you do not want to implement.

### **Viewing Layout in FPGA Chip**

To view the actual connection of the logic blocks start from the **Project Manager** window. From the top menu click **Tools -> Implementation -> FPGA Editor**.

You can check in a limited way how your chip has been configured. Check especially that the input/output pins that you expected are being used. Clicking the mouse pointer above a pin causes the pin number to be displayed at the bottom of the screen. Pan around by holding the right mouse button down while moving it around. Xilinx offers editing capability at this level, but for this project we do not attempt to use it.

### **If your implementation has errors**

Click on the Reports tab in project manager. Foundation usually stops implementation when it finds an error so the error should be in the last file (reading left to right). Double click on this report and use the search menu to search for the word error. If this does not give you joy then try searching the other report & log files and also searching for the word warning.

### **Searching for errors in schematic**

If a log or report file shows an error with a specific net/bus/component you can find it by selecting Mode..Query from the schematic editor. Either click on a net/bus/component to find its details or type the details in the box to get it to find the net/bus/component.

## **5. DOWNLOADING CONFIGURATION DATA INTO THE FPGA**

Downloading is done by connecting a special cable from the PC parallel port to the CDCC. A dedicated computer has been set up for this purpose, and you will need to bring your CDCC, and configuration file on floppy disc, to that computer for download.

A communications program called SEND, running under DOS, allows the CDCC to be programmed from the PC. The .rbt file (“rawbits”) is the one that is downloaded. To achieve a SEND you will need to do the following:

- In Windows Explorer go to H:\*projectname*, and find the .rbt file,
- Copy the .rbt file to the floppy,
- Put the floppy into the Configuration PC, connect the CDCC and ensure it is powered,
- At the DOS prompt type **send store a:*projectname*.rbt lpt1**

This saves the configuration data into the on-board EEPROM, and then configures the FPGA. You can also eter

**Send direct a:*projectame*.rbt.**

This sends the configuration data directly to the FPGA. It will however be lost when the CDCC is powered down.

To display the Help screen for this communications program, enter **SEND** at the command line.

*Note: On Send Store the program occasionally and incorrectly reports an error at the Configure FPGA stage. If all has gone well to here, then ignore the error message.*

**RT/TJW April, 1999**

**RT 11/4/00**

**RT 18/5/01**

**RT 07/4/02**

## Appendix 1: Designing with Xilinx

Note: It is useful to have the blue XACT Libraries Guide book with you as you read this document, as a number of abbreviations used below, eg OPAD, OBUF etc, are names of logic symbols found in it.

### 1. Features of the FPGA

#### CLB's, IOB's, Routing -- What's in the FPGA?

As described in the Xilinx XC3000 FPGA Data Booklet, there are 3 main resources in the Xilinx FPGA chip:

**Configurable Logic Blocks (CLBs).** Each CLB contains 2 D-type flip-flops and a configurable logic circuit, together with the necessary multiplexer switches to link them together. The function performed by a block is fixed when the chip is configured, and can't be changed by your logic circuit. It can (of course) be changed by reconfiguring the chip.

**Input Output Blocks (IOBs).** These units link your circuit (built up from CLBs) to the outside world. Each one contains a 3-state output buffer, an input buffer, and 2 D-type flip-flops, one of which is used for output (the logic symbol OFD) and one for input (IFD).

**Routing Resources.** These form the "wires" that link the CLBs and IOBs together.

The only combinational logic in the device is in the CLBs, and consists of a circuit that can be configured either as a single function with 5 inputs, or as a pair of functions of the *same* 4 inputs. (OK, this is a simplification; if you know exactly what circuits can be fitted into a single CLB, then you should be writing this.) A function can be a lot more complex than an AND gate, though -- if you took 4 signals, ANDed them in pairs, ORed the result, and then XOR'ed the result with a fifth signal, that counts as a single function of 5 inputs.

