5 System Considerations

The implementation of combinational logic and registers has been presented in the previous chapter. This chapter considers the additional features required when translating from the system design level to logic. It includes a discussion of suitable control and timing techniques plus a description of the facilities that must be included in order to ensure that a design is testable. This is preceded by a general description of the translation process and the design options available to the designer.

5.1 System to Logic Translation

At the highest level in the design hierarchy, the system description is expressed in terms of functional blocks and data paths. This architecture diagram effectively details the operations performed on the data and the information flow through the system, without specifying how these features are implemented. This part of a design can be regarded as the data architecture.

The control and timing circuitry associated with a design are normally omitted at this level. This is because their inclusion is implicit in any design and also because such circuitry can be regarded as an implementational detail. Similarly, although the approach to testing must be established at the system design level, test paths and associated circuitry will probably not appear on the top level diagram.

Figure 5.1 shows an example of the data architecture of a general-purpose central processing unit (CPU) plus the functional blocks for timing and control. Unusually, some of the control paths are indicated; these show the interaction between the data and control logic. The exact registers required in the system are dependent upon the chosen ordercode. Nevertheless, the diagram is sufficiently detailed to serve as an example for discussion here and elsewhere in this chapter.

The design is centred around the arithmetic and logic unit (ALU). Operations proceed by the control enabling a source register on to highway A and/or highway B, specifying an ALU function and finally clocking the ALU result on highway C into a destination register. The control signals are updated by the timing circuitry so that a sequence of operations to fetch and execute intructions can take place.

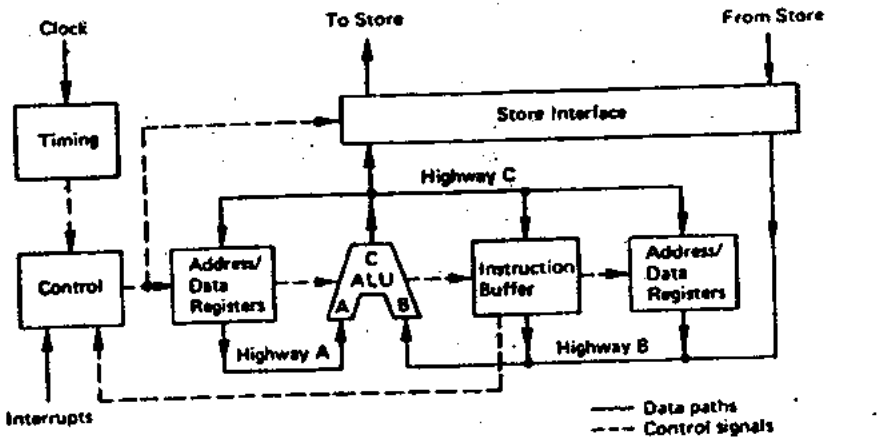


Figure 5.1 Processor architecture example

In general, a design is testable if its flip flops can be initialised to a known state and subsequently monitored. Figure 5.1 has an in-built mechanism for observing internal states, since the content of all processor registers can be placed on highway C. Furthermore, if all registers connected to highway A (or highway B) are disabled, then external data can be placed on this highway and written into any register; this allows initial states to be entered into the machine. Thus the processor is fully testable.

The logic diagrams detail every logic element that is required to implement a design. Normally, the translation of the data architecture to logic diagrams is fairly direct and straightforward. The composition of the control and timing circuits tends to be more difficult. The control circuits have to cater for every condition that will arise within the system and for complex systems it is easy to overlook some of the conditions. Hence, some formal method of specifying the control signals is highly desirable; this aspect is covered in more detail in section 5.3. The timing logic is closely associated with the control and evolves from the choice of control type.

Once the logic diagrams are drawn, decisions have to be made as to how to implement the logic. Some functions may be very efficiently implemented with existing LSI or VLSI chips available 'off the shelf' from a manufacturer and are therefore clearly not candidates for integration; random access memory, for example, is in this category. However, functions that can only be implemented with existing SSI or MSI chips or architectures suited to bit slice techniques which are unobtainable are suitable for integration. The possibility of integration

should also be examined where existing LSI functions are inefficient to use because of redundancy of included features and/or lack of some required functions.

Larger chip sizes and smaller geometries allow large sections (or all) of a design to be accommodated on a chip. Furthermore, it is likely that to exploit the full area, a chip will contain circuitry from more than one designer. One important aspect of the logic design is the interfacing of logic designed by different members of a team, since poor communication between people can often result in faults arising here. It is therefore vital to check carefully the interfaces between different logic sections. Unfortunately, the number of logic elements on a chip is likely to necessitate partitioning the design for the purposes of logic simulation and this makes the checking of the interfaces more difficult.

In general, designers wishing to integrate some logic will have access to either semi-custom or full custom facilities. The next section outlines the differences between these two design approaches.

5.2 Full Custom and Semi-custom Design

The most efficient implementation of a function is realised by a full custom design. The content of each layer is entirely under the designer's control, resulting in an optimised use of the silicon area.

The implementation of a full custom design requires an understanding of all aspects of the design procedure. Thus the designer has to be knowledgeable about suitable system architectures for integration, the approaches to logic design, the electrical characteristics of MOS circuits, the fabrication process and the design rules appertaining to layout. This information is put to practical use in realising each level of the design hierarchy and in translating to the next level down.

In addition, the designer needs to be familiar with the software package to be used for entering and checking the chip description at the geometric layout, transistor circuit and logic element levels. Since it is necessary to initiate this package, some understanding of the user-machine interface of the computer on which it runs is also required.

It will be appreciated that it takes time to assimilate this large amount of information. In particular, significant times can be encountered if documentation is unavailable or poorly presented. Consequently there is a 'start-up' time before design can commence.

Once the design is underway, considerable design effort is required to define each layer on the chip; for example, the NMOS process described requires seven masks. This leads to a long overall development time: eighteen months would be realistic for a first-time designer implementing a 'hand-crafted' design. This time

can be shortened by automating some aspects of the design. The most useful design aid in this respect is automatic routing of interconnections between circuits. Manual routing dominates the development time, even for a relatively small design.

The design effort needed and the number of masks required also makes it expensive to manufacture a chip. In a commercial environment, such a cost is only viable if a large number of chips are required (more than 100 000 devices per year) or if high performance is the design criterion. In situtions where low-volume prototypes are required, costs can be reduced by including several designs on a wafer.

However, a more effective approach for a low-volume requirement in terms of design effort and cost is semi-custom design. Here the user designs the logic in terms of available logic functions which the manufacturer has designed and characterised. In its simplest form, the area is organised as a two-dimensional array of logic cells surrounded by peripheral cells. This arrangement is referred to as a 'gate array' or an 'uncommitted logic array' (ULA).

Each logic cell in the array is normally identical and typically consists of four to eight transistors which can be connected up in different ways (on the metal layer(s)) to form primitive logic functions. For example, consider a CMOS cell consisting of two NMOS and two PMOS transistors. By appropriate connections between these devices and to power and ground, this cell can be configured to form two inverters, one two-input nor gate, one two-input nand gate, two pass gates or an inverter plus a transmission gate.

The peripheral cells provide an interface between the circuitry of the logic cells and circuits external to the chip. These cells are placed in fixed positions around the edge of the chip and include a power and ground pad. The remaining peripheral cells can be configured by the user to be either an input or output pin.

With a ULA, all silicon layers are pre-formed, except for the metal layer(s). Thus the user only has to determine the interconnections required between cells and the configuration of the cells to realise the required logic; all these connections appear on the metal layer(s). Thus the geometrical layout stage in semi-custom design reduces to specifying one mask for a fabrication process with one metal layer (or three masks for a two metal layer process). Again automatic routing of this stage is essential for a short development time.

A semi-custom approach also simplifies the design process in other ways. The designer can think of and implement the design solely in terms of logic gates with well-characterised electrical parameters; the design process is similar to that of designing a system using an available logic family. This enables the design to proceed at a much higher level in the design hierarchy than is possible in full custom, and also results in a much shorter 'start-up' time.

The ease of semi-custom design leads to a relatively short design time of a few weeks. In addition, the simplicity of this design approach and the need for only one (or three) user-specified mask(s) make it probable that a working design will be obtained at the first attempt.

The low manufacturing cost of a semi-custom design compared with full custom also makes the gate array attractive. The pre-formed ULA is a general-purpose circuit and has the low cost associated with a high-volume product. The user, of course, has in addition to bear the cost of the metal mask(s), but this is far lower than the cost of providing all the masks required by full custom design.

There are disadvantages from adopting a semi-custom approach. These arise from a less efficient utilisation of area, since functions are constructed from general-purpose primitive functions rather than special-purpose circuits. Also, the use of primitive functions leads to a large number of interconnections and this usually prevents more than about 80 per cent of the logic cells being used. A range of gate array sizes is available, starting at around 500 gates and increasing by a factor of approximately two up to about 10 000 gates. Thus the amount of logic that can be placed on a ULA is rather limited.

In addition to the primitive functions available from a logic cell, most manufacturers now provide a library of more complex functions, such as flip flops, decoders, counters etc. These are usually formed from a group of cells which are interconnected on the metal layer(s).

An alternative approach to semi-custom design is that based on standard cells. Here, the manufacturer designs and provides performance and area-efficient functions. The range of functions being offered is rapidly expanding and general-purpose circuits at the LSI and MSI levels are now emerging. These vary from random access memory to logic elements such as arithmetic and logic units, PLAs, shifters and multipliers.

The user now constructs the final chip design from these building blocks which are proven and well characterised. This approach leads to the fast development time associated with gate array design, while the silicon area can be utilised effectively at the cost of providing an individual mask for each layer of a chip design. An intermediate semi-custom approach, which avoids this cost penalty, results if the silicon is pre-formed below the metal layers but contains a mixture of functions; these could include memory elements (flip flops or random access memory), gate array cells, arithmetic and logic functions.

Approaches based on standard cells allow an increased packing density on that available for a gate array. Consequently, chips in excess of 20 000 gates are becoming achievable and up to 50 000 gates are anticipated. An extension of the standard cell technique would seem to offer a viable method for managing and implementing a VLSI design.

5.3 Control

The control unit in a design determines the action of the hardware at any time. Techniques for implementing control are either formal or informal. In the former type, operations are synchronised to a clock. Informal or asynchronous

control results when actions are initiated by the detection of states arising within the hardware.

In general, a chain of delay elements driven from detected states within the hardware forms the basis of an asynchronous control unit. This approach yields the fastest possible function speed. However, such a unit consists of special-purpose hardware which can only deal with the conditions built into it. Thus compared with synchronous methods of control, it is more difficult to design, commission, monitor and modify.

The problems of designing an asynchronous control and the need to cater for all the conditions that will arise within the hardware make it virtually impossible to produce a correct control at the first attempt (despite careful logic simulation). Thus a vital feature of this type of control is an ability to change it! It should, therefore, be apparent that this informal approach to control is very unsuitable for inclusion in a chip design.

Formal control techniques can be split into two groups. The first is referred to as 'fixed control logic'. Here, control is hardwired into the system and typically consists of a chain of flip flops connected as a shift register. The principle is illustrated in figure 5.2. Each flip flop shown is a master-slave device and its (slave) output is associated with a set of control signals which cause a particular action to be performed in the data logic. Thus a flip flop is required for each control state and in a non-overlapped system only one flip flop is set at any time.

Initialisation of the chain sets the flip flop in stage 1. Thereafter, the master and slave clocks are alternately applied and at every slave clock, control progresses sequentially down the chain. The slave clock causes a flip flop in the next stage to be set and the set flip flop in the current stage to be reset. At points in the chain, the action required next may depend upon conditions within the hardware. For example, the implementation of an ordercode in a processor necessitates a different sequence of actions for each instruction and the sequence required is determined by the function bits in the order. This choice of actions is implemented in the control logic by the control chain splitting, as shown at stage 3; the conditions applied at the input of this stage effect a conditional branch at this point.

Ideally the time for the action associated with each stage is similar and this determines the clock period or system beat time. However, some actions span more than one clock period and waiting for data from a store may involve several beats. Clearly, the next operation can only be initiated when the current action has completed. Hence it is necessary in these cases to inhibit the application of the clocks to the control chain. This is achieved by gating the master and slave clocks with a Wait signal which is generated by the control whenever appropriate. (Strictly, only the slave clock need be gated.)

