CHAPTER FOUR

XC6200 - RPU

The reconfigurable co-processor selected for the H.O.T. Works Board is the new Xilinx Xc6200 family of RPUs.

The H.O.T. Works Development System represents a major step in breaking the barriers between hardware and software; offering Hardware-On-Demand™ . This was made possible through the integration of the XC6200 RPU in the H.O.T. Works System.. This chapter discusses the XC6200 Family.

CONTENTS

BLANK PAGE

The XC6200 RPU

Overview

The Reconfigurable Processing Unit (RPU) is a new Programmable Logic Device from Xilinx Inc. for Run-Time Reconfigurable Computing. Programmable logic devices, particularly FPGAs, continue to gain momentum over traditional ASICs as the logic solution of choice for today's systems design. These programmable logic devices (PLDs) are viewed as appropriate solutions for prototyping or low to medium volume production. The FPGA market is driven by the need to find a replacement for the ASIC in the product development cycle. In these applications, the function of the FPGA is more ASIC-like than general-purpose processing.

In the last five years, research has shown that reconfigurable logic devices have a great potential in algorithm acceleration within a computing system environment. The multiple run-time reconfiguration use of PLDs is called Configurable or Reconfigurable Computing. Applications such as data mining, image/signal processing and program acceleration, require 'in-system run-time' reconfigurablity, making new demands upon the programmable logic device. While there are many similarities between the architectures of the FPGA and RPU, their differences are significant and should be taken into consideration when evaluating the application at hand.

Features of an RPU

The RPU is a natural evolution of the very successful Programmable Logic Device Class and as such should be familar to users of FPGAs. The primary difference of the RPU and FPGA come from the added features needed for reconfigurable computing. The RPU needed to integrate with the computer's microprocessing unit, operating system and the standard software language applications being used in the marketplace.

There are three unique features in an RPU:

- Open Architecture

  Resulting in 3rd development of tools and compilers

- Dynamically & Partically Reconfigurable Logic

  Enabling Hardware On Demand.

- A Microprocessor Interface

  Configured times in micro seconds

The RPU is a combination of reconfigurable logic cells and an efficient interface to the microprocessor. The RPU can be used for ASIC Replacement or Custom Computing applications. Figure 1 compares some of the features off the RPU with other devices.

	MPU	DSP	RPU	FPGA	ASIC
Performance	Limited	Better than MPU	Better than DSP	Near ASIC	Very Fast
Architecture	OPEN	OPEN	OPEN	CLOSED	CLOSED
Programmable	Easily	Programmable	Programmable	Difficult	Not
Development Tools	Lots Low Cost	Good Low Cost	First Low Cost	High Cost	Very High Cost
Who Programs	Software Engineer	Software Engineer	Software Engineer	Hardware Engineer	Hardware Engineer
Use	General Purpose	Embedded	Embedded & Gen. Purpose	Embedded	Embedded
Applications	Lots End Users	Lots Embedded	Embedded & End User	Lots Embedded	High Volume Low Cost

PROGRAMMABLILTY ← → POWER

Figure 1

XC6200 Family

The XC6200 family is a new type of high performance RPU from Xilinx Inc., the makers of Field Programmable Logic Devices.

XC6200 is a family of fine-grain, sea-of-gates FPGAs. These devices are designed to operate in close co-opera-tion with a microprocessor or microcontroller to provide an implementation of functions normally placed on an ASIC. These include interfaces to external hardware and periph-erals, glue logic and custom coprocessors, including bit-level and systolic operations unsuited to standard processors.

An XC6200 part is composed of a large array of simple, configurable cells. Each basic cell contains a computation unit capable of simultaneously implementing one of a set of logic level functions and a routing area through which inter-cell communication can take place. The structure is simple,

symmetrical, hierarchical and regular, allowing novice users to quickly make efficient use of the resources available.

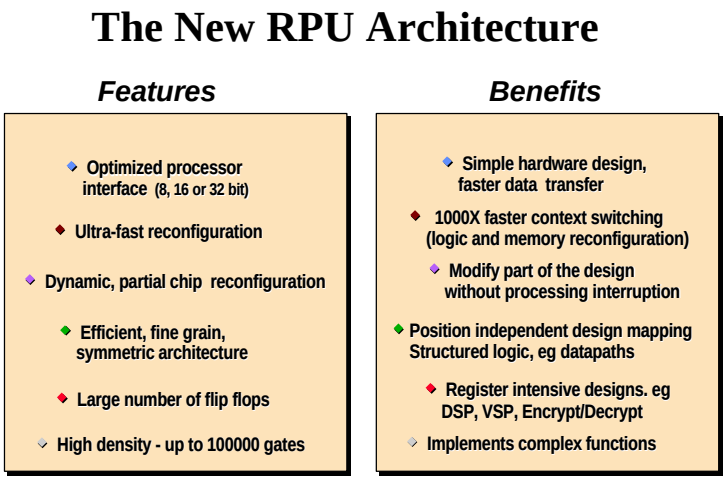

Figure 2

The nearest-neighbor interconnect of the underlying cells is supplemented with wires of length 4 cells, 16 cells and Chip-Length, which provide low delay paths for longer connections. In addition there are four global input signals which provide a low skew distribution path for critical high fan-out nets such as clocks and initialization signals.

XC6200 parts are configured by an integral, highly stable six-transistor SRAM control store. This allows XC6200 parts to be quickly reconfigured an unlimited number of times. The SRAM control store can be mapped into the address space of a host processor and additional support logic is provided to allow rapid reconfiguration of all or part of the device. In addition, the outputs of function units within the device can be read by a processor through the FastMAP™. Processors can read or write registers within logic implemented on the device. Data transfers can be 8, 16 or 32 bits wide, even when register bits are distributed over a column of cells. These capabilities allow XC6200 RPUs to support virtual hardware in which circuits running on the RPU can be saved (`swapped out') to allow the RPU resources to be assigned to a different task, then restored (`swapped in') at a later time with the same internal state in their registers. Sections of the device can be reconfigured without disturbing circuits running in other sections. Thus an XC6200 in a coprocessor application can be shared by several processes running on the host computer.

Architecture

The XC6200 architecture may be viewed as a hierarchy. At the lowest level of the hierarchy lies a large array of simple cells (Figure 3). This is the 'sea of gates'. Each cell is individually programmable to implement a D-type register and a logic function such as a multiplexer or gate. Any cell may also be configured to implement a purely combinatorial function, with no register.

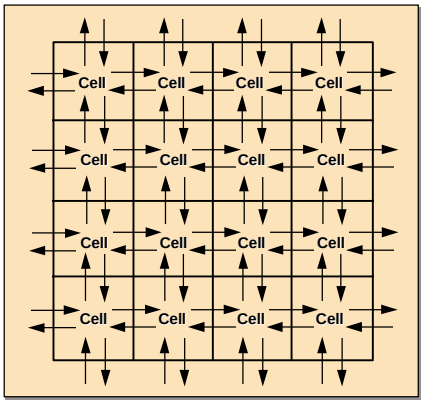

Figure 3

First generation fine-grain architectures implemented only nearest-neighbor interconnection and had no hierarchical routing (Figure3). XC6200 is a second generation fine-grain architecture, employing a hierarchical cellular array structure. Neighbor connected cells are grouped into blocks of 4x4 cells (Figure 4) which themselves form a cellular array, communicating with neighboring 4x4 cell blocks.