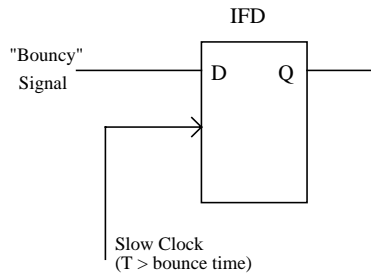
It is possible to run out of combinational logic (or more precisely the combinational parts of CLBs) and it's possible to make a circuit that's so complex that it is impossible to route, particularly if you have several large buses. Since there are, in general, plenty of D-type flip-flops, you should use these in preference to combinational logic whenever possible. If you need for example to serialise a data word, use a shift register, not a counter and multiplexer.

#### Linking up to the outside world

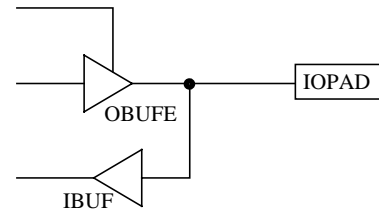
In order to link up your internal Xilinx design to the package pins you need to use the IOBs, entering them explicitly on the schematic. There are 3 main cases to consider.

Debouncing Inputs. It is possible to use the IFD to debounce switch inputs, as shown in the diagram below left.

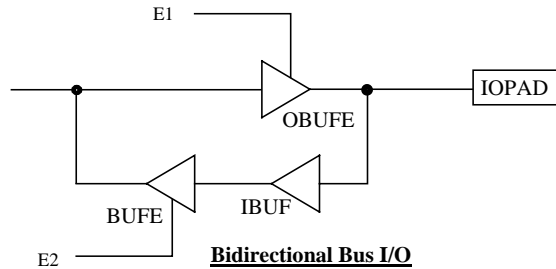
Bidirectional Lines You need these when you want to connect, say, to the bidirectional data pins of a memory chip. The physical connection is represented by the IOPAD symbol, and 2 possible configurations are shown below. You still need to design the logic to control the 3-state enable line(s).



**Using the IFD for Synchronisation and Debouncing**



**Bidirectional I/O**



**Bidirectional Bus I/O**

### Connecting to the right pins

Normally, you don't care how the logic is positioned within the Xilinx chip -- all the CLBs are equivalent. But, of course, IOBs correspond to physical pins on the package (ignoring "unbonded" IOBs here) and these pins are connected to the external circuitry. Therefore you must position Input/Output blocks yourself. You do this by Adding an Attribute to the pad. The format is given in the "Adding Attributes" section of this document.

You can only have 1 pad located at a given pin. If you want to have a bidirection I/O buffer as described above, you *must* use an IOPAD symbol, and not a separate IPAD and OPAD, even if these are located in different blocks of the schematic.

The correspondence between the IOB's on the silicon die and the pins on the package is dependent on the type of package, and you must make sure that you compile the circuit not only for the right type of device but also for the right package type.

### Clock Nets -- GCLK and ACLK

As well as the general routing resources described above, there are 2 special routing nets that can only be used for clocks. The first is the **global clock**, driven by the GCLK buffer, and is used by simply linking the appropriate clock inputs to the output of the GCLK buffer symbol.

The output of the GCLK buffer can only drive CLB flip-flop clock inputs, or inverters that drive CLB flip-flop clock inputs. (Strictly these inverters are not CLB combinational functions - they're implemented by a separate gate on the flip-flop clock input). It can't drive any combinational functions, and can't be gated.

The input of the GCLK buffer can be driven either from any normal logic signal, or from a special IPAD called TCLKIN. This IPAD is directly linked to the input of the GCLK buffer without using an IBUF. In this project, the main 32kHz clock is externally connected to TCLKIN. You use a LOC attribute to correctly place this IPAD.

The other net is the **alternate clock**, and is driven by the ACLK buffer. The output of this buffer has the same restrictions as those for the GCLK buffer, while the input can be driven from any normal signal (Yes, there are other possibilities in the data book, but none of them are useful for this project). The routing resources used by the ACLK net may be used for other purposes if this net is not in use, so it may be best to avoid using it.