A synchronous control leads to some redundant time at the end of some actions since the next operation will not commence until the start of the next beat. Thus a poorer speed is obtained than with an asynchronous control. How-

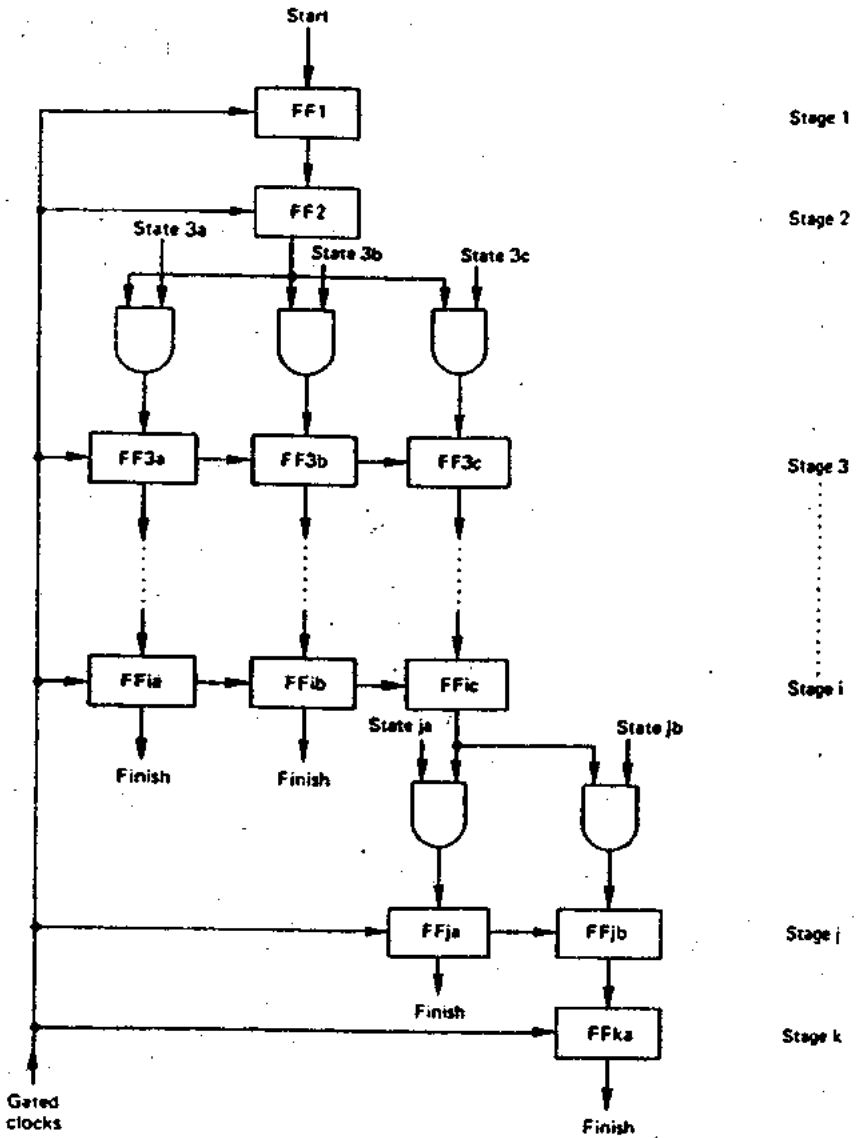


Figure 5.2 Fixed control logic

ever, the main drawback of the fixed control approach, like the asynchronous control, is the specialised nature of the control hardware. Again, the control needs to be capable of modification and changes are difficult to incorporate since they often lead to alterations in the control signals generated by existing portions of the chain. Thus the fixed control logic technique is only practical for small chip designs or for small portions of a chip, such as a multiplier.

A general-purpose approach to formal control is that of a microprogrammed control. Here the control information is held in a memory known as the microprogram store. Each line of the store is associated with a particular control state and contains the control signals for that state. Thus store lines can be regarded as holding microinstructions or microcode, and operations are implemented by accessing and obeying the correct sequence of microinstructions.

Microprogramming does not yield the same functional speed as the other techniques described. However, the flexibility and advantages gained by adopting this approach in terms of chip design more than outweigh any drawbacks. It is now possible to change the existing control merely by rewriting lines in the microprogram store. In the same way, new facilities can be included and existing operations removed by adding new microinstructions and deleting others. Thus it should be apparent that the system seen by the user is that provided by this store.

The organisation of a microinstruction closely reflects the data architecture. The lines are normally divided into fields, where each field has a fixed position in the line and has a specific purpose; for example, the processor organised around an ALU, as in figure 5.1, would require three fields to perform a register-to-register transfer; these are a source register field, an ALU function field and a destination register field. Using the concept of fields, the microcode required in a store line can be written in terms of a simple, low-level assembly language. This enables appropriate software to check that the specified control actions are compatible and then to generate the required binary pattern. This mechanism can also be used to produce up-to-date documentation of the contents of the control store.

Figure 5.3 shows hardware for a microprogram control. The address of the next microinstruction to be obeyed is loaded into the address register (Addr Reg) when the LDA clock is applied. The store is accessed at this address and the line contents are loaded into the microcode register on the LDD clock. Part of the information in the line comprises the control signals which cause activities within the data logic; these are equivalent to the control signals generated by the fixed control logic when a flip flop in the control chain is set. The remaining fields are used to determine activities within the timing and control units.

Some control states lead directly to the action contained in the next microinstruction, while others require the activities that follow it to depend upon conditions and states within the data logic. This is effected by including a condition field and a next-address field in the microinstruction. The address-generation hardware combines these fields with incoming states and conditions from the

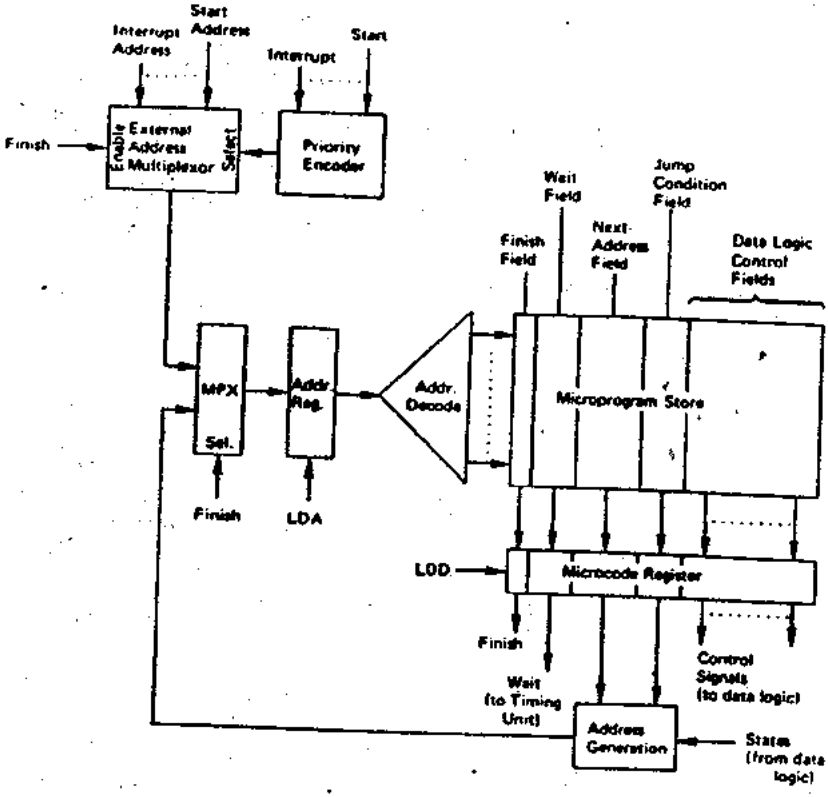


Figure 5.3 A microprogram control

data logic, to determine the next microprogram address to be jumped to. The next-address field contains the address normally accessed next from the microprogram store, while the condition field specifies the type of jump required (conditional or unconditional) and whether the next-address is to be modified.

It is usual to include a number of different types of jumps in a microinstruction. Normally available options include an unconditional jump to the next-address, an unconditional jump to the next-address modified by states within the system and a conditional jump to the next-address (with a jump to the next-address modified if the condition is not true). The two latter options can effect a multi-way switch and a conditional branch respectively.

Once a sequence of microinstructions has commenced, it continues until the finish field indicates that the operation is complete. The finish signal is used to terminate the selection of addresses from the address generation logic via the

two-to-one multiplexor (MPX). Initialisation of a new code sequence now arises from sources external to the control unit which are arranged in order of priority. In this way, a number of tasks can be initiated and executed by the microcode. Hence the microprogram hardware shown will support both simple and complex designs.

The microinstruction has to control all parts of the data logic, normally at the register or logic block level. This, combined with the fields associated with the control and timing, leads to the width of the microprogram store being much wider than the width of the data logic paths, even when the fields are encoded. However, the number of store lines can be relatively modest. Thus for example, a 512-line, 80-bit wide microprogram store would be capable of controlling most 16-bit wide processors and assisting with 'housekeeping' tasks, such as dealing with interrupts. This level of hardware support may make the choice of a fixed control more attractive for a small amount of logic.

The system beat time is determined by the maximum time to perform the activities defined by lines in the microprogram store, which do not have to wait for external data. Microinstructions which request information external to the system, such as from a store, are likely to take more than one beat to execute. Again, this is effected by using gated clocks in the control. Microinstructions where waiting occurs are indicated by a bit being set in their wait field. This bit is used to stop the LDA and LDD clocks in the control unit and these are only recommenced when the required data has arrived.

There is also an execution time associated with the circuitry of the microprogram control. Here, time is needed for a line read from the microprogram store to be loaded into the microcode register and the microinstruction decoded. Following this, time is required for the next-address in this store to be calculated and then for this new line to be read out. Clearly this execution time should be less than the system beat time, otherwise the data logic will be held up by the control. For this reason, the operation of the control and data logic are usually synchronised to the system beat.

The microprogram store can be implemented in random access memory and this is useful in a prototyping environment. However, it should be noted that the data is only retained while the memory is powered up and it needs to be rewritten at each power-up. Once the microprogram store contents have been finalised, it is more convenient to store microinstructions in a read only memory (ROM). This is a non-volatile memory whose data is fixed. Thus an initial operation is required to establish the desired information in the memory and thereafter the data is only read.

In the MOS technology described, an ROM can be implemented using the PLA structure. Referring to figure 4.5, the AND plane performs the address decode. Thus the AND plane inputs are the true and inverse phases of the address bits. The outputs from the AND plane are the address decode lines and there is one for each microinstruction contained in the OR plane. The address decode line selected by the input address is activated (high), enabling the asso-

ciated microinstruction in the OR plane to be output. All other AND plane outputs are low and thus have no effect on the OR plane operation. The placement of transistors in the selected microinstruction causes the associated output lines from the OR plane to go low while all other OR plane output lines remain high. Clearly, the size of ROM that can be implemented in this way is limited by capacitive loading and area constraints.

5.4 Timing

While the control specifies the activities within a system during a beat, the timing unit determines when these occur and updates the control at the end of the beat. Thus the control and timing circuitry are closely coupled. Integration imposes additional constraints if a correct design is to be obtained at the first attempt. A formal approach is needed which incorporates 'safe' timing strategies. These avoid flip flop synchronisation problems and the initiation of incorrect actions by race hazards. The former arise if the flip flop set-up and hold times are not observed, while race hazards are the unwanted voltage pulses which arise as a result of differences in delays through gates and interconnections. Again, synchronous timing, rather than asynchronous, is a fundamental requirement in achieving these aims.

The activities performed in a beat normally have to be sequenced and thus there are a number of distinct phases associated with an action. For example, consider the microprogram control of figure 5.3 controlling the processor of figure 5.1. If both are synchronised to the system beat, then an action in the data logic and control can be split into four phases (see figure 5.4).

At the start of the beat, the new microinstruction is loaded into the microcode register. During phase 1, this register's outputs settle to their new value and the microinstruction is decoded. Referring to figure 5.1, the inputs to the ALU are then selected and become valid during phase 2. The ALU function is performed during phase 3 and the ALU output is strobed into the required destination register during phase 4.

Parallel to this activity, the microprogram control (figure 5.3) is accessing the next microinstruction. Again during phase 1, the microcode register outputs are assumed to be settling. During phase 2, the address generation hardware calculates the address of the next microinstruction. At the start of phase 3, this address is clocked into the (microprogram) address register and the line is accessed during the remainder of phase 3 and phase 4.

The four (non-overlapping) timing pulses T1, T2, T3 and T4 associated with the four phases are shown in figure 5.4. These are used by the data logic and control to perform clocking operations during a beat. The data logic uses T4 to clock information into the destination register, while the loading of the microcode register and the (microprogram) address register are synchronised to T1 and T3 respectively.

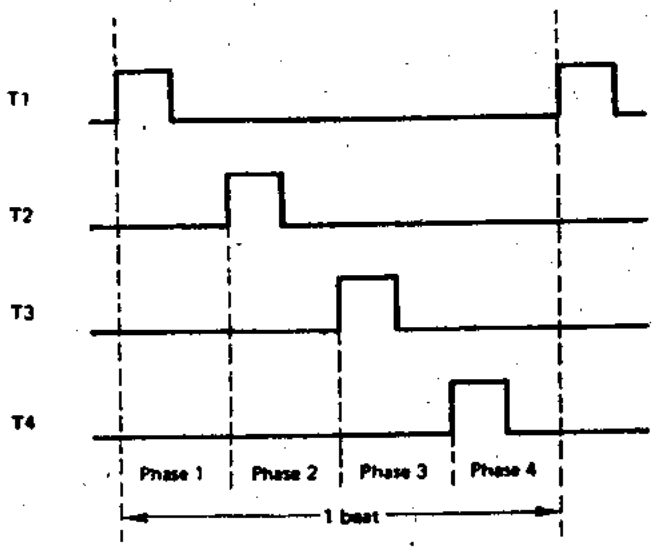


Figure 5.4 Data logic and control timing

A ring counter can be used to form these four pulses, as shown in figure 5.5. Each flip flop shown is a master-slave device and the arrangement is essentially a shift register with the last flip flop in the ring (FF8) connected back to the start (FF1). The counter is initialised by setting FF1 and clearing all other flip flops. Thereafter, master and slave clocks are alternately applied. At each slave clock, the '1' propagates to the output of the next flip flop in the ring and the set flip flop in the current stage is cleared (see table 5.1). Since the output of FF8 propagates to FF1, a '1' continues to circulate indefinitely around the ring of flip flops. Phases can be prolonged by not applying clocks to the counter, and these gated (master and slave) clocks effect the wait conditions within the control.

The outputs of FF1, FF3, FF5 and FF7 form pulses T1, T2, T3 and T4 respectively. Non-overlapping of the timing pulses results from using the outputs of alternate flip flops. In practice, the capacitive loads associated with T1, T3 and T4 would necessitate their buffering prior to being used in the system. As a result, buffer gate chains would be inserted into the shift register data path and these cause a delay in data reaching their succeeding flip flop.