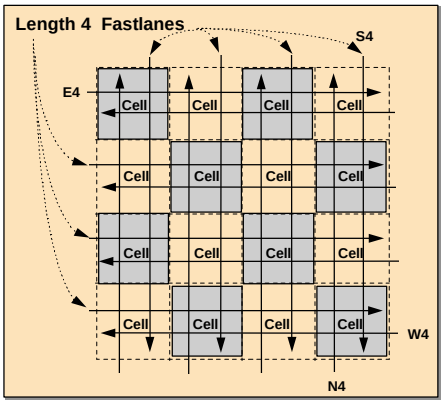

Figure 4

A 4x4 array of these 4x4 blocks forms a 16x16 block (Figure 5).

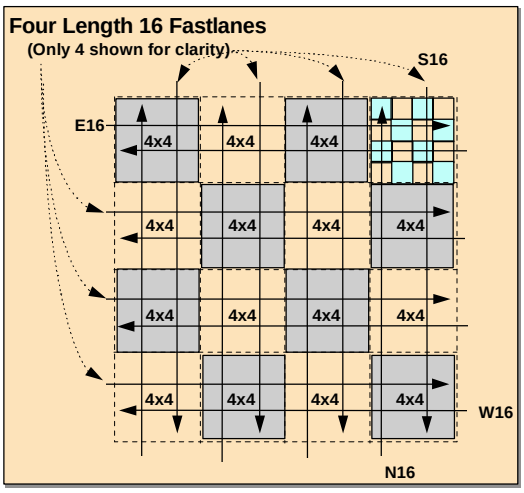

Figure 5

In the XC6216 part, a 4x4 array of these 16x16 blocks forms the central 64x64 cell array which is then surrounded by I/O pads (Figure 6).

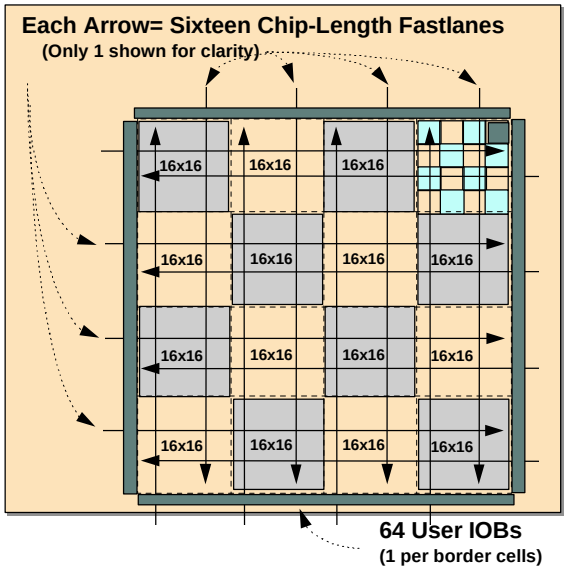

Figure 6

Routing Resources

Each level of hierarchy (unit cells, 4x4 cell blocks, 16x16 cell blocks, 64x64, etc.) has its own associated routing resources. Basic cells can route across themselves to connect to their nearest neighbors and thus provide wires of length 1 cell. Note that cells used for interconnect in this manner can still be used to provide a logic function. Wires of length four cells are provided to allow 4x4 cell blocks to route across themselves without using unit cell resources. Similarly 16x16 cell blocks provide additional wires of length 16 cells and the 64x64 array provides Chip-Length wires. Larger XC6200 products extend this process to 256x256 cell blocks and so on, scaling by a factor of 4 at each hierarchical level as required. Intermediate array sizes (e.g. 96x96) are created by adding more 16x16 blocks. Switches at the edge of the blocks provide for nections between the various levels of interconnect at the same position in the array (e.g. connecting length 4 wires to neighbor wires).

The longer wires provided at each hierarchical level are termed 'FastLANEs™'. it is convenient to visualize the structure in three dimensions with routing at each hierar- chical level being conceptually above that in lower hierar- chical levels, with the cellular array as the base layer. The length-4 FastLANEs are driven by special routing multi-plexers within the cells at 4x4 block boundaries. All routing wires are directional. They are always labeled according to the signal travel direction. For example, S4 is a length-4 FastLANE heading from North to South. In Figures 2, 3 and 4 each individual cell has a length 4, 16 and Chip- Length FastLANE™ above it. However only a small number are shown for clarity.

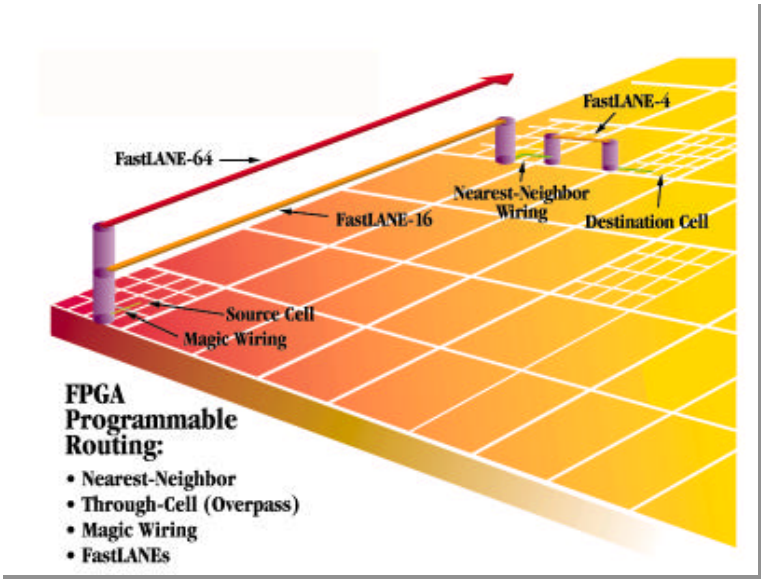

Figure 7

The benefit of the additional wiring resources provided at each level of the hierarchy is that wiring delays in the XC6200 architecture scale logarithmically with distance in cell units rather than linearly as is the case with the first generation neighbor interconnect architectures. Since 4x4 cell block boundaries lie on unit cell boundaries, the con-switching function provided at 4x4 cell boundaries is a superset of that provided at unit cell boundaries; i.e. it provides for neighbor interconnect between the adjacent cells as well as additional switching options using the length 4 wires. Similarly, the switching unit on 16x16 cell block boundaries provides a superset of the permutations available from that on the 4x4 cell block boundaries. Further switching units are also provided on the 64x64 cell boundaries to provide the Chip-Length FastLANEs.

Additional Routing Resources

Magic Wires

The majority of interconnections are routed using the nearest-neighbor and FastLANEs described above. Each cell has a further output (labeled 'Magic') which provides an additional routing resource. A cell's Magic output is not always available for routing. Its availability is dependent on the logic function implemented inside the cell. More information on the physical nature of the Magic wires is given in the section "Function Unit".

Each cell's Magic output is routed to two distinct 4x4 block boundary switches. The Magic wire can be driven by N, S, E or W from adjacent cells or from the N4, S4, E4 or W4 FastLANEs passing

over the cell. This makes it particularly useful for corner-turning (all other routing resources are straight). The Magic wires are illustrated inFigure 8.