The main advantage of these clock nets is the lack of skew. There are significant delays in the routing of signal across the Xilinx chip, so that 2 flip-flops clocked from the same signal routed on normal nets may not change at the same time. This may cause problems in complex circuitry. The GCLK net avoids this problem, and flip-flops clocked by it all change at the same time (at least to within the typical propagation delay of a CLB). *You can only use one each of GCLK and ACLK in a single FPGA, so in shared designs you must determine where these components are placed.*

### Ground and Power Supply Connections

Ground and  $V_{CC}$  connections are treated as components, and should be called from the XC3000 library (GND and VCC).

## 2. Some Tips on Circuit Design with FPGAs

### If possible make it synchronous

One of the biggest problems with designing for the Xilinx chip is that the switching time of a gate or flip-flop may be less than the time it takes for a signal to go across the chip (through various routing switches), or the difference in propagation delays from 2 inputs on a single CLB to its output.

This means that circuits that would appear to be glitch-free may, in fact, fail to work when implemented in a Xilinx device. A typical example is that (when using normal discrete chips), the outputs of a synchronous counter all change at the same time, and a simple AND decode of these outputs is glitch-free (at least compared to the switching time of most logic families). In a Xilinx, this is not the case. If you make this circuit, you *may* find glitches on the output of that AND gate. And if you then use that output to trigger a flip-flop, it is likely to mistrigger.

Notice that we said "may" and "likely". The width of these glitches is dependent on how the circuit is placed within the chip, and how the signals are routed. In general, you don't control

that yourself, you let the software do it. So you may have a circuit that appears to work until you add some more, totally unrelated, logic to it. Then the compiler places the circuit in a different way, and the glitches become noticeable.

To avoid these problems, you should try and make a synchronous design. Clock all the flip-flops off a single, ungated, clock (preferably GCLK). Control the Enable and D inputs to the flip-flops with your logic - glitches here, especially if far from the active edge of the clock, cause few problems.

Needless to say you should avoid asynchronous counters unless you simply want to divide down a clock by a power of 2. An Asynchronous counter seems to use a few less CLB's than a synchronous counter the same length, although the savings are not that sizeable (about 2 CLB's on a 16 bit counter, typically), so you should consider using one only if you are very short of CLB's.

#### Use the clock enable inputs

Each CLB flip-flop (Although not the IOB flip-flops, alas) in the Xilinx chip has a clock enable input. This is *sampled* on the active edge of the clock, and glitches away from the clock edge (no matter what the level of the clock input) have no effect. Whether this is the rising or falling edge is selected by a multiplexer in the CLB (it's shown in the XC3000 data booklet), and it is set when the chip is configured.

It's therefore a lot safer than ANDing a clock signal with a possibly glitchy enable signal. If you do that, glitches on the enable signal while the clock is high may cause false clock signals to be given to the flip-flop, which will cause your circuit to malfunction.

#### Use Synchronous Resets

Most counters are available in 2 versions, one with an asynchronous clear input (this signal clears the counter irrespective of the clock input), and the other with a synchronous reset input (it is examined on the active edge of the clock, and clears the counter on that edge if it is asserted). If possible use the latter one. Suppose you want to design a counter that counts from 0 to 5 and then back to 0. There are 2 ways to do this : The wrong way is to use an asynchronous clear, and to clear the counter when the state 110 (=6) is reached. The other is to use a synchronous reset and to assert this reset signal when the counter is set to 101 (=5).

There are 2 problems with the first approach. The first is the obvious one - the "runt" state (of 110) may exist for long enough to cause problems. The second problem is more subtle. The state may not exist for long enough - that is long enough to clear the counter. As soon as the counter state moves from 110, the clear signal is deasserted. This could occur after only one of the flip-flops has been cleared, and the other one may not see a long enough clear signal to be set to 0. In other words the counter may reset to 100 (=4) or 010 (=2) instead of the required 000.