The delay time resulting from buffering T4 is likely to be the largest, as T4 is used extensively throughout the data logic. This time directly contributes to

Table 5.1 Ring counter sequence

FF1	FF2	FF3	FF4	FF5	FF6	FF7	FF8	Pulse
1	0	0	0	0	0	0	0	T1
0	1	0	0	0	0	0	0	
0	0	1	0	0	0	0	0	T2
0	0	0	1	0	0	0	0	
0	0	0	0	1	0	0	0	T3
0	0	0	0	0	1	0	0	
0	0	0	0	0	0	1	0	T4
0	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	0	T1

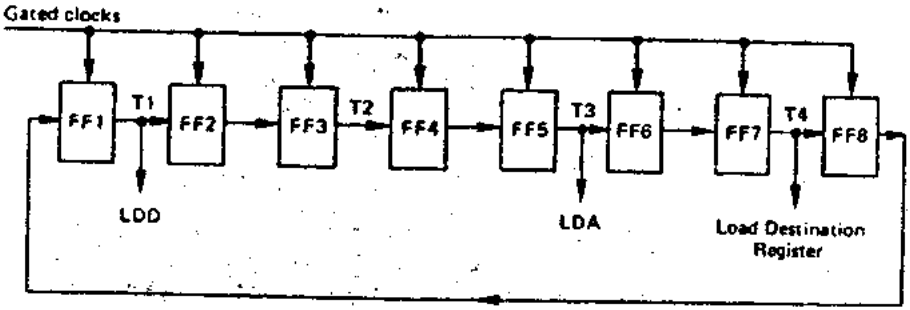


Figure 5.5 Generation of timing pulses

the time between the application of the slave and master clocks, since the master of FF8 cannot be clocked until its input data has arrived. Thus the effect of buffering is to extend the clock period. It therefore follows that by slowing down the clock rate, the configuration of figure 5.5 can always be made to operate correctly with guaranteed non-overlap between the timing pulses, regardless of the delay introduced by capacitive loading.

The adoption of two-phase non-overlapping clocks, $\phi 1$ and $\phi 2$, for the master and slave clocks has previously been discussed in section 4.5. These can be generated off-chip and applied via two input pins. However, since the two phases are normally generated from a common clock source, it is more sensible to derive them on-chip from a single clock waveform (see figure 5.6a).

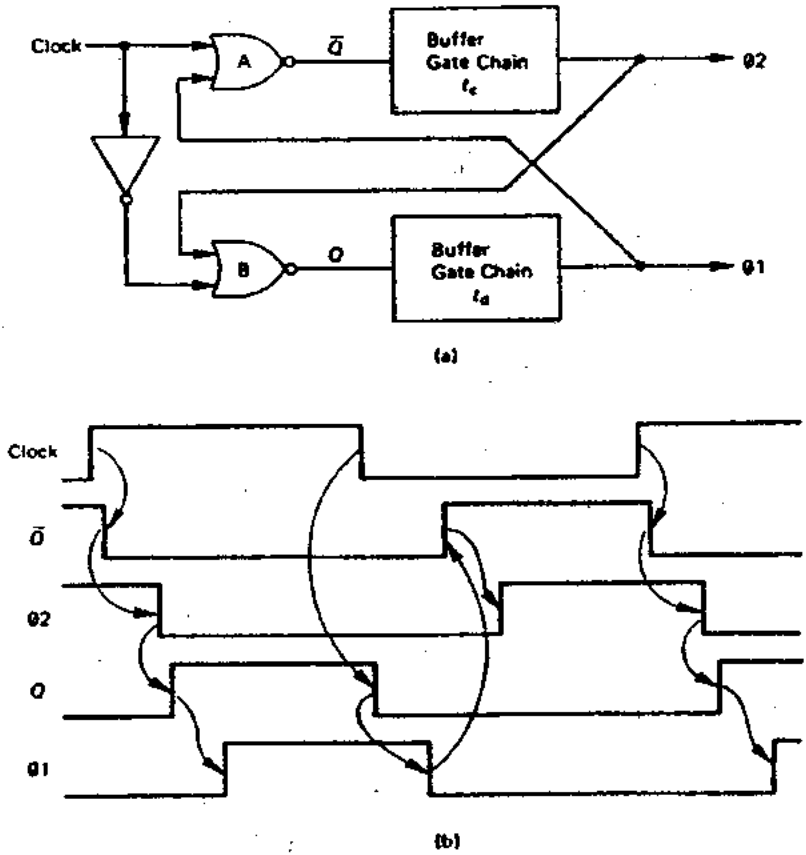


Figure 5.6 Generation of two-phase non-overlapping clocks: (a) logic, (b) timing

The arrangement is essentially that of a Set–Reset flip flop with Clock applied to the set input and Clock to the reset. Conventionally, the feedback connections for such a flip flop arise from connecting $\bar{Q}$ to the input of nor gate B and Q to the input of gate A. However, clock signals need to be buffered to drive their capacitive load. These buffer gate chains are incorporated into the flip flop to ensure that ϕ1 and ϕ2 do not overlap; it can be seen from figure 5.6a that if the feedback to gates A and B were from Q and $\bar{Q}$, and the delays t_d and t_c through the buffer gates forming ϕ1 and ϕ2 differed, then ϕ1 and ϕ2 could overlap. This is avoided by making the feedback connections from the buffer chain outputs to gates A and B.

The timing waveforms are shown in figure 5.6b. When Clock goes high, $\bar{Q}$ is forced low and, after the delay t_c through the non-inverting buffer chain, $\phi 2$ is also forced low. This causes both inputs to nor gate B to be low and thus Q goes high. After the delay t_d through the non-inverting buffer chain, $\phi 1$ goes high. This state is maintained by the feedback until Clock changes. When Clock goes low, the input to nor gate B goes high, forcing Q low. After the delay t_d, $\phi 1$ goes low. As a result, both inputs to gate A are now low and $\bar{Q}$ responds by rising. Thus after a further delay t_c, $\phi 2$ goes high. This output state is maintained until Clock goes high again.

It should be noted that the operation of the flip flop is such that when Clock changes state, the circuit output currently high is forced low, and this causes the other circuit output to go high. The time $\phi 1$ and $\phi 2$ both remain low is equal to the combined delay of a nor gate and its associated buffer chain. Thus, even if buffer gates are not required, gates should be inserted at this point to provide a delay.

$\phi 2$ is derived from Clock going low and its width is dependent upon the time Clock is low. Similarly, the width of $\phi 1$ depends upon the time Clock is high. These widths are reduced by the combined delay through a nor gate and its associated buffer chain. For example, assuming a Clock high and low time of t_p and equal buffer chain delays t_c significantly greater than all other gate delays, then the widths of $\phi 1$ and $\phi 2$ are approximately $t_p - t_c$ with non-overlap times of t_c. It therefore follows that by increasing the clock period, the circuit of figure 5.6a can be guaranteed to produce two-phase non-overlapping clocks, regardless of the buffer chain delays.

The clock-generation circuit of figure 5.6a can also be used to generate two-phase clocks using one of the ring counter outputs as the Clock input, in order, for example, to increment counters during a particular phase.

Clock signals normally fan out to many places on a chip and consequently are associated with long line lengths. It is essential that such lines are implemented in metal to avoid the significant line delays associated with long diffusion or polysilicon lines. The delay down a metal line is negligible and thus no timing skew arises when distributing a clock to different parts of the chip. Running clocks on metal lines also ensures that any non-overlap of clock pulses is preserved.

5.5 Testability

Chip testability is the ability to determine the correct functionality of a design. There is clearly no point in proceeding with a design unless it can be tested, and this section considers design features that allow a system to be tested. This is usually referred to as 'designing for testability'.

The testing of large systems presents many problems. The test time is proportional to the number of gates to the power n, where n is between 2 and 3. Thus the time to test makes it impossible to test exhaustively an entire system. For this reason, if a large system can be partitioned into sub-systems which can be tested, then the test time can be very much reduced. It should be noted that there are likely to be few spare input and output pins available. This results in the use of common input lines to the sub-systems plus common output lines. In addition, an address to specify the sub-system is required.

This structure is exhibited by the processor example of figure 5.1. Here the sub-systems are the registers attached to the highways. The common input is the data on highway A (or B) and the multiplexed output is the data on highway C. Reading and writing to a register for test purposes can most easily be accomplished by use of a microprogram control, such as that shown in figure 5.3.

A request to monitor a register within the system is treated as originating from an external source. This causes a jump to a microprogram routine at the end of the current microcode sequence. By arranging for the register address supplied with the monitor request to replace that specified in the microprogram for highways A, B and/or C, any addressable register in the CPU can be accessed for reading from or writing to. In this way, the data logic can be externally tested. Furthermore, no additional hardware is needed as the necessary circuitry is incorporated in the existing structure.

It should be noted that the structure of figure 5.1, when controlled by a microprogram, is highly suited to testing itself. Here, a test program is stored in the microcode. Once initiated, each part of the data logic addressable by the microcode is exercised and the results checked against those expected by the system. This self-test runs to completion if there are no errors, while a detected error causes a halt with a fault indication.

A system may not be suited to testing by partitioning or by microcode; for example, if many flip flops are either not addressable or not connected to a data highway. In this case, other methods for providing testability have to be employed and three approaches are particularly suitable for chip designs.

The first approach is that of signature analysis. This takes a serial (or multiple) stream of output test data from the system and compresses it using a feedback shift register, so that a much smaller unique signature (that is, a number) is created. Testing is centred around an n-bit feedback shift register which is used to generate a (pseudo) random number sequence of $2^n - 1$ different states before repeating. The randomness of the generated pattern is a result of using particular feedback connections. These are combined by the use of exclusive-or gates and feed a '0' or '1' into the shift register, depending upon its current state.

Figure 5.7 shows a 4-bit number generator with feedback connections from the first and fourth stages; each flip flop shown is a master-slave device. Table 5.2 gives the fifteen states it generates, assuming that the external input *DI* is a '0' and thus has no effect on the sequence. The shift register omits the all-zeros pattern, since this would result in no further changes in the register outputs. It is thus necessary to initialise the register to a non-zero value.

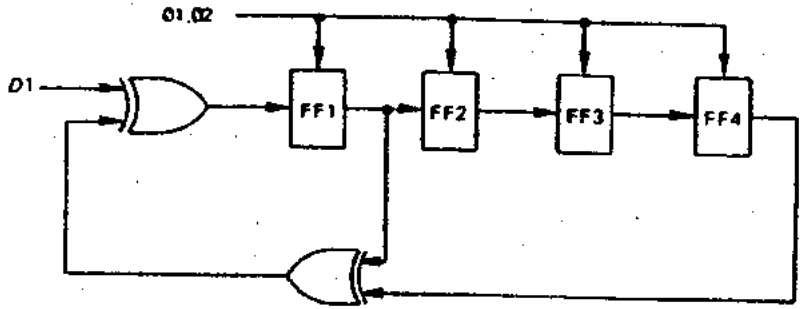


Figure 5.7 Four-bit pseudo random number generator

Table 5.2 Fifteen-state random number sequence

Register state				*Feedback factor*
FF1	*FF2*	*FF3*	*FF4*	*f*
1	1	1	1	0
0	1	1	1	1
1	0	1	1	0
0	1	0	1	1
1	0	1	0	1
1	1	0	1	0
0	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	1
1	0	0	0	1
1	1	0	0	1
1	1	1	0	1

When testing, the system and the number generator are initialised to a known starting state. Thereafter, inputs applied to the system cause the output data stream applied, via *D1*, to the number generator to pass through a pre-determined pa' ern. This pattern is superimposed upon the number generator sequence and, ..er a specific number of clocks, a unique number or signature is held in the ..nber generator. This can be compared with the expected signature to indi-ate a test pass or fail.

The error-detection rate is dependent upon the number of bits n in the generator register and, for a data stream much longer than n, the probability of an undetected error is approximately $1/2^n$. Thus this method can provide very good error coverage even with modest generator register lengths. Clearly, this technique reveals no information concerning the nature or cause of any fault and a correct design is necessary to form the expected signature. Hence this form of testing is most suited to a production environment where the design is proven.

Prototype designs require testability that can initialise the flip flops to a known state and allows them to be monitored thereafter. The scan path approach has both these attributes. Here all flip flops on the chip are reconfigured during test mode to be a single shift register with an external input and output (see figure 5.8a). This implies additional circuitry as all flip flops now need to be (*D*-type) master-slave devices. A multiplexor is also required to select either the normal data path or the adjacent slave flip flop output when in test mode (see figure 5.8b). This logic also causes extra delay to be incurred in the normal data path, which will reduce the operating speed.

Under test, each flip flop in the chip can be loaded with known data via the shift register serial input. Thus the system can be initialised to any known state. Furthermore, this state can be shifted out via the shift register serial output to check that the system is correctly initialised. If the test path can be established then the system switches to normal operation. A specific number of clocks are now applied at the normal operating speed, together with appropriate system inputs. Finally, test mode is re-entered and the flip flop contents read via the serial test path and analysed for correctness.

Thus this testing technique has excellent observability. The generality of this approach makes it a very flexible and powerful aid to fault detection and diagnosis. It also has the advantage of requiring only one pin each for the shift register input and output, plus a pin to control whether the chip operates in normal or test mode.

The shift register configuration of the chip during test mode leads to a considerable time to input and output data in a large system. The approach also requires off-chip generation of input test data and analysis of the test results. These features can be avoided or greatly reduced by combining the scan path and signature analysis techniques so that test data generation and testing is performed on-chip.