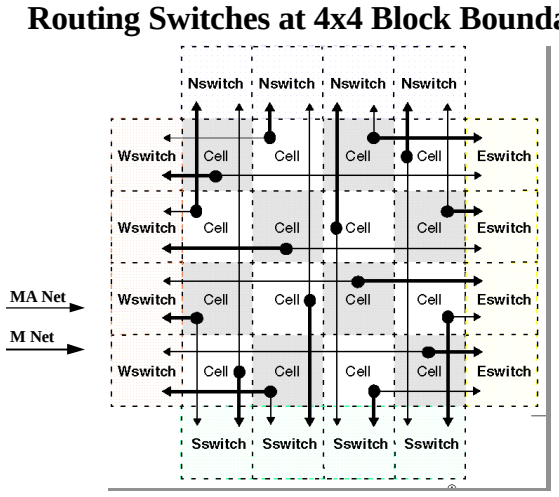

Figure 8

Global Wires

The XC6200 architecture permits registers within a user design to be clocked by different clocks and cleared by different asynchronous clears. Clocks and Clears may be provided by any user I/O pin or generated from user logic internally. In line with good synchronous digital design practices, it is recommended that a single global Clock and Clear are used. This minimizes the likelihood of timing problems and gives more reliable simulations.

Four Global wires (G1, G2, GClk and GClr) are provided for low skew, low delay signals. These wires are intended for global Clock and Clear or other high fan-out signals and are distributed throughout the array in a low skew pattern. A global signal can reach the clock and clear inputs of any cell on the array passing through very few routing switches. The four Globals are very similar. It would be possible to use GClk as a global Clear signal, however for minimum delay, it is recommended that GClk be used for global clocks and GClr for global clears. GClk and GClr can reach the inputs of any register in the array, passing through only a single routing switch. G1 and G2 may be used for secondary global clocks or clears. G1 and G2 have a slightly larger delay than GClk and Gclr.

Ample Routing Resources

- **FastMAP busses**
  - **underlying bus structure used for FastMAP interface access**
  - **independent from normal FPGA routing**
- **Global Routing**
  - **4 global resources (GCLK, GCLR, G1, G2)**
  - **low skew/low delay signals**
- **Nearest Neighbor**
  - **high-speed direct connection from each function cell to the 4 adjacent cells**
- **Overpass (through-cell)**
  - **routes signal from one function cell to another through an intermediate cell**
  - **does NOT interfere with or sacrifice logic capability of intermediate cell**
- **Magic Wiring**
  - **special routing optimized to provide high-performance bus-turning**
- **FastLANEs (FastLANE4, FastLANE16, FastLANE64)**
  - **large number of short-delay "longlines"**

Function Unit

Figure 9 shows the basic XC6200 cell in detail. The inputs from neighboring cells are labeled N, S, E, W and those from length 4 wires N4, S4, E4, W4 according to their signal direction. Additional inputs include Clock and Asynchronous Clear for the Function Unit D-type register. The output from the cell function unit, which implements the gates and registers required by the user's design, is labeled F. The Magic output is used for routing as described earlier. The multiplexers within the cell are controlled by bits within the configuration memory.

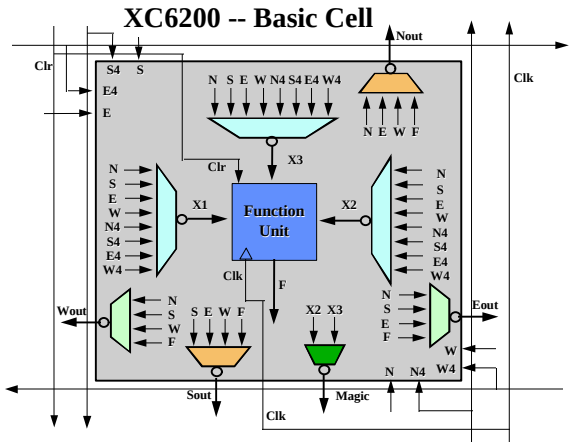

Figure 9

As can be seen from Figure 9, the basic cells in the array have inputs from the length 4 wires associated with 4x4 cell blocks as well as their nearest neighbor cells. Figure 10 shows the Function Unit Routing. schematic representations of the basic cell functions possible.

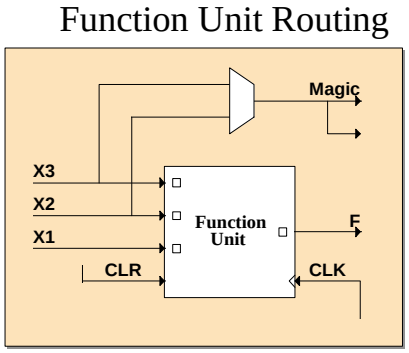

Figure 10

The function unit design allows the cells to efficiently support D-type registers with Asynchronous Clear and 2:1 multi-plexers, as well as all Boolean functions of two variables (A and B) chosen from the inputs to the cell (N,S,E,W,N4,S4,E4,W4) (Figure 11 - Table 1).

Function Derivation Table

Function	X1	X2	X3	Y2	Y3	RP	CS	Q
0	A	A	A	X2	$\overline{X3}$	X	C	X
1	A	A	A	$\overline{X2}$	X3	X	C	X
BUF (Fast)	A	X	X	Q	$\overline{Q}$	Q	C	0
BUF	X	A	A	$\overline{X2}$	$\overline{X3}$	X	C	X
INV (Fast)	A	X	X	$\overline{Q}$	Q	Q	C	0
INV	X	A	A	X2	X3	X	C	X
A.B (Fast)	A	B	X	$\overline{X2}$	$\overline{Q}$	Q	C	0
A.B	A	B	A	$\overline{X2}$	$\overline{X3}$	X	C	X
$\overline{A.B}$ (Fast)	A	X	B	$\overline{Q}$	$\overline{X3}$	Q	C	0
$\overline{A.B}$	A	A	B	X2	$\overline{X3}$	X	C	X
$\overline{A.B}$ (Fast)	A	B	X	X2	Q	Q	C	0
$\overline{A.B}$	A	B	A	X2	X3	X	C	X
A+B (Fast)	A	X	B	Q	$\overline{X3}$	Q	C	0
A+B	A	A	B	$\overline{X2}$	$\overline{X3}$	X	C	X
$\overline{A+B}$ (Fast)	A	B	X	$\overline{X2}$	Q	Q	C	0
$\overline{A+B}$	A	B	A	$\overline{X2}$	X3	X	C	X
$\overline{A+B}$ (Fast)	A	X	B	$\overline{Q}$	X3	Q	C	0
$\overline{A+B}$	A	A	B	X2	X3	X	C	X
A=B	A	B	B	X2	$\overline{X3}$	X	C	X
$\overline{A=B}$	A	B	B	$\overline{X2}$	X3	X	C	X
M2_1	SEL	A	B	$\overline{X2}$	$\overline{X3}$	X	C	X
M2_1B1A	SEL	A	B	X2	$\overline{X3}$	X	C	X
M2_1B1B	SEL	A	B	$\overline{X2}$	X3	X	C	X
M2_1B2	SEL	A	B	X2	X3	X	C	X

Figure 11 -- Table 1

Function Unit Internals

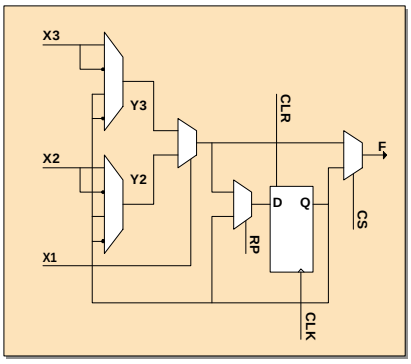

Figure 12

Figure 12 shows the implementation of the XC6200 function unit. The design uses the fact that any function of two Boolean variables can be computed by a 2:1 multi-plexer if suitable values chosen from the input variables and their complements are placed on its inputs. The Y2 and Y3 multiplexers provide for this conditional inversion of the inputs. The CS multiplexer selects a combinatorial or sequential output. The RP multiplexer allows the contents of the register to be 'protected'. If register pro-tection is enabled then only the programming interface can write to the register. It does not change when the X inputs to the function unit change, **even if it is clocked or cleared**. This feature is useful in designs containing control registers which are only to be written by an external microprocessor. The control inputs of all the mul-tiplexers, except the one switched by X1, come from con-figuration memory bits.