Again, this problem may be layout dependent. The circuit may work fine until you add something else. It's therefore best to avoid it totally.



### Don't depend on gate delays

If you have ever designed a circuit using TTL chips, you may be aware of the trick of using a couple of inverters to delay a signal by a few nanoseconds, for example to ensure a clock pulse arrives after a reset.

Don't try to do this on a Xilinx circuit -- it won't work for several reasons. Firstly, the individual gates don't exist as they appear on the schematic - they are combined into the Xilinx CLB's. Secondly, the logic optimiser will remove "unnecessary" inverters -- including pairs of them linked in a chain. And thirdly, the switching time of a CLB is comparable to the time it takes for a signal to be sent across the chip through the routing switches.

If you need a delay, use a D-type flip-flop clocked from a suitable fast clock.

### Tie unused "Enables" to the correct state

Most Xilinx enables on the macro circuits used are active high. Make sure you tie them to the correct state, or you may find that large sections of your logic are "optimised out" by the compiler.

### Design complex combinational circuits using multiplexers

You may already know that you can use a  $2^n$ -input multiplexer to make an  $n$  input combinatorial circuit. You simply connect the  $n$  inputs to the select lines of the multiplexer and tie the data inputs high or low according to the truth table.

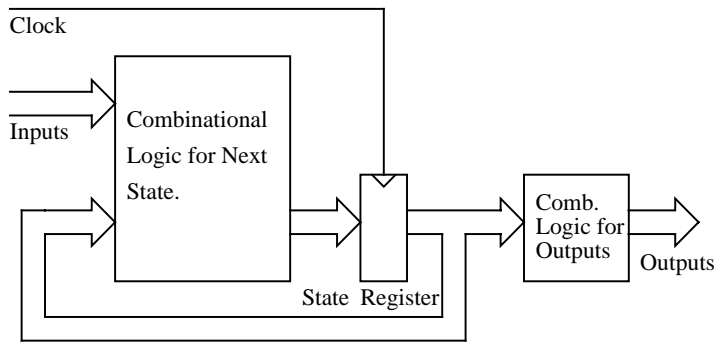
Although in general a multiplexer is a large combinational circuit with many inputs, and thus does not fit nicely into the Xilinx CLB's, a multiplexer with the inputs tied high or low is reduced by the logic compiler into a simple combinational circuit of  $n$  inputs, and will fit into a single CLB, at least if  $n$  is 5 or less. It's therefore just as efficient in Xilinx resources to do this, rather than to optimise the logic by hand (e.g. by using Karnaugh Maps or Quine-McCluskey reduction), and it's a lot quicker to do. The logic optimiser will reduce your logic to the minimal form, and implement it for you. (It's a pity we have to go through these shenanigans. The CLB logic is implemented as a RAM-based look-up table - in other words a multiplexer fed with the truth table. But there's no easy way to set up a CLB to a given truth table).

### Designing State Machines

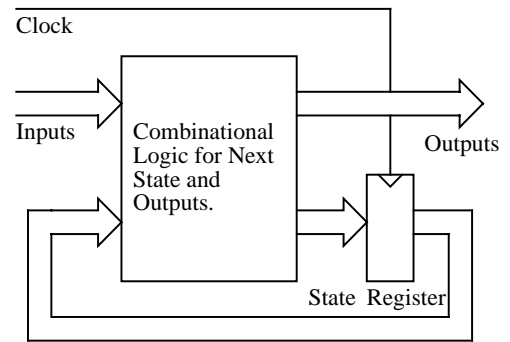
A State Machine is a sequential logic circuit which produces a sequence of output states which depend both on the previous state, and on the value of input signals.

A simple counter is one type of state machine, except that it has no dependence on any input. An up-down counter, whose count direction depends on an "up-down" input, is a better example of a general-case state machine.