This built-in self-test approach uses built-in logic block obervation (BILBO) registers and the elements of a four-bit BILBO register are shown in figure 5.9. Each bit consists of a multiplexor plus a master-slave flip flop, allowing the register to be configured in one of five ways. For normal operations, input 1 of the multiplexor is selected. Here, the input data *D1* to *D4* is clocked into the flip flops and appears on *Q1* to *Q4* respectively.

When input 3 is selected, the register is connected as a shift register with feedback f from stages 1 and 4 to the input of FF1. The register therefore generates a pseudo random number sequence of fifteen patterns. Input 2 of

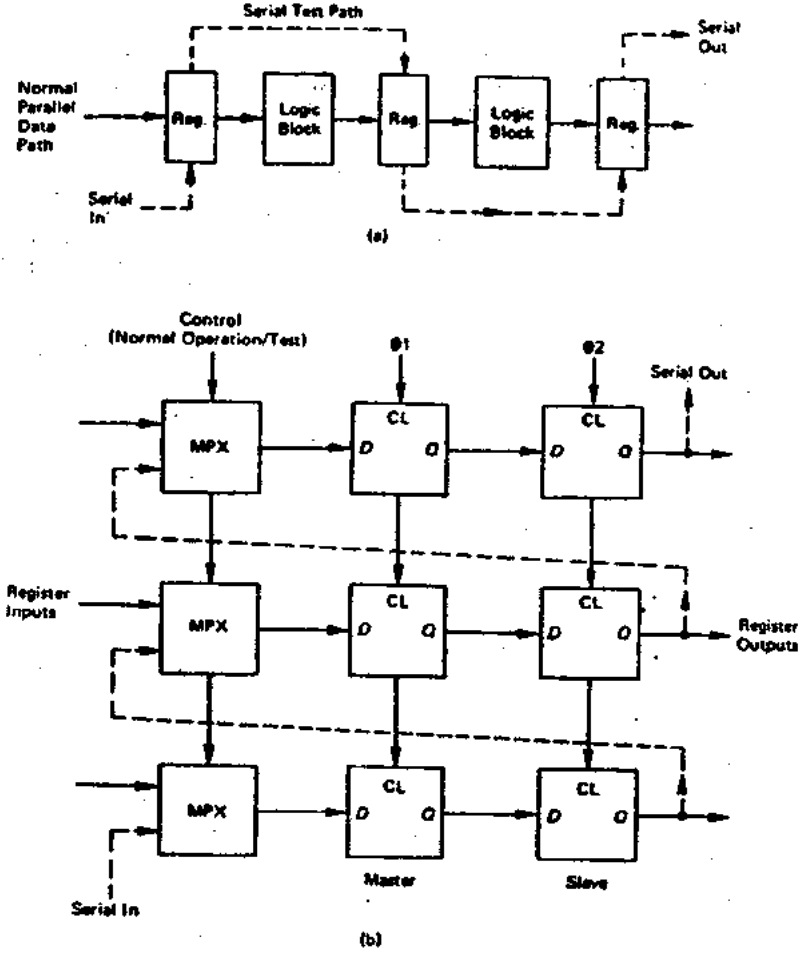


Figure 5.8 Principle of a scan path: (a) system example, (b) logic for three bits of a register

the multiplexor also configures the register as a shift register with feedback f, but here the data inputs $D1$ to $D4$ are superimposed upon the generated pattern sequence using exclusive-or gates; this configuration is that of a multiple input signature register.

Input 4 is selected if the system is in the scan path mode. Here, the register is configured as a shift register. This mode creates a serial test path from the

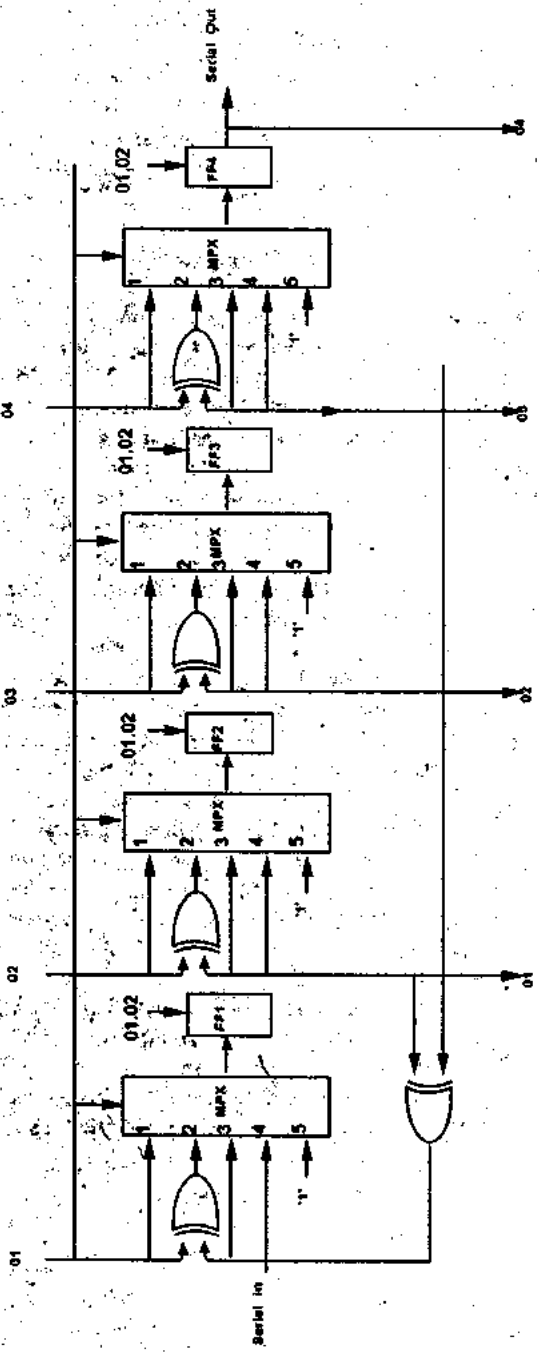


Figure 5.9 A four-bit self-test observation register

input of FF1 to the output of FF4. Finally, system initialisation results from selecting input 5 since all flip flops are set.

When testing, the system is first initialised and this can be checked via the serial test path. The chip then enters signature mode, as shown in figure 5.10a. By setting the multiplexor control bits appropriately, alternate registers in a pipeline generate a random number pattern (denoted Pat.) or hold a signature (denoted Sig.). The number generator registers are effectively isolated from the preceding logic block, and this is shown as a break in the normal data path.

In figure 5.10a, the odd-numbered registers are number generators and the even-numbered registers hold signatures. The number sequence generated by register 1 is applied to logic block 1 and the block outputs are superimposed upon the number sequence generated by register 2. Thus after a number of clocks, register 2 holds a unique signature. Similarly, all other even-numbered registers will contain a unique signature appertaining to the outputs from the preceding logic block. Serial test path mode is then entered, allowing the signatures held by the even-numbered registers to be checked; a correct signature indicates that the preceding (odd-numbered) logic block functions correctly.

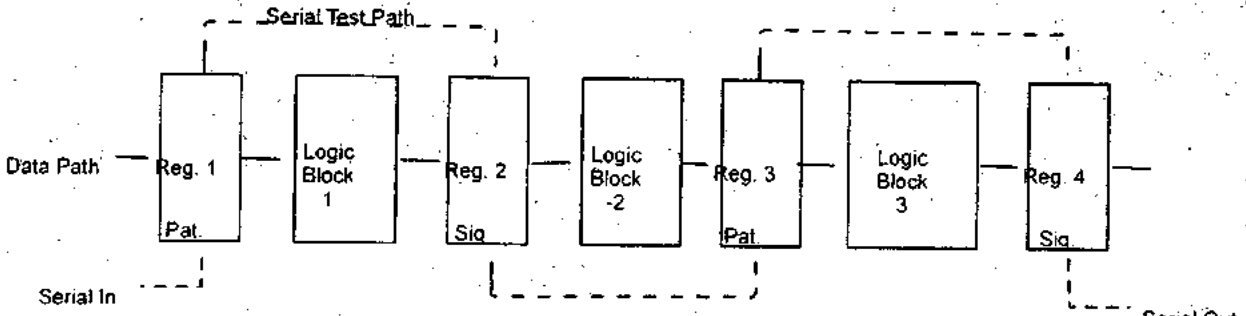
To test the even-numbered logic blocks, the registers are again initialised, checked and signature mode re-entered. This time, the even-numbered registers are configured as number generators and the odd-numbered registers hold signatures (see figure 5.10b). Testing proceeds in a similar manner as before. Thus, random numbers generated by register 2 are operated on by logic block 2 and the block outputs superimposed upon the pattern sequence generated by register 3. Again after a number of clocks, the system switches to serial test path mode to allow the signatures in the odd-numbered registers to be checked for correctness.

With this approach, the test data to be applied is generated internally and is probably more comprehensive than tests devised by the designer. Testing can be relatively fast, as the system is partitioned so that half the logic blocks can be tested in parallel. The checking of test data is simple and yields a pass/fail indication for each logic block; further fault diagnosis of blocks that fail is, of course, possible via the serial test path.

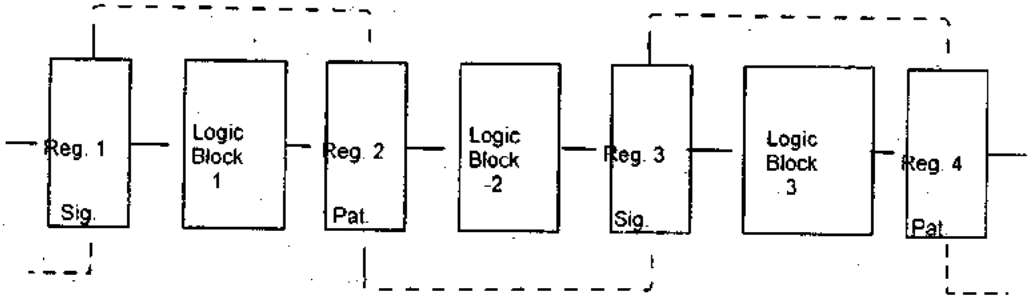
There is a considerable silicon area overhead associated with BILBO registers. As a result, it is likely that only parts of an architecture suited to this approach (such as a data pipeline) would be implemented in this way, with other techniques being used elsewhere. Nevertheless, the self-test approach would seem to offer a viable solution to the problems of testing VLSI designs.

5.6 Further Reading

R. A. Frohwerke, 'Signature analysis: a new digital field service method', *Hewlett-Packard Journal*, May (1977) pp. 2–8.



(a)



(b)

Figure 5.10 Logic self-testing: (a) testing odd-numbered logic blocks, (b) testing even-numbered logic blocks

P. B. Lomas, 'Ordercode Implementation in a High Performance Mini-Computer', M.Sc. Thesis, Manchester University, 1978.

C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

T. W. Williams and K. P. Parker, 'Design for testability – a survey', *IEEE Trans. Comput.*, C-31, No. 1 (1982) pp. 2-15.

6 A Design Example

The best way of learning about chip design is by 'doing'. With this in mind, this chapter outlines a first attempt at producing a small, conservatively designed, 'handcrafted' full custom chip. It is hoped that this will give a comprehensive view of the design process.

The correct implementation of any design requires that great care is taken at all stages of the design hierarchy; it is essential that thorough checking (and double checking) for errors be performed at each stage and that any available software aids be used. This carefulness and thoroughness are of particular importance when the design (or part of it) is to be integrated. Here, any design errors in any stage will cause the chip to malfunction and the fault cannot, of course, be corrected on the manufactured chip.

Incorrectly designed chips are costly in terms of both money and time. Hence the design emphasis at all levels must be to implement a correctly working chip at the first attempt. For this reason, the design proceeds systematically through the levels of the design hierarchy. In addition, the approach down to the logic design stage is to successively partition the problem into smaller and more manageable sections.

6.1 System Specification

The purpose of the chip is to incorporate controllability and observability into a register so that the monitoring of its states can proceed in parallel with the register's normal operation. Since many registers need to increment and decrement, the register is to be implemented as an up/down counter. The ability to control and monitor the counter's operation is achieved by the addition of a shift register which operates independently of the counter.

The system specification of the chip is shown in figure 6.1. The number of counter and shift register bits, n, which can be accommodated on the chip is not known at this stage but is normally chosen to be a multiple of 4. If the design is approached and discussed in terms of a 4-bit device, then a larger device is just a matter of duplicating the data architecture.

To observe the counter's state, its contents are transferred into the shift register. This can then be shifted out serially while the counter continues to operate normally. The shift register incorporates a two-way shift path to aid

fault diagnosis on the shift register; a two-way path allows the position of a break in this path to be determined.

Controllability is achieved by loading the shift register via one of its serial paths and then transferring its contents to the counter. This feature requires the addition of a multiplexor on each input bit of the counter to select between the normal input data and the shift register data. Thus the addition of a shift register and associated multiplexors incorporates excellent monitoring facilities into the counter. Furthermore, since all sequential elements on the chip can be initialised to a known state and monitored thereafter, the design is fully testable. These features also enable testing and fault diagnosis of prototype systems incorporating this counter design.