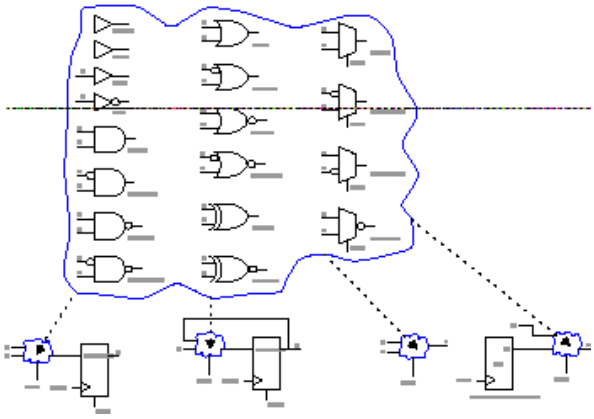

Figure 13

Cell Logic Functions

Each cell can be configured as any two-input gate function, any flavour of 2:1 multiplexer, constant 0 or 1, single input functions (buffer or inverter) or any of these in addition to a D-type register. This is illustrated in Figure 13. The gate names given correspond to standard Xilinx library part names for these primitives. Although three inputs are shown entering the combinatorial 'cloud', dual and single input functions are also possible. e.g. inverter + register or register alone. The buffer symbol is available in the CAD libraries. There is no requirement for the designer to buffer signals with this architecture. This is because signals are regularly buffered by routing multiplexers. Symmetrical functions are also possible but not shown in Figure 13. e.g. A.B (AND2B1) is shown but A.B (AND2B2) is not. This is because A and B are assigned to user signals by the logic mapping software to provide the required function. Thus a multiplexer with inversion on the SEL input is unnecessary because the mapping software can simply swap the signal assignments for A and B.

The sources of the X1, X2 and X3 input multiplexers are set automatically by CAD software during the logic mapping phase.Table 1 shows the assignments for all the cell multiplexers to compute the various logic gate functions. A NAND2B1 is equivalent to an OR2B1 with the inputs swapped and a NOR2B1 is equivalent to an AND2B1 with the inputs swapped therefore these gates are not listed in Table 1. The C and S signals are taken as the 'true points' for the gate mappings in this table. If the register within a cell is not used in the design then a special 'fast' version of most gates can be configured, using the register to provide a constant 1 or 0. For example a fast AND gate (A.B) can be configured by setting the register to 0 during configuration and assign-ing Q to Y3. A is routed to X1 and B to X2. X2 is assigned to Y2. When A changes to 0, Y3 is selected and F is forced Low as soon as the X1-controlled multiplexer switches. In the normal AND gate, there would be an additional delay as A propagated through the Y3 multi-plexer. Fast or normal gates may be specified by the designer but for optimal layout density this is best left to the logic mapping software. The multiplexer functions have a straightforward mapping with fixed assignments to X1,X2 and X3, with Y2 and Y3 providing input inversions as required.

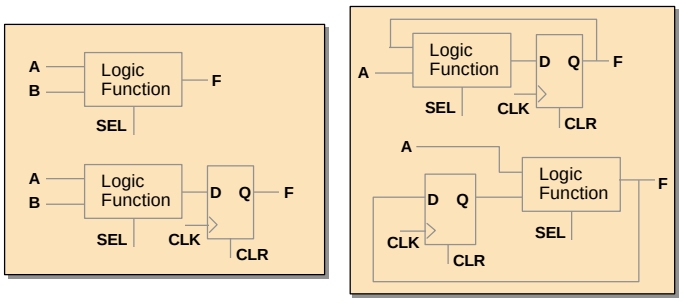

Figure 14

Register Access

XC6200 supports direct accesses from the processor to nodes within the user's circuit: the output of any cell's function unit can be read and the flip-flop within any cell can be written. During state reads a number of cell outputs are routed onto the CPU data bus. The signal which is actually read is either C or S in Figure 12, depending on whether the combinatorial or sequential output is selected. (See XC6200 Data Sheet Table 11 on page 21 for details of signal inversions) . These inversions are cancelled out by the readback circuit so that the true value of C or S is read.

These accesses are carried out through the control store interface and involve no additional wiring within the user's design. The CPU interface signals involved in addressing the cell state can be routed into the configurable array so that user circuits can detect that an access has been made and take appropriate action: for example, calculate a new value for an output register or process a value placed in an input register.

XC6200 Block Diagram

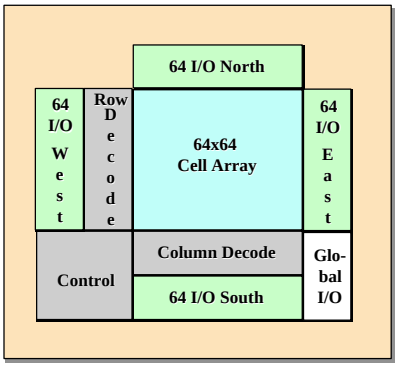

Figure 15

In many applications this access to internal nodes is the main path through which data is transferred to the processor and in some coprocessor type applications it may be the only external I/O method: user programmable I/O pads may not be required at all.

To allow high bandwidth transfers between the processor and internal nodes it is necessary to be able to transfer a complete processor data word of up to 32 bits in one memory cycle. For this reason state access bits within XC6200 are mapped into a separate region of the device address space from configuration bits so that all the bits in a word contain state access bits. Figure 15 is a block diagram of the XC6216 part, showing the row and column address decoders. Figure 16 shows the mapping of this area of the address space: there are 64 I/O signals from each column of cells and a 6-bit column address selects a particular column of cells to access. This row and column addressing scheme puts a

constraint on the placement of registers within the user's design which are to be accessed word-wide: they must be on the same column of cells within the array.