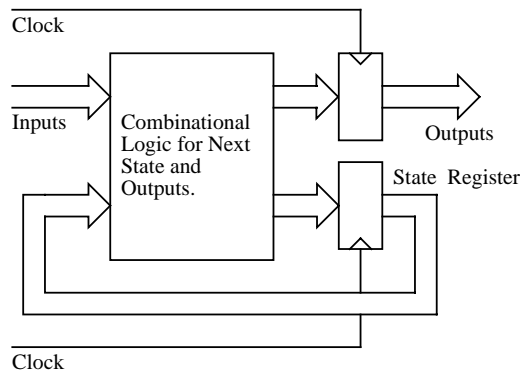
Several models of machine exist, in particular Moore and Mealy. You may also come across the "One Hot" state machine. This can be particularly useful in some FPGA applications, but we will not introduce it for this project.



**Moore Model Block Diagram**



**Mealy Model Block Diagram**



**Synchronous Mealy Model Block Diagram**

Many of the state machines in this project will be required to produce a simple sequence on receipt of a trigger input.

This type of circuit fits rather neatly into the Xilinx CLB structure using the D-type flip-flops to store the current state, and the combinational blocks for the feedback logic. Flip-flops can in principal be J-K or D-type, but the Xilinx CLB structure favours a D-type solution.

A procedure for designing state machines is as follows:

1. Determine which model you wish to use.
2. Work out how many flip-flops you need. For a simple Moore configuration,  $n$  flip-flops gives you  $2^n$  states.
3. Draw the state transition diagram (which states can come from each states, and under what conditions). Label the states with the values of all the flip-flop outputs.
4. Work out the truth table for the D-input of each flip-flop, based on the output of all the flip-flops and the external inputs.
5. Design the combinational logic & implement the logic using multiplexers. Dont bother simplifying logic using Karnaugh maps or Quine-McCluskey reduction on the truth table, the logic optimiser will do this hard work for you.
6. Enter the logic into the schematic capture program.

We will do an example.

### Debugging the old-fashioned way

For complex circuits, it's often easier to debug the design on the real hardware, rather than setting up and using the simulator. One problem is that you can't connect your logic analyser, oscilloscope, or whatever to internal Xilinx signals, and you may want to check their state. The way round that is to add an extra OBUF and OPAD to your design, connected to the signal that you want to monitor. LOCate the OPAD on one of the pins that are connected to the test points on the Data Logger board, and monitor that testpoint with your instruments

### It doesn't work!

It is extremely rare for a Xilinx design to work first time, and in many cases the reason for failure is simple. Here are a few things to check - we would appreciate receiving others to include in this section.

- Make sure that you have positioned I/O blocks on the right pins. Check the attributes. Use the Epic Design Viewer (as described earlier in this document) to check that the pins are, in fact, in the right place.
- Check there are no labeling errors.
- Read the Log File from the compilation. If large sections of logic have been removed because they are "disabled", find out why. You may have tied an enable signal to the wrong state.
- Check that you have correctly tied all enable signals, resets, clears, etc.
- Make sure you've not mixed up the MSB and LSB of buses, counter outputs, multiplexer select inputs, etc
- Check very carefully that all clock signals (and those to external circuitry) are glitch-free. Do not gate clocks with glitchy signals unless you are *sure* you know what you are doing.
- Check your state machine logic. Make sure the machine can't get into a state (even at power-up) from which it can't get into one of the states in the normal sequence.
- Sometimes (and for no apparent reason) labels or attributes are ignored by the compiler. If you are getting "strange" results, such as the Epic Design Viewer showing no signal going to a pin that you know you've assigned, then it sometimes helps to delete the label or attribute and retype it.
- Like all other chips, the Xilinx FPGA needs a stable power supply. Check that you've got 5V at the board and at the pins of the chip, particularly if you're using a supply of your own design.
- If all else fails, ask a demonstrator. Don't spend a long time worrying about a particular problem.

**ARD/TJW**