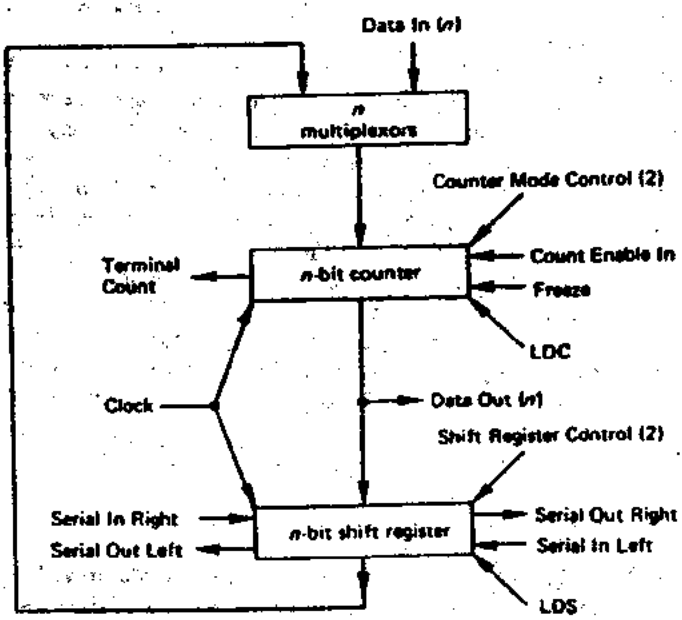


Figure 6.1 System specification

The architecture can also be used for the conversion of a serial bit stream to parallel data, by loading up the shift register with the serial data and then transferring it to the counter. Alternatively, a parallel-to-serial data conversion is obtained by transferring the counter contents to the shift register and then shifting data out serially from this. Thus the chip architecture can be regarded as being general purpose.

It can be seen from figure 6.1 that a large number of control lines are needed (ten). This reflects the versatile nature of the chip operation. The binary counter is cascadable; this allows it to be used in conjunction with other similar chips to extend the counting range. In this case, incrementing or decrementing is only performed on a chip when all bits of lesser significance than it in a count chain are either all ones (if counting up) or all zeros (if counting down). This is indicated by the Count Enable In signal, which is used (in conjunction with the counter mode control bits) to enable counting on a chip.

Similarly, the Terminal Count output indicates that the counter bits of a chip are either all ones if incrementing or all zeros if decrementing. This output signal is used by chips of greater significance in a count chain for the formation of their Count Enable In signal.

All operations on the chip are synchronised to the Clock. Normal counter operation is specified by two mode control bits. These either cause the existing data to be incremented or decremented (if the Count Enable In input is also active), or held, or the external data DI1 to DI*n* to be loaded. Similarly, two bits determine the shift register function. Here, the shift register can be loaded from the counter or the existing contents held, shifted right or shifted left.

The LDC signal overrides the counter mode control bits but allows normal shift register operations; it causes the shift register contents to be transferred to the counter. Similarly, the LDS signal transfers the counter contents to the shift register; it overrides the shift register control bits but allows counter operations. If both the LDC and LDS signals are present, the counter and shift register contents are swapped. Finally, the Freeze signal is used to inhibit all counter operations and thus overrides the LDC and the counter mode control bits; it has no effect on the shift register actions.

6.2 Architecture

In this example, the functional blocks of the system architecture can be directly derived from the system specification. Clearly, the design is centred around the counter and shift register.

Table 6.1 shows the sixteen states of a 4-bit binary counter when incrementing and decrementing; Q1c is the (slave) flip flop output of the counter's most significant bit and Q4c is its least significant bit. It can be seen that the least significant bit changes at each count. When incrementing, a bit only changes state on the next count if all bits of lesser significance are ones. Similarly, if decrementing, a bit only changes state on the next count if all bits of lesser significance are zeros. Thus when counting, the next state is dependent upon the existing state. The shift register's next state is also dependent upon its current state (when shifting). Hence, the counter and shift register have to be organised

on a master–slave basis and require two-phase non-overlapping clocks. These are provided by the clock-generation logic of figure 6.2, which forms the master and slave clocks, $\phi 1$ and $\phi 2$, from a single Clock waveform.

Table 6.1 Four-bit binary counter sequence

	Data out					
	Q1c	*Q2c*	*Q3c*	*Q4c*		*Count*
	0	0	0	0		0
	0	0	0	1		1
	0	0	1	0		2
	0	0	1	1		3
	0	1	0	0		4
	0	1	0	1		5
	0	1	1	0		6
	0	1	1	1		7
Count	1	0	0	0	Count	8
up	1	0	0	1	down	9
	1	0	1	0		10
	1	0	1	1		11
	1	1	0	0		12
	1	1	0	1		13
	1	1	1	0		14
	1	1	1	1		15

Although figure 6.1 shows information loaded into the counter arising just from the input data or the shift register, the counting requirement necessitates that it also be derived from the existing counter outputs. It should be noted that the counter's true (slave) flip flop outputs are required when incrementing and the inverse outputs when decrementing. These are used to change the state of bits, rather than directly loading in information as in the case of the input data or shift register contents.

The formation of the data to be clocked into the counter is performed by the counter's data-selection circuitry. The information source selected by this circuitry is determined by control signals from the counter control logic; these in turn are derived by the control logic from input signals to the chip appertaining to the counter operation. The terminal count logic is activated when counting nd indicates when the count is all ones if incrementing or all zeros if decre- nting.

ta-selection logic is also required for the shift register input since its input es from the counter contents, or an adjacent bit of greater significance right), or an adjacent bit of lesser significance (if shifting left). This s determined by the shift register's control logic.

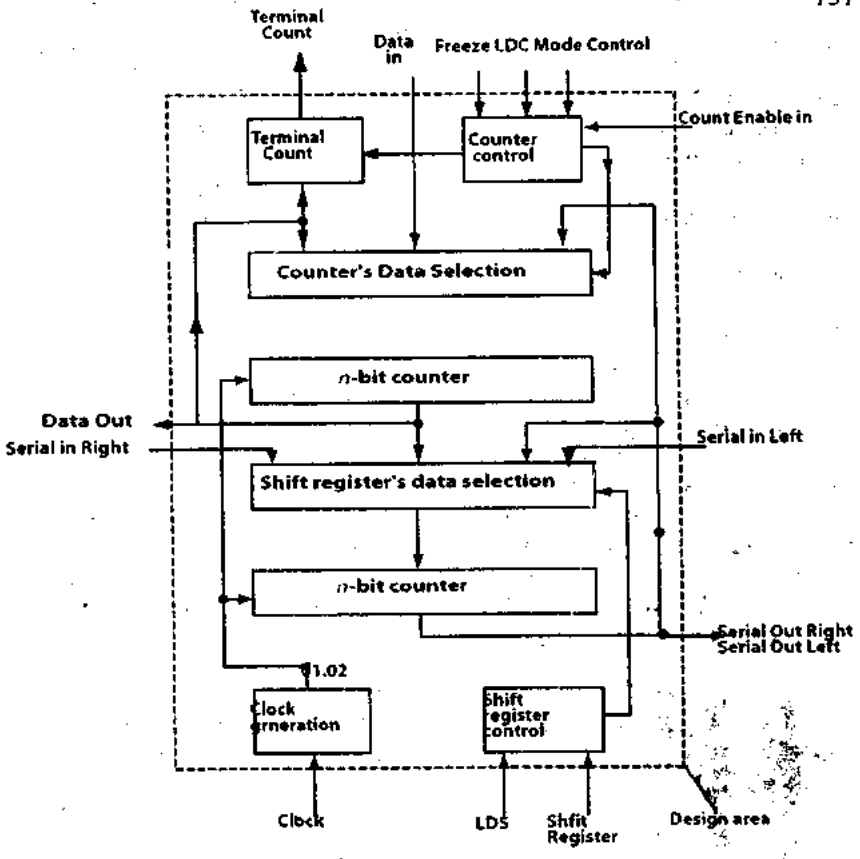


Figure 6.2 Floor plan of system architecture

Figure 6.2 shows the logic blocks of the system arranged as a floor plan which reflects the flow of date through the chip. This plan indicates the ideal arrangement of the system blocks as their area is not known at this stage. It can be seen that including the power and ground pins, 24 pins- am required for a 4-bit chip and 32 pins for an 8-bit chip. 7bit pads for these signals are placed around the edge of the available chip area (we figure 6.10).

At the commencement of a design, it is usual to know which fabrication line is to be used for manufacture. Furthermore, it is likely that specific chip sizes are associated with the line. For this example, it is assumed that a chip size of 4 mm x 4 mm is available from a 6 um NMOS process which supports a single metal layer and buried contacts. After allowing for the manufacturer's para-metric test circuits, a chip area of 3.5 mm x 4mm remains. The area of an out-put pad is greater than the area of all other pad types (input, power- and ground) and for an NMOS process with a g of 3um, an output pad is typically 0.35 mm

wide and 0.5 mm long. This leaves an inner design area of 2.5 mm x 3 mm to accommodate the logic; the amount of logic that can be placed in this design area dictates the number of bits which can be incorporated on the chip.

Seven output or other pad types can be comfortably accommodated down a 2.5 mm side while eight can be placed along a 3 mm side. This gives a total pad count of approximately 30 pins, indicating that control signals may have to be encoded externally and decoded on-chip to drive an 8-bit chip.

6.3 Logic Design

The functional blocks of the architecture diagram have now to be translated into logic elements. At this point, there are many ways of implementing a design. A design can be judged to be successful if it performs the specified task at the required speed, conforms to any interfacing requirements and is completed within the design schedule time. Thus the decisions made for this example at the logic design level reflect only one of the approaches which can be taken.

The preceding discussion has shown that the major blocks (in terms of circuitry) are the counter and shift register plus their data selection. The remaining functional blocks are small in comparison with these. The type of clocked master-slave flip flop used in the counter and shift register affects the design of the data-selection logic and hence needs to be decided first.

Existing data in the counter and shift register may have to be held indefinitely. Therefore, suitable flip flop types are the clocked static devices) that is, the *D*-type or *J-K*) or a dynamic *D*-type flip flop with feedback applied so that the data can be maintained. The basis of such a dynamic flip flop is shown in figure 6.3.

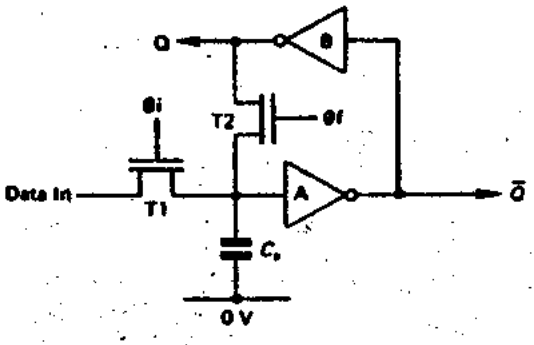


Figure 6.3 A dynamic flip flop with feedback

φi and φf are non-overlapping clocks and except for a short non-overlapping time, φf is active when φi is inactive, and vice versa. The pass transistor T1 and the inverter A form a dynamic flip flop so that during φi, the inverse of Data In

is transferred to $\overline{Q}$; T2 is off at this time. When ϕi is removed, T1 turns off and the state at $\overline{Q}$ is maintained by the charge on C_s. After the non-overlap time, ϕf is applied, turning the pass transistor T2 on. This completes the feedback round the loop from the output of gate A to its input. This reinforces the existing charge on C_s and hence the current state of $\overline{Q}$. In this way, the flip flop state can be indefinitely maintained between applications of ϕi; the device thus exhibits static behaviour.

Two *D*-type flip flops per bit are required for the counter and shift register. It should be noted that although the dynamic flip flop of figure 6.3 requires less area than the static *D*-type of figure 4.11, extra logic is needed to generate the feedback clocks ϕf_m and ϕf_s for the master and slave flip flops.

In order to hold or toggle a bit on particular clock pulses when using (dynamic or static) *D*-type flip flops, it is necessary to apply a gated clock to the master device. This gating is different for each bit of the counter, introducing some clock skew across it. The gated clocks associated with *D*-type devices would necessitate the addition of two functional blocks on the architecture diagram of figure 6.2 for their formation.

A *J-K* flip flop does not have these features since holding, loading or toggling can be achieved by simply driving the *J* and *K* inputs appropriately. It follows that it is not necessary to gate the clock. Thus no clock formation logic is required and two-phase non-overlapping clocks, ϕ1 and ϕ2, can be directly applied to the master and slave flip flops of the *J-K* device. The flexibility of operation of a *J-K* flip flop best meets the requirements of the design and for this reason this type of flip flop is used.

The design of the data-selection logic for the *J* and *K* inputs of the master flip flops can now be discussed. Depending upon the control signals to the data-selection logic, the data in a master flip flop of the counter is either held (*J* = '0', *K* = '0'), loaded, or toggled (*J* = '1', *K* = '1') on ϕ1. Consider loading data into a master flip flop. If the data to be loaded is a '1', the desired effect can be obtained if *J* is a '1' and *K* is a '0'. If the existing data is a '1', then loading a '1' causes no change of state and the same effect is obtained if *J* and *K* are both '0'. If, however, the existing data is a '0', then loading a '1' causes a change of state and this can be achieved by taking *J* and *K* to '1'.

Similarly, a '0' can be loaded by taking *J* and *K* to a '1' if the existing state is a '1', and by taking *J* and *K* to '0' if the current state is '0'. Thus if the data to be loaded is the same as the existing data, loading can be effected with *J* and *K* both '0', while if the data to be loaded differs from that currently held then *J* and *K* must be '1' to load.

All operations can be performed on the counter with *J* and *K* equal. In this case, there is no need for the separate formation of the *J* and *K* inputs for each bit. This is shown in figure 6.4 for the most significant bit of a 4-bit counter and shift register.