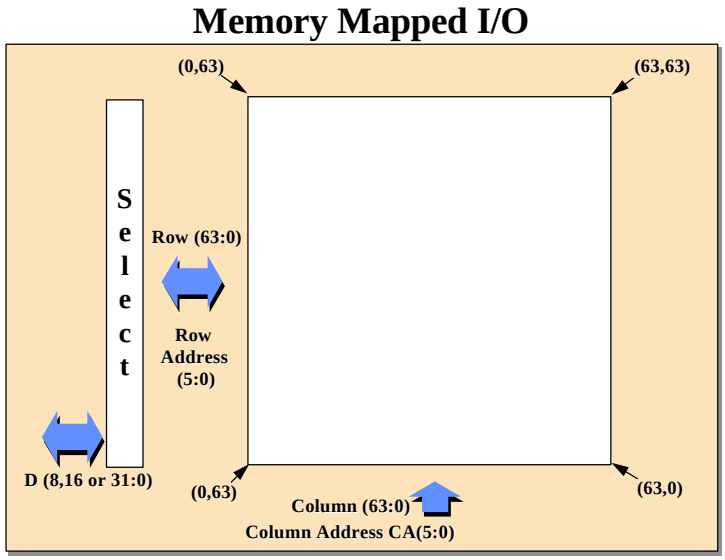

Figure 16

Timing

Externally, timing for state accesses is the same as for reads and writes to configuration SRAM. This is illustrated in Figures 17 and 18 and described in the section (see XC6200 Data Sheet --Parallel CPU Interface on page 19). Internally, state accesses differ from configuration SRAM accesses slightly in that writes to registers are synchronous. When the processor writes data to registers, the data is clocked in on a rising edge of the register's clock. The FastMAP interface presents the CPU data on the register's 'D' input and expects the register to clock it in. (The internal timing relationships are shown in XC6200 Data Sheet Figure 32 on page 46.) Data is present on the 'D' input from just after the first falling GClk edge of the Write Cycle to the second falling clock edge. If GClk is routed to the register then the data is clocked in at time t_2 in Figure 32 od XC6200 Data Sheet. If GClk is not being used to clock the register then the designer must ensure that the register's clock has a rising edge during the period the data is present.

During state reads there is no requirement for the regis-ters to be clocked. Data is read from the output of the cells and sampled by the FastMAP™ interface at time t_2 in XC6200 Data Shett Figure 33. Thus the data should be stable before and after this edge of GClk to guarantee a correct read.

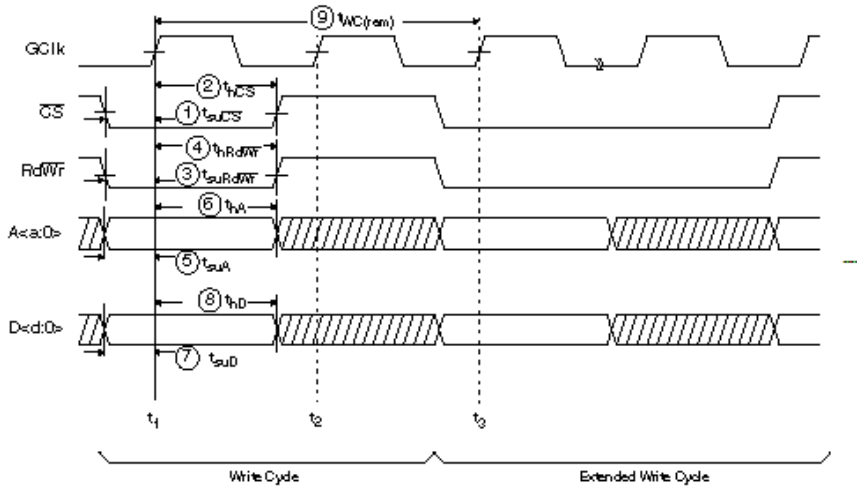

Figure 17 Configuration Memeory Write Cycle

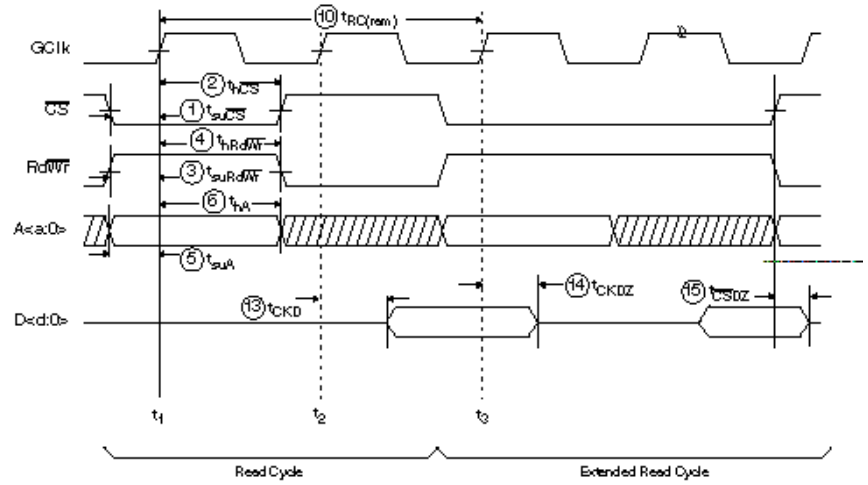

Figure 18 Configuration Memeory Read Cycle

The internal timing relationships illustrated in Figures 32 and 33 are the same regardless of whether the cycle is stretched by holding CS low, as shown in the extended cycles in Figures 30 and 31.

Map Register

XC6200 provides a mechanism for mapping all the possible cell outputs from a column onto the 8,16 or 32- bit external data bus, selecting only those cells which implement bits of the register to be accessed. Without this unit the processor would have to implement a complex sequence of shift and mask operations to discard those bits corresponding to cells not within the register, or the user would have to constrain the layout so that the register bits were in adjacent cells. The mechanism provided by XC6200 takes the form of a **Map Register**, one bit for each row I/O signal from the array. This Map Register can be read and written through the control store interface and is set up prior to state accesses. A logic 0 in the Map Register indicates that the cell in the corresponding row is part of the register to be accessed. The unit maps rows from the cell array onto external data lines starting with the least significant bit: thus the first row with a 0 in the Map Register connects to external data bus bit 0, the second row with a 0 in the Map Register to data bus bit 1 and so on.

This technique puts a further constraint on the user's layout: the cells implementing the bits of the register must be ordered so that less significant bits occur below more significant bits. However, there are no constraints about the relative separations of the cells. In practice these two placement constraints: cells occurring in the same column and in order vertically are easy to meet in datapath type designs.

Normally, the Map Register is set once to indicate the placement of the user I/O register which is then accessed many times. Therefore the two write operations required with a 32-bit bus to set up the Map Register represent a small overhead. In data path type designs where several registers are required, for example two input operand reg-isters and a result register, it is easy to ensure that the corresponding bits of the registers occur on the same row but different columns of the array so that the same Map Register value can be used with different column addresses to access the various registers.

If more 0s exist in the Map Register than there are valid data bus bits then a form of wildcarding occurs during writes. The data bus bits are allocated to the rows of the array with a 0 in their Map Register bit. Once all of the data bus bits have been allocated, Bit 0 of the data bus is allocated to the next row whose Map Register bit is a 0, Bit 1 of the data bus to the next row and so on. This feature means that an entire column of state registers can be written with a single 8-bit write. For example, if the Map Register contains all 0s and the CPU writes FFh to a particular column. All the state registers in that column are written with a 1. The default state of the Map Register is all 0s.

During reads, if there are more 0 bits in the Map Register than data bus bits, the first rows with 0 bits are mapped onto the bus.

If there are fewer 0s in the Map Register than data bus bits, the upper data bus bits, which are not mapped, are undefined during CPU reads and ignored during CPU writes.

An example of Map Register operation is shown in Figure 19. The position of the user-defined register within the cell array is defined by the 0s in the Map Register. Similar registers could be defined for every column in the array if desired.