Ignoring the signals in brackets which relate to the data selection for the most significant bit of the shift register, the loading of data into the counter from the

external data (DI1) is enabled if control signal C1c (from the counter control logic) is present. Here, the data to be loaded is compared with the currently held counter data (Q1c) by and gates A and B. If DI1 is equal to Q1c, then both and gate outputs remain low, so J and K are low. However, if DI1 and Q1c are not equal, then the output of either gate A or B goes high which in turn causes J and K to be high. Similarly, and gates C and D compare the most significant shift register bit (Q1s) with the counter contents if control signal C2c is present. Effectively, and gates A and B or C and D perform an exclusive-or operation if enabled.

And gates E and F determine whether a bit toggles when counting up or down respectively. These gates inspect the true and inverse state of the counter bits of lesser significance on the chip and are enabled by C3c and C4c respectively. Thus, for example, if C3c is active, J and K for the most significant bit are '1' only if and gate E's output is high; this occurs when Q2c to Q4c are high, indicating an existing count state of seven or fifteen. Only one of the four control signals can be present at any time and if none is activated, then J and K are '0' and a hold operation is obtained on $\phi 1$.

The J and K inputs for the master flip flops of the shift register can be formed in a similar manner. Here, data is loaded in from one of three sources by comparing it with the existing shift register state. Referring to figure 6.4, and gates A and B, C and D or E and F perform an exclusive-or function if enabled by control signal C1s, C2s or C3s respectively; these control signals are formed by the shift register control logic. C1s selects the counter data for loading while C2s selects data from the adjacent bit of the shift register on the left (shift right) and C3s selects the adjacent bit on the right (shift left); thus in figure 6.4, Q1s is compared with Q1c, Serial In Right (SIR) and Q2s respectively. Again the absence of all three control signals causes the execution of a hold operation on $\phi 1$.

The formation of the J and K inputs for all other bits of the counter and shift register is similar to that shown in figure 6.4. This suggests that a common element be designed which can then be used $2n$ times in the design.

The remaining functional blocks of the system architecture can be dealt with more briefly. The counter and shift register control logic can easily be derived from the equations relating the chip input signals to the control signals for the data-selection logic. These are given in table 6.2 and the relationships shown are directly derived from the system specification. Thus for example, the enabled count up and count down control signals, C3c and C4c, are only activated when Count Enable In is present, and the Freeze signal inhibits all counter control signals. It can be seen from table 6.2 that only primitive logic functions, such as nand, nor, and-or-not and not (invert), are required for these sections. These blocks can be implemented in random logic and/or a transistor array.

The Terminal Count signal indicates when the count has reached the all-ones state when incrementing and all-zeros when decrementing. An n-input and gate detects the former case while an n-input nor gate indicates the latter state. The

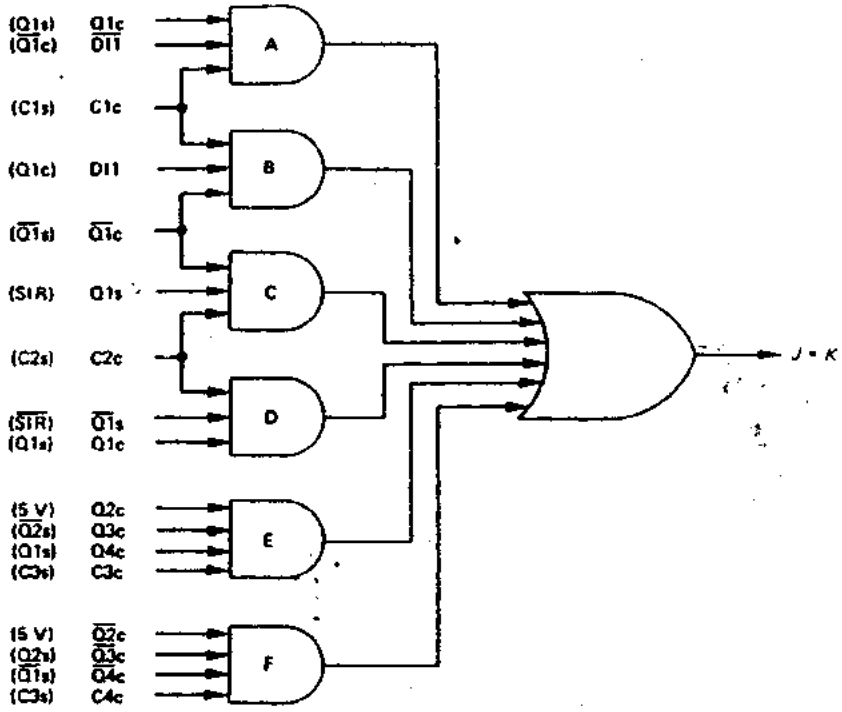


Figure 6.4 Data-selection logic for most significant bit of counter and shift register

outputs from these gates are then combined with the count up and count down signals respectively (CMC0, CMC1 and CMC0, CMC1) in an and-or-not element which is then inverted to produce the Terminal Count output. Again only a small number of primitive logic functions are required to implement this logic.

Finally, the clock-generation logic to produce $\phi 1$ and $\phi 2$ from a single clock waveform is as described in section 5.4 with the logic shown in figure 5.6. However, a single superbuffer is used rather than a buffer gate chain, as the master and slave clock loading is relatively small. Apart from the superbuffers, only 2-input nor gates plus inverters are required.

The logic elements to be used in a design are dependent upon the details of the circuit implementation. Thus, in effect, the logic diagram and logic simulation cannot be completed until the circuit design phase. This illustrates that in a practical design, design work on different levels overlaps and significantly interacts.

Table 6.2 Formation of the data selection control signals

Counter Control logic block

(chip) Inputs: Freeze
LDC
Count Enable In (Co En In)
Counter Mode Control – CMC0 CMC1 – decoded as

0	0	hold
0	1	load DI1 → DI4
1	0	count up
1	1	count down

Outputs (to counter's data selection logic):
C1c = load DI1 to DI4 = Goc.$\overline{\text{CMC0}}$.CMC1
C2c = load shift register contents = $\overline{\text{Freeze}}$.LDC
C3c = enabled count up = Goc.Co En In.CMC0.$\overline{\text{CMC1}}$
C4c = enabled count down = Goc.Co En In.CMC0.CMC1
where Goc = $\overline{\text{Freeze+LDC}}$

Shift Register Control logic block

(chip) Inputs: LDS
Shift Register Control – SMC0 SMC1 – decoded as

0	0	hold
0	1	load counter
1	0	shift right
1	1	shift left

Outputs (to shift register's data selection logic):
C1s = load counter contents = LDS+$\overline{\text{SMC0}}$. SMC1
C2s = shift right = $\overline{\text{LDS}}$.SMC0.$\overline{\text{SMC1}}$
C3s = shift left = $\overline{\text{LDS}}$.SMC0.SMC1

6.4 Circuit Design

Again, there are many ways of implementing the logic and only one particular approach will be described. The counter and shift register contain sequential elements. These are the *J-K* flip flops described in section 4.5 and implemented exactly as shown in figure 4.13. The remaining functional blocks contain combinational logic. Here, random logic and PLAs are preferred to pass transistor arrays, because of their superior speed. Again, the majority of the combinational

logic resides in the data-selection circuitry and hence its implementation represents the most important design decision at this level.

Considering figure 6.4, this logic is unsuitable for implementation in a PLA because many of the inputs are only used by one and gate and the others are only common to two gates. A 16 x 6 AND plane would be required to implement the most significant counter bit, of which only 20 transistor positions would be used. The inefficiency of such a sparse array would be improved by folding but, even so, this will only halve the AND plane size at best.

Thus it is more sensible to design this function in random logic. Although the logic can be designed in three levels of gating using seven nor gates and an inverter, greater area efficiency is obtained by designing a complex function to directly produce the inverse of the required J and K signals; this is then inverted to obtain the true phase. Figure 6.5 shows the stick diagram of the $\overline{J}$-$\overline{K}$ formation circuit directly derived from the logic diagram of figure 6.4. Each branch performs an and function while the connection of the branches to V_{out} forms a nor of these products.

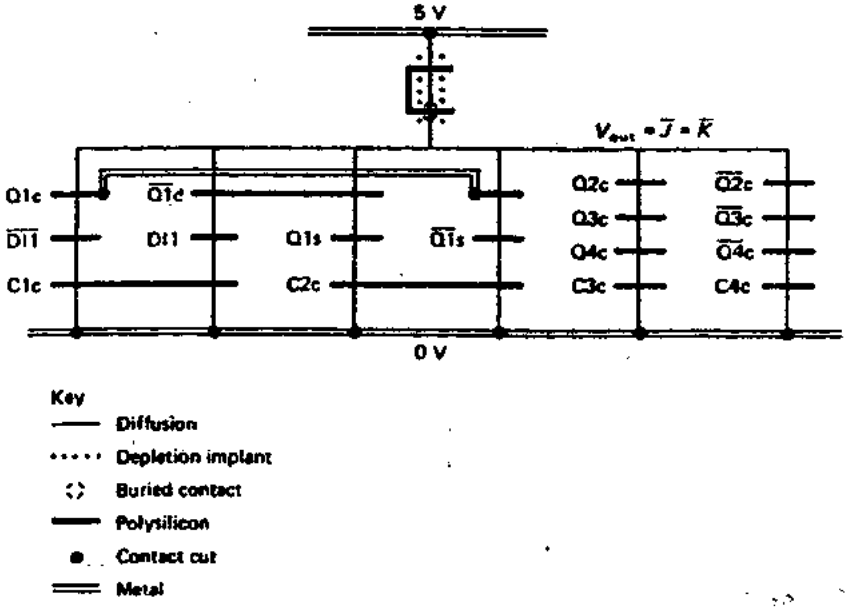


Figure 6.5 Stick diagram of $\overline{J}$-$\overline{K}$ formation

The counter control and shift register control can likewise be implemented with random logic or an AND plane. The shift register control signals C1s to C3s and the counter control signals C1c, C3c and C4c can be efficiently imple-

mented in two 6 x 3 AND planes but the remaining logic in these blocks requires a random logic approach. The clock-generation and terminal count logic both require a random logic approach. In view of the fact that there is little use for a PLA approach in this design, it seems reasonable to use random logic throughout the design.

Clearly, while a large number of different random logic functions can be designed and implemented, the effort and managability of a design is facilitated by using just a few function types. Furthermore, the and-or-not, nand, nor and invert functions can be implemented in one level of gating, whereas the and, or and non-inverting buffer functions require two levels of gating. Bearing this in mind, an inspection of the logic functions performed in each logic block shows that only eight different logic elements are required to efficiently implement the entire design; these cells are listed in table 6.3.

Table 6.3 Logic elements

	Cell width (μm)	*Cell height (μm)*	*No. used in 4-bit design*
not	48	87	14
2 × not	87	87	14
2-input nor	75	84	3
4-input nand	81	123	8
3-input and-or-not	105	120	2
inverting superbuffer	105	75	25
$\overline{J}$-$\overline{K}$ formation	216	150	8
J-K flip flop	219	273	8

It can be seen that apart from the $\overline{J}$-$\overline{K}$ formation and the J-K flip flop cells, all other elements perform primitive logic functions which have been described elsewhere in this text. (See section 2.9 for the inverter design, section 4.2 for nand, nor and and-or-not gates, and section 2.16 for a superbuffer.) The 2 x not cell consists of two inverters and produces the true and inverse of a signal. This function can be implemented with two inverters but its frequent use made it desirable to design an element occupying less area.

A cell approach introduces some inefficiency in terms of the number of logic elements required; any repeated inefficiencies that arise can and should be remedied by the introduction of additional cell designs. Nevertheless, the advantage of a cell approach is that only a few circuit elements have to be designed and these are specific to a design. Once proven and characterised, the circuit can be regarded in purely logical terms. This simplifies the design thereafter, as it reduces the task to the interconnection of logical elements.

Having determined the logic elements for the system, the logic diagram can now be finalised. Figure 6.6 shows the data-selection plus the counter and shift register logic for the most significant of four bits. This logic is repeated for each bit on the chip.

The fan-out load of points can be estimated relative to driving an inverter which counts as a load of one. Thus a 4-input nand gate having a gate width four times that of the inverter represents a load of four on a driving gate. Considering the load on Q1c, figure 6.6 shows that Q1c is an input to two 3-input and gates (load=6) in the counter's $\overline{J}$-$\overline{K}$ formation and is input to one 3-input and gate (load=3) in the shift register's $\overline{J}$-$\overline{K}$ formation. In addition, the signal is input to a 4-input nand gate (load=4) in the terminal count logic. This gives an estimated total load of 13 on this point and necessitates that the signal be buffered (by an inverting superbuffer).

In general, any point loaded with a fan out in excess of 10 is buffered. As a result, all true and inverse outputs of the counter and shift register flip flops are buffered plus the counter control signals, C1c to C4c, and the shift register control signals, C1s to C3s.

The logic can now be simulated by describing the system as a set of interconnected logic elements. Appropriate input signals are applied and the resulting output responses are compared with expected responses, so that any differences result in an error indication. The accessibility of all sequential elements and the relatively small size of this design enables exhaustive testing of the system.

Functional tests can verify that the logic meets the system specification. These tests can be used on the fabricated chips later to determine working devices. If typical logic element delays are included in the simulation, then the approximate operating speed emerges and any potential timing problems within the design can be identified and corrected.

A particularly valuable feature arises if the logic description can be compared with the circuit and/or layout it generates for discrepancies; this helps to ensure consistency and integrity throughout the design process. When such a check is not available, the logic simulation only validates the design down to the logic design level of the hierarchy.

Circuit simulation is necessary (and useful) to verify the logical operation of cells and characterise them. While most of the details for this simulation can be derived from the circuit diagram, the capacitance of points in a cell needs to be estimated to give an indication of the speed of operation; unfortunately this calculation requires the geometric layout. Again this illustrates the overlapping nature of the design activity at the different levels in the hierarchy.