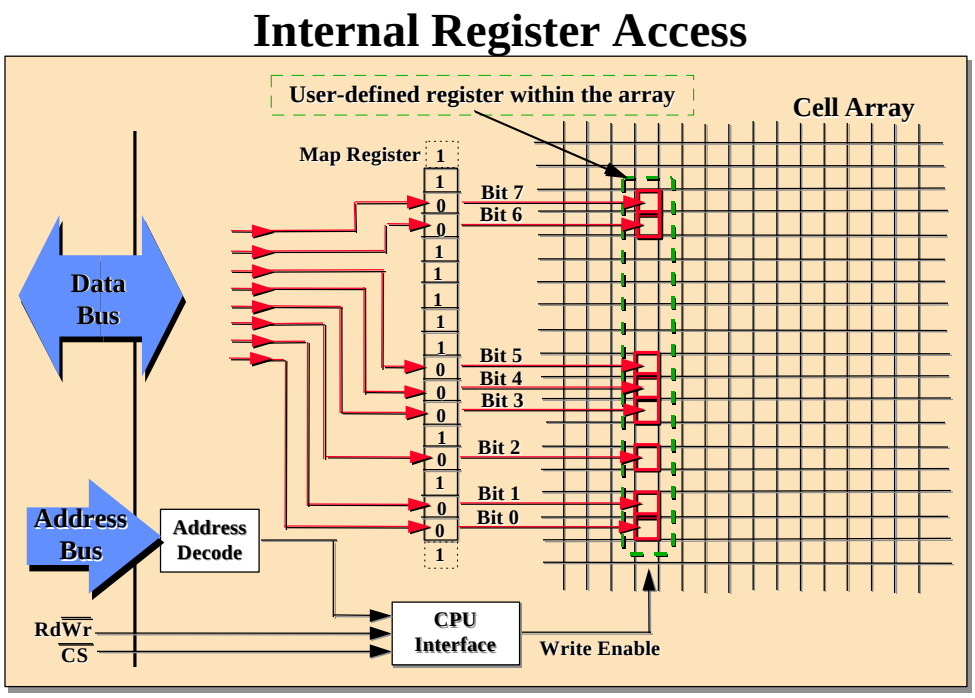

Figure 19

There is a delay of TMPST (see XC6200 Data Sheet - pg. 43.) after a write to the Map Register before the change takes effect. No state accesses should be carried out during this time. There are important notes on the register clocking requirements for reliable register reading and writing in the (see XC6200 Data Sheet - Parallel CPU Interface) section.

Mask Register

A mask unit controlled by a 32-bit register is placed between the external data bus and the internal data con-nections. When the external data bus is 8 or 16 bits wide only the bottom 8 or 16 bits of this register are significant. A logic '1' in a bit of this register indicates that the corre-sponding bit of the internal data bus is not relevant. Bit locations corresponding to 1s in the Mask Register retain their

values when written. On a write operation the corre-sponding bit line is not enabled and the state information for that bit is not changed. When the device is reset the Mask Register contains all logic 0s corresponding to all data bus bits valid.

During CPU reads, valid register bits which are disabled are read as '0'. Invalid bits (bits which do not physically exist for the register being read) may be read as '0' or '1'.

The Mask Register does not affect state register accesses. In this case the Map Register can be used to prevent certain bits being modified.The Mask Register is also ignored during reads and writes to XC6200 Control Registers. These are memory locations which control various XC6200 functions and are defined in "XC6200 Data Sheet -- Address Mode 11 - Programming Control Registers" on page 25

Programming

The binary data for configuring XC6200, generated by CAD software from the textual description of a user design, must be downloaded into the part itself. This may be performed in several ways. Generally the fastest and most efficient way is by writing directly to the control store, mapped into the address space of a host processor. If a microprocessor or other parallel data source is not available then the serial programming interface may be used.

Parallel CPU Interface

XC6200 has a full parallel CPU interface, referred to as 'FastMAP '. This makes all the configuration SRAM and logic cells appear as conventional memory mapped SRAM. The FastMAP interface is based on Chip Select (CS) and Read/Write (RdWr) control signals. The CS signal can be used to address a single part within an array of devices and allows data to be read or written. Timing for these signals is illustrated in Figures 17 and 18. These figures show that the programming interface is synchronous. The GClk input is used to sample all the interface signals. GClk is also used when accessing user registers as illustrated in Figure 19. This is an important point, as only registers clocked directly by GClk can be reliably read or written using this method. This is because the value written by the CPU is presented to the inputs of the cell registers just before t_2 in Figure 17 and held there until GClk goes Low again. Thus it is essential that the register receives a rising clock edge at t_2. This can be guaranteed if GClk is used to clock the registers. If another signal is used then it must have a rising edge at t_2 for FastMAP™ register writes to work. For reliable register reads, the register contents must be stable between t_1 and t_2 in Figure 18. Figure 18 shows two separate read cycles - a normal cycle immediately followed by an extended cycle. In the normal read cycle CS is sampled Low on the first rising GClk edge (t_1) and High on the next (t_2). The data bus is then driven until the next rising GClk edge (t_3). In cases where this is not long enough, the read cycle can be extended by keeping CS asserted beyond t_3. This is equivalent to adding wait states. In this case the data bus is driven until CS is deasserted. CS should not be allowed to go High and Low again. This would cause another cycle to begin. CS is sampled on every rising GClk edge. Other CPU interface signals such as RdWr and the Address Bus are only

sampled on the first GClk edge of the cycle (t_1 for the first cycle and t_3 for the second in the figure examples).

Extended write cycles are also possible, however these are functionally no different to normal write cycles, the data and address busses still being sampled on the first rising GClk edge of the cycle (t_3 in Figure 17).

CS must always be sampled as a '1' before the next cycle can begin. In Figure 18 the extended read cycle starts immediately after the normal read cycle at time t_3. A write cycle could not start until the next rising GClk edge as the data from the read cycle is still on the data bus.
The SRAM programming interface is supplemented by additional hardware resources designed to minimize the number of processor cycles required for reconfiguration. These resources are initially inactive after a reset so the device looks like an SRAM. Before any data can be read from the device using the FastMAP™, the Device ID Registers must all be correctly written. This is descibed under "XC6200 Data Sheet -- Device Identification Register" on page 26
The control store layout is designed to minimize the overhead of computations required for dynamic access while maintaining adequate density to minimize the external storage required for device configurations. When an external processor is used to configure the device it may be convenient to use a compressed format of the configuration information.

A feature of the XC6200 architecture is that a rectangular area of cells specified as a hierarchical block within a user's design corresponds directly with a rectangular area within the configuration memory of the XC6200 device. This means that a block within the user's design can be dynamically replaced with another block by the host processor, reconfiguring only the corresponding area of the control store. The binary data for both blocks can be pre-calculated from the cellular design and the actual replacement can be carried out very rapidly using block transfer operations.

The format of the address bus to the XC6216 device is shown in Table 2. Larger XC6200 devices have proportionally more bits allocated to row and column addresses.

Mode(1:0)	Column(5:0)	Column Offset(1:0)	Row(5:0)
15:14	13:8	7:6	5:0

Table 2 -- Address Bus Format (XC6209 and XC6216)

Mode(1:0)	Column(5:0)	Column Offset<1:0>	Row(5:0)
17:16	15:9	8:7	6:0

Table 3 -- Address Bus Format (XC6236 and XC6264)

All the configuration memory can be accessed as 8-bit bytes. When a 16-bit transfer occurs Address<0> is irrelevant. When a 32-bit transfer occurs Address<1:0> is irrelevant. Data Bus bits <7:0> are written to the address with Address<1:0>=00, bits <15:8> are written to the address with Address<1:0> = 01, etc. The Address Mode bits are used to determine which area of the control store is to be accessed according to Table 4.