Unfortunately, circuit simulation of large parts of the design is often not practical because of the time to simulate and the amount of memory required. Thus while it is feasible and desirable to check the logic functionality of cells, it is not realistic to check the circuit design against the system specification. Likewise, validated circuits often cannot be checked for consistency with the layouts generated at the next level down. Thus circuit simulation only checks the correctness of the design at the circuit level.

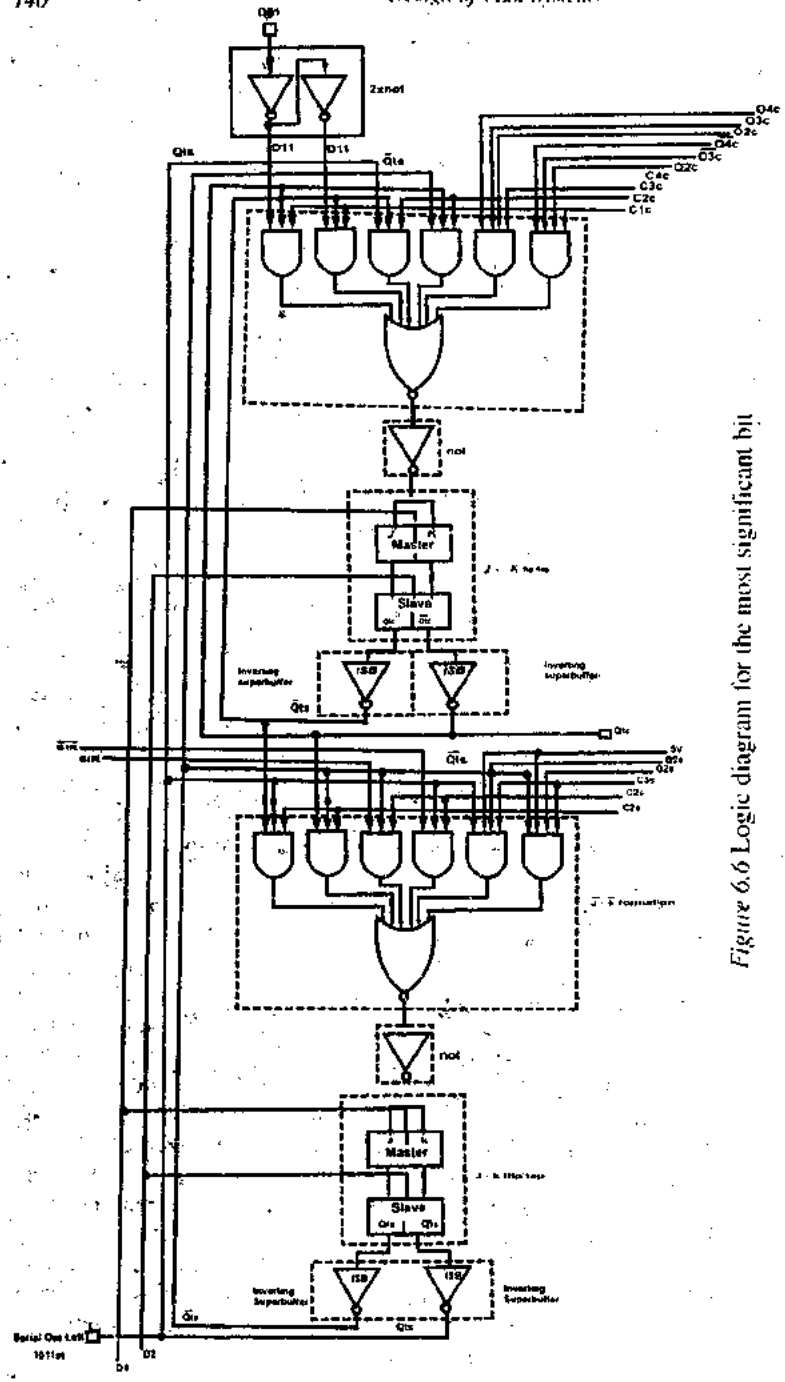


Figure 6.6 Logic diagram for the most significant bit

6.5 Layout, Placement and Interconnections

The adoption of stick diagrams for the cell designs facilitates the layout stage since they only require fleshing out. Again, it is worth while concentrating on minimising the area of the *J-K* flip flop and the $\overline{J}$-$\overline{K}$ formation circuit, since these cells are much larger than any other.

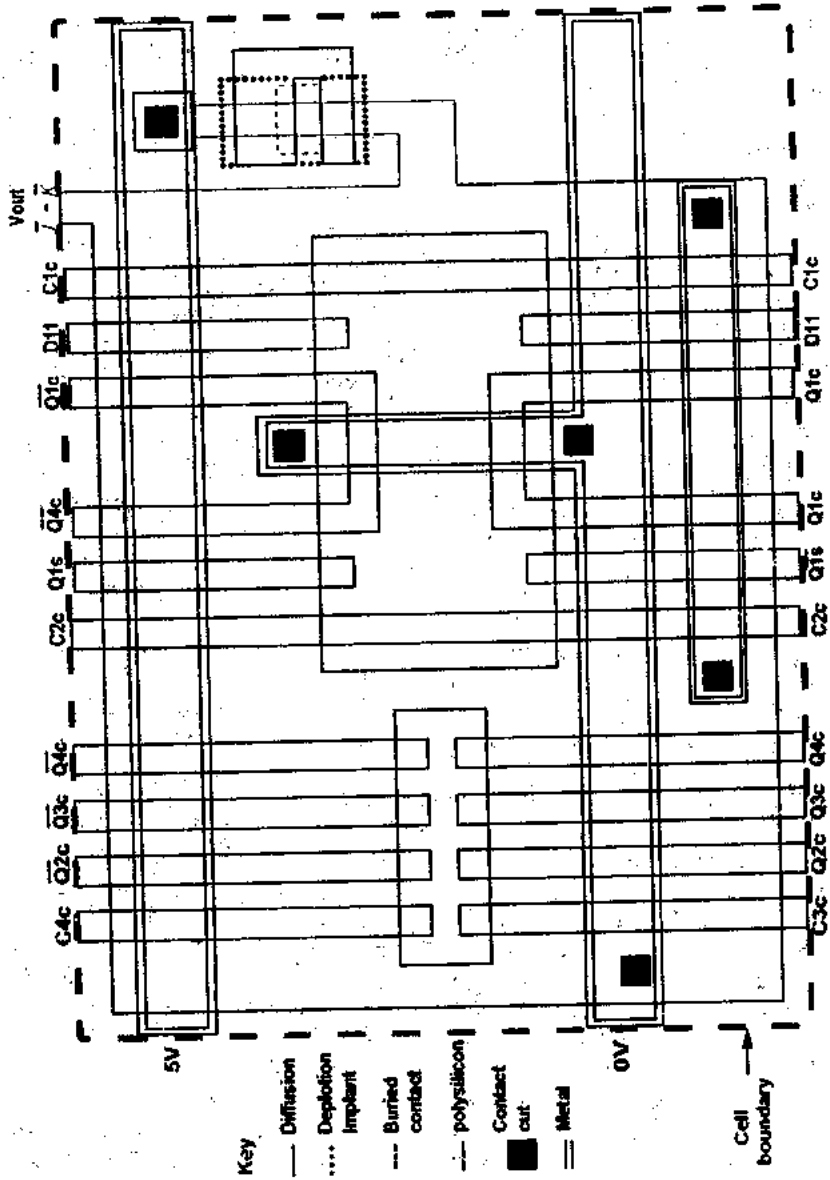
These particular cells require more than one attempt at the layout to successively reduce wasted area and eliminate unnecessary contact cuts. Figure 6.7 shows the layout of the simpler of these two cells, the $\overline{J}$-$\overline{K}$ formation element. Careful comparison with the stick diagram of figure 6.5 shows that they are equivalent. The inverter ratio used throughout the cell is 5/1. Thus a logic '0' voltage of 0.35 V has been accepted in the interests of reducing the cell area and the input and output capacitance.

It should be noted that there is a notional rectangular boundary surrounding the cell. The cell inputs and output plus power and ground are routed to this boundary in order to provide the cell with connection points. Taking the bottom left corner of the boundary as the origin, the cell can be regarded as a rectangular box with connection points at known positions. Furthermore, the cells can be abutted without infringing the layout rules if connection points are compatible. Once designed, no other interconnection lines are allowed through the cell. Again, this allows the cell to be treated in logical terms, removing the necessity to be concerned about the circuit or the layout within. For this reason, these design features have been used in all cell designs.

It should be noted that the cell has a 5 V and a 0 V rail running horizontally through it near the top and the bottom of the cell. Again, this is adopted for all cells as it aids the distribution of power and ground throughout the design. It can also be seen that the inputs and output run vertically to connection points at the top and the bottom of the cell. Again, this convention is adopted for all cells. All cell inputs run vertically in polysilicon. Cell outputs also run vertically, mostly in polysilicon and the remainder in diffusion. In addition, where the cell area permits, the output or input of primitive cells is routed to the top and bottom of the cell to facilitate interconnecting.

The conventions adopted for the cell designs imposed some uniformity on the layouts and table 6.3 lists the cell boundary sizes for a λ of 3 μm. A check should be made at this point to verify that layouts do not infringe the design rules of the fabrication line (section 3.7); this does not, of course, verify that the layout is correct from a circuit viewpoint.

Cell placement follows. This is based on the system architecture floor plan (figure 6.2), since this reflects the flow of data through the system and should therefore lead to the shortest interconnections. It is sensible in terms of power distribution and interconnections to organise the logic in rows. Thus the placement is ideally organised as six rows of logic, where each row corresponds to a level shown on the system architecture floor plan.


Vout
C1c
D11
Q1c
Q4c
Q1s
C2c
Q4c
Q3c
Q2c
C4c
5V
0V
Key
Diffusion
Depletion implant
Buried contact
polysilicon
Contact cut
Metal
Cell boundary

It is essential that adequate space between cells is left to cater for interconnections. Thus a gap is left for this purpose between rows. Some cells in a row abut. However, gaps between cells in a row are left when there is clearly to be tracking from a cell to rows above or below it. The different widths of cells makes it difficult to form cell gaps at similar positions in the rows, introducing deviations in some of the vertical tracking.

Placement is performed using a co-ordinate grid of the design area to place cells at a particular position. Figure 6.8 shows a scaled drawing of the placement of elements for the counter control logic; this occupies the top right-hand corner of the design area. All co-ordinates shown are in μm and assume an origin at the bottom left-hand corner of the design area.

Again, more than one attempt at the placement is normally necessary to create adequate inter-row and inter-cell gaps. Thus an aim of the placement phase is to facilitate the interconnection of cells. With this in mind, figure 6.8 shows that cells have been rotated 180°, so that the positions of inputs are in general along the top of the cell with the output(s) along the bottom; this reflects the flow of data from the top of the design area to the bottom. Even so, cells might have to be moved later at the interconnection stage to fit in tracking in the congested areas.

The size of the terminal count and the counter control logic is too large to fit into a row width of 2.5 mm and these blocks therefore occupy two rows. The counter flip flops and their associated buffers are allocated to a row, as are the shift register flip flops and their buffers. The relative height of a flip flop (273 μm) and an inverting superbuffer (75 μm) allows the buffers for a flip flop to be placed one above the other in the row; this increases the inter-cell gap for vertical tracking.

Even so, table 6.3 shows that each flip flop and its buffers occupies a width of 0.32 mm. This indicates that while four bits can be accommodated on a row, leaving 1.2 mm (out of a width of 2.5 mm) for tracking, eight bits and its buffering cannot be accommodated. An eight-bit chip can be implemented in the given design area by splitting this logic and placing the flip flops in one row with its buffering in another row. However, this increases the number of rows of logic from seven to nine which cannot comfortably be accommodated in the vertical direction. Thus it is not until this rather advanced stage in the design that the decision can be made to implement four rather than eight bits on the chip.

The cells in the seven rows occupy a vertical height of approximately 1.22 mm (= 2 x 273 μm + 2 x 150 μm + 3 x 123 μm). This leaves 1.78 mm for inter-row gaps and space at the top and bottom of the design area for tracking to the input and output pads. Thus the average inter-row gap is 0.22 mm (= 1.78 mm/8).

A single layer of metal makes it sensible to adopt the conventions used within the cell layouts for routing. Thus metal interconnections run horizontally, while polysilicon and diffusion interconnections run vertically. This allows horizontal tracks to run over vertical routing without electrical effect (see figure 6.8).