Mode1	Mode0	Area Selected
0	0	Cell Configuration and State
0	1	East/West Switch or IOB
1	0	North/South Switch or IOB
1	1	Device Control Registers

Table 4 -- Address Mode Selection

Understanding The Configuration Bits

The full memory map is given in Tables 5 and 6. From these tables it is possible to work out the address for any bytes of configuration or cell state register in the FPGA. The address/data pairs are normally calculated automatically by XACT step Series 6000 and written to a .cal file.
From Tables 7 to 21 in XC6200 Data Sheet it is possible to work out what data needs to be written to the above addresses to change the configuration of any routing multiplexer or cell in the FPGA. These tables are split into subsections - Cells, East/West Switches and North/South Switches. Within each section, the bytes which control the switches are defined first. This table is then followed by a group of tables which define the coding of the bits within these bytes.

A[15:0]	Register	A[15:0]	Register
C000	Device Config	C001	ID (Byte1) (='I')
C004	Row Wildcard	C002	ID (Byte2) (='I')
C005	Column Wildcard	C003	ID (Byte3) (='I')
C008	Mask (Byte0)	C004	ID (Byte4) (='h')
C009	Mask (Byte1)	C005	ID (Byte5) (='X')
C00A	Mask (Byte2)	C006	ID (Byte6) (=' ')
C00B	Mask (Byte3)	C007	ID (Byte7) (='X')
C010	Map (Byte0)	C008	ID (Byte8) (='C')
C011	Map (Byte1)	C009	ID (Byte9) (='6')
C012	Map (Byte2)	C00A	ID (Byte10) (='0')
C013	Map (Byte3)	C00B	ID (Byte11) (='0')
C014	Map (Byte4)	C00C	ID (Byte12) (='0')
C015	Map (Byte5)	C00D	ID (Byte13) (=' ')
C016	Map (Byte6)*	C00E	ID (Byte14) (=' ')
C017	Map (Byte7)*	C00F	ID (Byte15) (=ID#)
C030	ID (Byte0) (='X')		

Table 5 -- Control Register Memory Map

A[17:0]	Register	A[17:0]	Register
30000	Device Config	3001D	Map (Byte13)*
30004	Row Wildcard	3001E	Map (Byte14)*
30005	Column Wildcard	3001F	Map (Byte15)*
30008	Mask (Byte0)	30020	ID (Byte0) (='X')
30009	Mask (Byte1)	30021	ID (Byte1) (='I')
3000A	Mask (Byte2)	30022	ID (Byte2) (='I')
3000B	Mask (Byte3)	30023	ID (Byte3) (='I')
30010	Map (Byte0)	30024	ID (Byte4) (='h')
30011	Map (Byte1)	30025	ID (Byte5) (='x')
30012	Map (Byte2)	30026	ID (Byte6) (=' ')
30013	Map (Byte3)	30027	ID (Byte7) (='X')
30014	Map (Byte4)	30028	ID (Byte8) (='C')
30015	Map (Byte5)	30029	ID (Byte9) (='6')
30016	Map (Byte6)	3002A	ID (Byte10) (='0')
30017	Map (Byte7)	3002B	ID (Byte11) (='0')
30018	Map (Byte8)	3002C	ID (Byte12) (='0')
30019	Map (Byte9)	3002D	ID (Byte13) (=' ')
3001A	Map (Byte10)	3002E	ID (Byte14) (=' ')
3001B	Map (Byte11)	3002F	ID (Byte15) (=ID#)
3001C	Map (Byte12)*		

Table 6 -- Control Register Memory Map

Programming Control Register Memory Map

Table 5 lists the address for all the control registers for the XC6209 and XC6216. Each register may be written in 8-bit bytes. If 16 or 32 data bits are available then less writes are required. For example eight write cycles (to C010, C011,.....,C017) are required to change every bit of the Map Register with an 8-bit bus. Only two write cycles (to C010 and C014) are required with a 32-bit bus. If only some bits are to be changed within a particular register then only the appropriate bytes need be written. XC6236 and XC6264 have 18-bit address busses. The Control Register memory map for these parts is given in Table 6.

I/O Architecture

User-configurable Input/Output Blocks (IOBs) provide the interface between external package pins and the internal logic.

One IOB is provided for every cell position around the array border. IOBs are connected to fixed pad locations. There are more IOBs than available pads, hence some IOBs are 'padless'. However it is still possible to route signals from padless IOBs to device pins.

Figure 20 is a simplified diagram of an IOB and its associ-ated IO pad. The IOB is located at the array border and the pad is located close to its device pin. The pad may be located some distance from its associated IOB. (The mapping of IOBs to tables starting on page 48 in the XC6200 Data Sheet.)

The XC6200 IOB architecture very powerful feature: routing either an array from the device pin. Every signals (e.g. CS, RdWr, Address Bus, Data Bus, etc.), passes through an IOB. This means that all the control signals can be routed designs. Similarly, user internal control circuitry. For example a user signal could be used to drive the internal CS signal rather than the CS pin.

As an example of the power of this feature, an XC6200 design could include an address decoder which decoded microprocessor read/write cycles and produced appropri-ately retimed signals for all the parts on a board including itself, thereby removing the need for address decoding PALs or discrete logic.

Each IOB has an array data input and a control data input, labeled ArrayDToPad and ControlDToPad in Figure 20. Associated with these inputs are two enable signals -ArrayEnable and ControlEnable. These signals control whether the pad associated with this IOB is in the input or output mode. Each IOB also supplies ArrayData and Con-trolData when acting as an input.

Input/Output Architecture

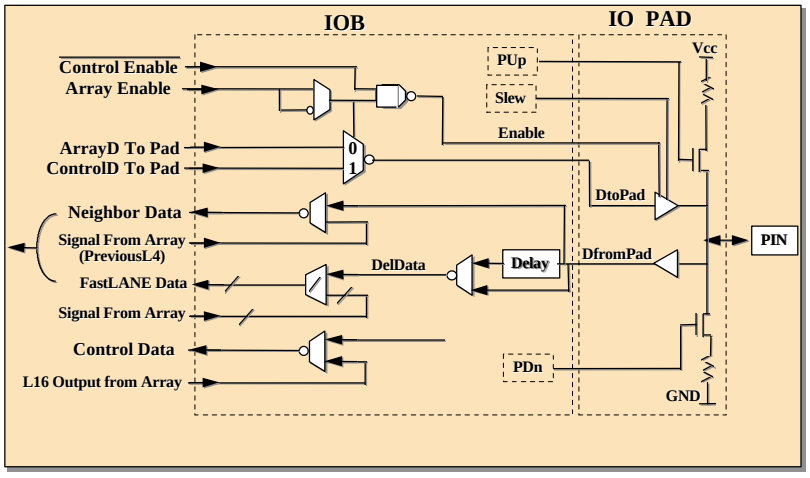

Figure 20

The 'Control' signals are routed to the internal XC6200 control circuitry. If control signals are not required all the time then these IOBs can be used to route other user signals into the array. For

example if only eight data bus bits were continuously required, the remaining twenty-four IOBs associated with the data bus could be used to route user signals to/from the array. ControlEnable comes either from the internal XC6200 control circuitry if there is a bidirectional control signal or output signal on that IOB, or it is tied inactive.