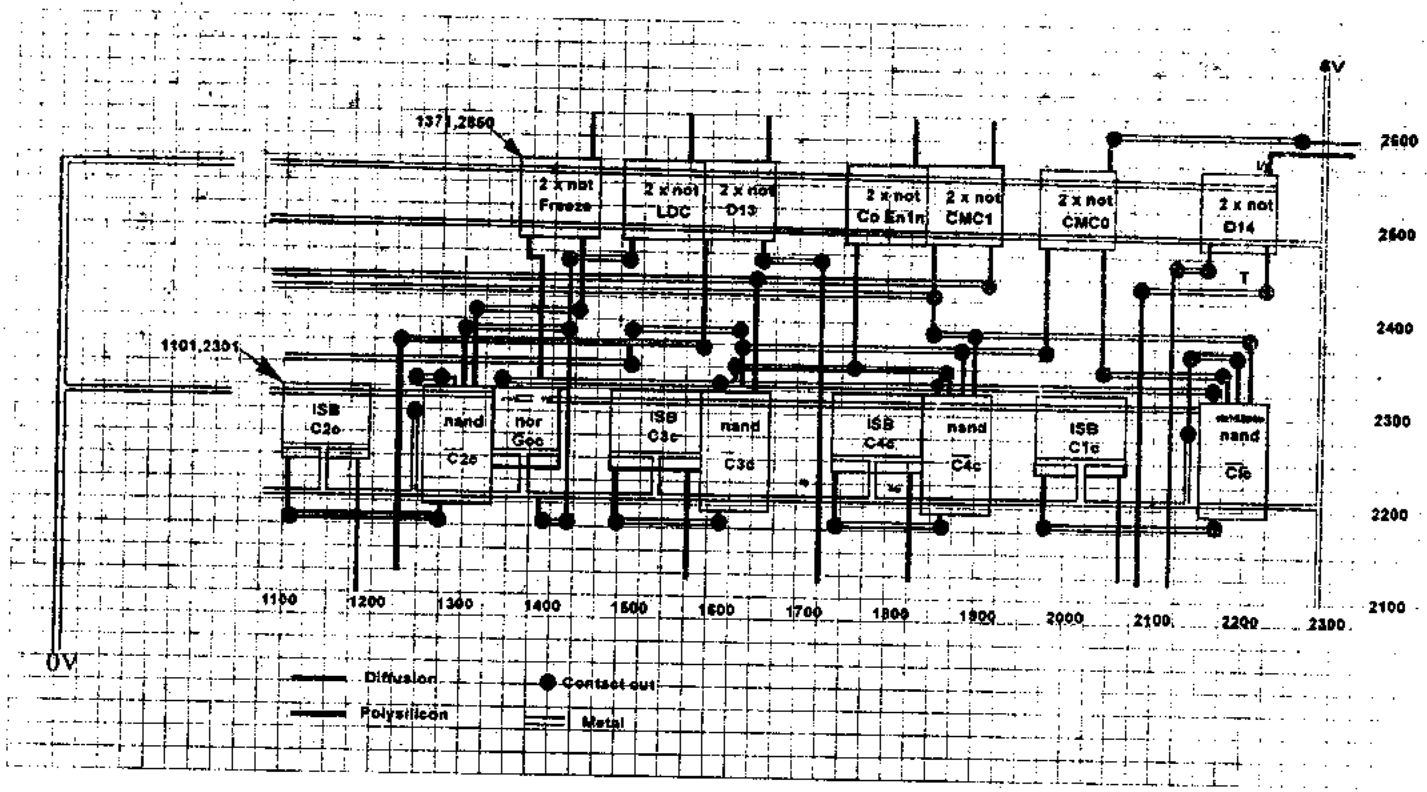


Figure 6.8 Floor plan and interconnections for counter control logic

Figure 6.9 shows that the area of metal surrounding a contact cut is 4λ x 4λ. A λ of 3 μm allows metal tracks of width 12 μm to be placed 6 μm apart. Thus an inter-row gap of 220 μm can accommodate twelve horizontal metal tracks.

Complex cells tend to generate a small but concentrated number of interconnections, while a large number of connections are associated with the primitive cells. This indicates that there will be a high concentration of tracks around the $\overline{J}$-$\overline{K}$ formation cell; this can be appreciated from figure 6.6 and proves to be the case in practice.

The layout of figure 6.7 shows that this cell has nine different inputs at the top and bottom of the cell which require connections. Allowance has also to be made for tracks to connection points in the logic directly above and below the formation cells. Since signals can approach a connection point from the left or the right, an inter-row gap of 220 μm should be adequate for these rows, and again this proves to be the case in practice. A smaller inter-row spacing can be used where the rows on each side of it contain primitive cells; there is a lower density of interconnections associated with these rows.

One consequence of routing horizontal tracks in metal and vertical tracks in polysilicon is that every time an interconnection changes direction, the conducting layer also changes. The other feature that arises is that the maximum polysilicon and diffusion length of any track is confined to the height of the design area. Furthermore, the system floor plan shows that vertically, the maximum distance data flows is across four rows. Thus the longest polysilicon and diffusion lines encountered should be 1.5 mm.

It had been hoped to approach the interconnection phase of the design in a hierarchical manner, so that, for example, having defined the interconnections for one bit of the data, this can be repeated for all bits. While this is true for small sections where the data flowed vertically from one row to the next row down, the interconnections for each $\overline{J}$-$\overline{K}$ formation cell of the counter are dissimilar. Also, control interconnections are irregular. This leads to individual routing for all sections of the chip, making the interconnecting phase the most time consuming task in the design process.

Figure 6.8 shows the routing for the counter control logic plus the external data bits DI3 and DI4. The cell outputs are indicated and an inspection reveals that it implements the relationships of table 6.2. Communication with other blocks of the system are also indicated. Thus CMC0, CMC1 and $\overline{\text{CMC1}}$ are sent to the terminal count logic, while the counter control signals, C1c to C4c, plus the true and inverse of DI3 and DI4 are sent to the $\overline{J}$-$\overline{K}$ formation cells on the next row down.

The cells in the rows are arranged so that the 0 V and 5 V lines align. However, the different height of cells in the second row requires a connection from the 5 V rail to the power rail of cells of smaller height. It can be seen that the effective height of a row is determined by the tallest cell, since it is difficult to utilise the spare row area created by cells of lesser height.

figure 6.9 High-density interconnection strategy

In figure 6.8, the 4-input nand gates have polysilicon inputs 6 μm wide spaced 6 μm apart. To maintain this density of vertical tracking when it connects to metal tracks requires some consideration. Figure 6.9 illustrates the configuration consistently used to achieve this. It can also be seen that the density of vertical tracks is optimised by alternating diffusion and polysilicon lines.

After defining the routing for the interconnections in the design area, the input, output, ground and power pads have to be placed around the design area and tracked to connection points within the design. In particular, the positioning of the ground and power pads have to be made bearing in mind any packaging conventions.

6.6 Fabrication and Testing

In order to submit a design for mask making, the content of each layer needs to be described in textual form. Here, the layout is usually described in terms of the polygons and rectangles comprising each layer. Thus for example, in the layout of figure 6.7, the 5 V line can be described as a metal rectangle of height 15 μm and length 216 μm, starting at an X, Y co-ordinate 0,123 μm with respect to the cell origin. This textual description can usually be automatically generated from a graphical description of the layout, or can be input directly.

The chip description is now checked for layout and electrical consistency. During layout checking, each design rule is taken in turn and its associated features located on the different layers; each occurrence of the features is examined to see that it conforms to the rule. For example, to check the overlap of metal around a contact cut requires that contact cuts and metal regions are located; their coincidence indicates positions where an overlap is required. Any errors detected during the layout checks cause an error message (or an error plot) giving the type of fault and its location.

The layout can also be used to extract electrical parameters and to generate a circuit description from which simple electrical checks can be performed. Device size extraction from the layout leads to the calculation of transistor aspect ratios, inverter ratios and the capacitance of points. Here, capacitances larger than a specified value or circuits with a low inverter ratio could be reported. Electrical checking generally consists of indicating faults such as circuit outputs connected together, or an output shorted to power or ground, or inputs that are unconnected (not driven).

After correcting any layout or electrical inconsistencies, the layout description normally leads to the production of a mask for each layer. This, in turn, is followed by chip fabrication as described in section 3.2. After production, the manufacturer performs parametric testing to ensure that parts of the wafer are within the processing tolerances. Provided the yield from this point of view is

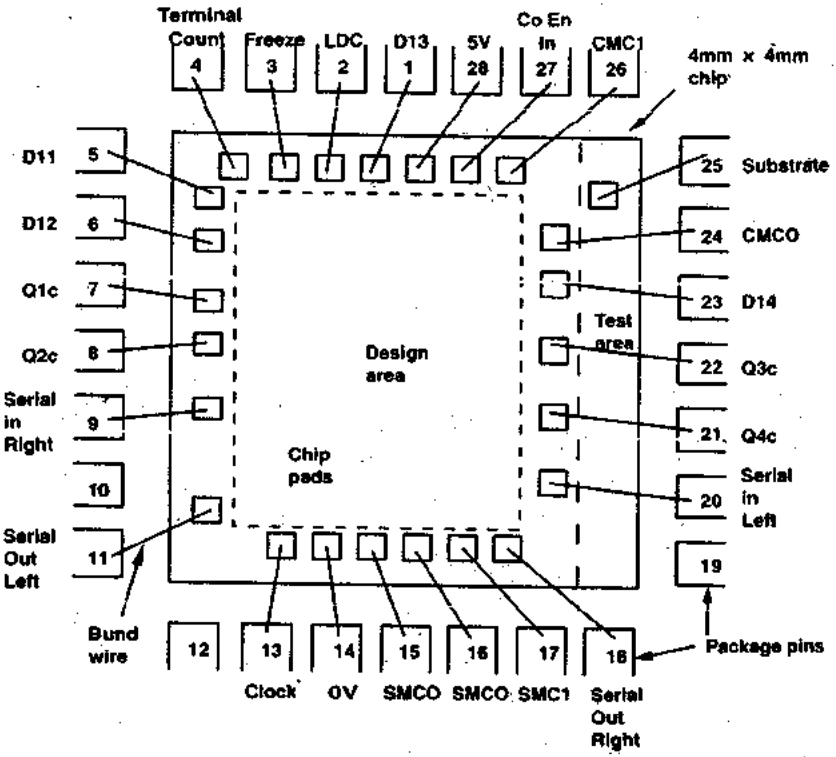


Figure 6.10 Pad bounding for a 28-pin dual-in-line package

reasonable, the wafer is broken. Chips are then selected from the good regions for packaging.

Chips are usually placed in a standard package which for small designs is likely to be a 24-pin, 28-pin or 40-pin dual-in-line pack. The wiring of the pack designer at the time the design is specified by a diagram supplied by the designer at the time the design is submitted for fabrication. this design example has 25 pins (including a pin for the substrate) and will therefore fit into a 28-pin package, as illustrated in figure 6.10.

The package has seven bounding pads on each of its four sides. Conventionally in such packages, ground is allocated to pin 14 and power to pin 28. This there only sensible to connect (that is, bond) from a pad in the design to a nearby pin on the corresponding side of the package (see figure 6.10); bounding wires should not cross, otherwise short circuits are likely to result. It will be noted that the substrate pad is in the parametric test area.

Table 6.4 Part of functional testing program

```
M:Q1C,Q2C,Q3C,Q4C$
L:CMC0,SMC0,SMC1,FREEZE,LDS,LDC,C0 EN IN,CLOCK$
H:CMC1$
T(1)$
L:DI1,DI2,DI3,DI4$
H:CLOCK$
L:CLOCK$
L:Q1C,Q2C,Q3C,Q4C,X$
T(2)$
M:Q4S$
H:SMC1,CMC0,C0 EN IN$
H:CLOCK$
L:CLOCK$
H:Q1C,Q2C,Q3C,Q4C$
L:Q4S,X$
T(3)$
L:SMC1$
H:SMC0$
H:CLOCK$
L:CLOCK$
L:Q4C,X$
T(4)$
H:CLOCK$
L:CLOCK$
H:Q4C$
L:Q3C,X$
T(5)$
H:CLOCK$
L:CLOCK$
L:Q4C,X$
T(6)$
.
.
.
```

Key

M	monitor output
L	low
H	high
T(N)	test number N
X	check outputs
$	end of line

Finally, the designer has to perform functional testing on the packaged chips to determine good and defective devices. These tests are normally the same as those performed at the logic simulation stage. Thus the packaged device is regarded and treated as a system of logic. While the tests indicate whether the chip is logically functional, it is normally performed at low speed and gives no information concerning the speed performance.

Table 6.4 gives a sample of part of the low level functional test for the chip which runs on an in-house tester controlled by a microprocessor. The program is interpreted line by line, so that operations occur in the order in which they are listed and the 'S' symbol marks the end of a line. In the initialisation sequence prior to test 1, the counter outputs Q1c to Q4c are monitored. The Clock and all input control signals except CMC1 are set low; CMC1 is set high. Thus the counter control indicates a load operation and $\phi 2$ is active.

Test 1 sets the four data inputs DI1 to DI4 low before applying the $\phi 1$ clock (Clock high) and the $\phi 2$ clock (Clock low). It should be noted that (in all tests) the control and data inputs are constant during $\phi 1$ when information is clocked into the masters; correct operation requires that any input changes occur when $\phi 1$ is inactive. When the execute symbol, X, is encountered in the listing, all monitored outputs are compared with those expected and, if they differ, an error is entered into a fault file. In test 1, the expected counter outputs are specified to be all low and this is compared with the outputs obtained for errors.

Test 2 transfers the all zeros counter content to the shift register and decrements the counter. It will be noted that it is only necessary to list the signals which change state. Thus the least significant shift register output, Q4s, is monitored in addition to the counter outputs. In test 2, after applying a master and slave clock, the counter output is checked for fifteen (all ones) and the least significant bit of the shift register for '0'.

Tests 3, 4 and 5 decrement the counter by one in each test and this is checked for. Thus in test 5, an error is flagged if the counter output is not twelve. Also in each test, the shift register contents are shifted one place to the right and Q4s checked each time against '0' to see that the counter to shift register transfer has been correctly performed.

In this way, the chip can be comprehensively tested and a chip with no errors in any test is logically correct. It has become almost obligatory at this point for the proud designer to display a picture of the design. However, the author has never found this exercise very illuminating and will therefore desist! It is more relevant to note that using the design techniques described in this chapter and by paying careful attention to the detail at all levels of the design, working designs can be implemented at the first attempt; furthermore, out of five packaged chips, four functioned correctly.