The 'Control' signals are also referred to as 'B' signals in this data sheet. ControlDToPad = DToPadB, ControlEnable = EnToPadB and Control Data = DFromPadB. The L16 output from the array, which can be routed onto Control Data, is also referred to as DForPadB.

The DFromPadB output is unconnected in IOBs which have an output-only 'B' signal, such as SECE. IOBs which have an input-only 'B' signal, such as CS, have the DToPadB input tied Low and the EnToPadB input tied High. IOBs which have no associated 'B' signal also have DToPadB tied Low, EnToPadB tied High and DFromPadB unconnected.

"Padless" IOB Configuration

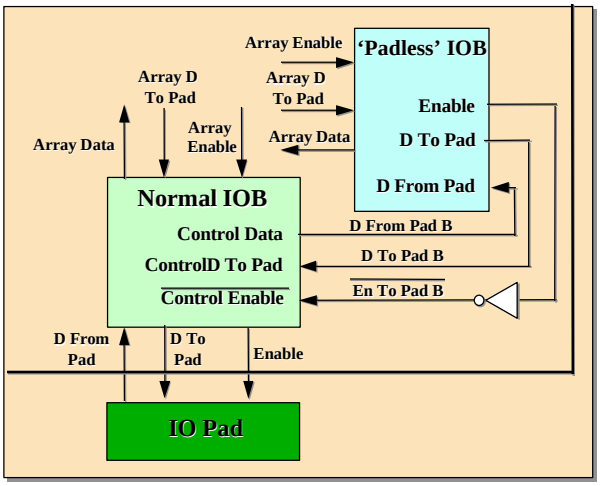

Figure 21

There are less real control signals than IOBs, hence the three signals, DFromPadB, DToPadB and EnToPadB, on some IOBs are not connected to the XC6200 control logic. Some of these spare 'B' signals are used to route data to and from the padless IOBs mentioned above. The 'B' signals on the padless IOB are not used. This is illustrated in Figure 21. This arrangement allows data to be routed in or out of the chip via an IOB which has no associated IOPAD. (The padless IOBs and their padded partner IOBs are detailed in the pinout tables starting on page 48 in the XC6200 Data Sheet). For example, in a XC6216 IOB W0 is padless and is partnered with IOB S12, which has a pad.The ArrayEnable, Array Data and Control Data multiplexers are controlled by configuration

RAM bits. A fixed delay may be optionally applied to Array Data inputs. This allows the input data hold time specification to be removed.

The ArrayEnable and ArrayDToPad signals can be config-ured to constant 0 or 1 values within the logic array. The constant values are particularly useful for the enable signal when the pin is to function as an input or output rather than a bidirectional pin. Constant values on the data signal and a computed value on the enable signal produce open drain pull-up (DToPad=1) or pull-down (DToPad=0) pins.

For more deatils on the XC6200 Family see the XC6200 Data Sheet from Xilinx Inc.

Designing for the XC6200

Editor View of Routing

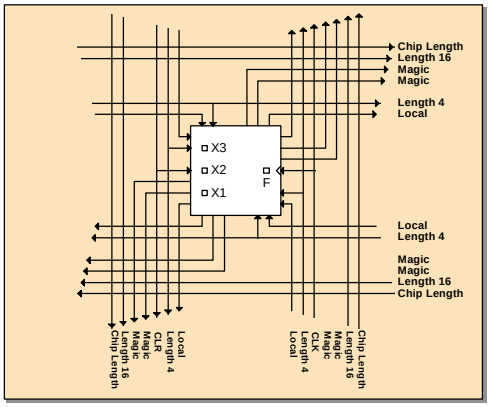

Figure 22

Keeping Design Heirarchy

- **Object oriented design**
  - promotes object re-use
  - designers think like this anyway in VHDL and schematics
- **"True" Hierarchy**
  - definitions & instances

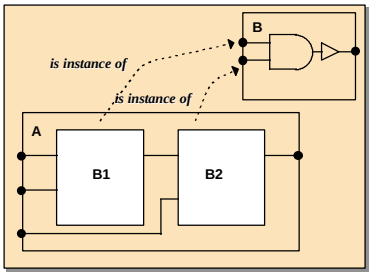

Figure 23

- **Simple Block with Multiple Layouts**
  - can have tall, wide, fast, etc.
  - optimized for speed, area

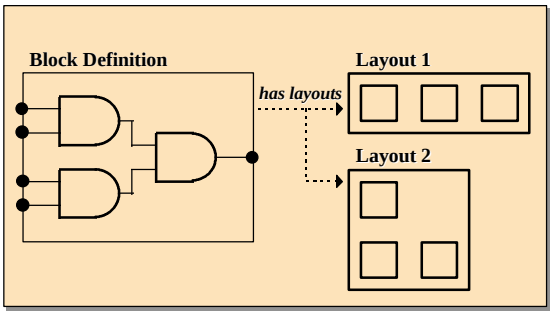

Figure 24

Block, Layouts and Instances

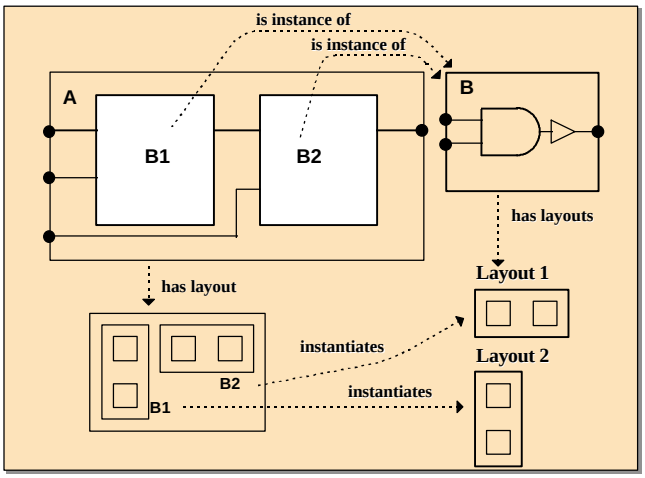

Figure 25

Coordinate System

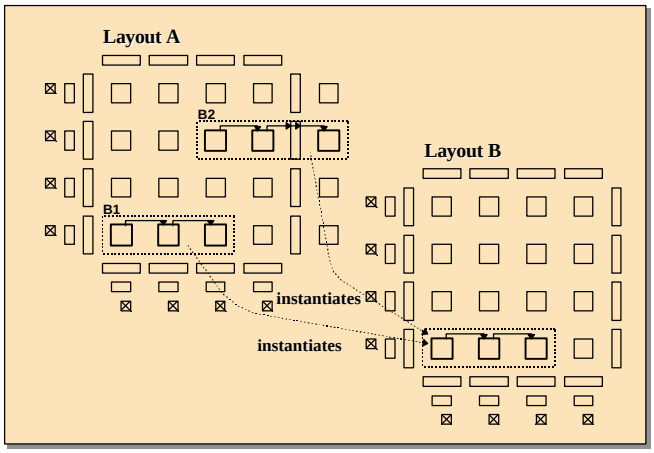

Figure 26