Accelerated Learning Series

(www.ALearnOnline.com – A site dedicated to education)

Modules in Computer Science & Engineering

Fundamentals in ASIC Design

A set of notes detailing fundamental concepts in Application Specific Integrated circuit design.

Author: SKG.
March 2006.

Revision History

Version 1.0 (March 2006)

- First version created

Table of Contents.

Prerequisites

- Fundamentals in Digital Design (ALS notes in Hardware Engineering)

Preface

In the notes on the “Fundamentals in Digital Design”, we discussed the scientific theories and techniques involved in the translation of logic equations into logic realizations using solid state digital gates. In this set of notes on the “Fundamentals in ASIC Design”, we will go a step further by investigating a “real world” problem and systematically walking through the various phases of an engineering project lifecycle. The problem we will address will involve the automation of a cashier stand in a grocery store. The solution for the problem will entail the design of an **Application Specific Integrated Circuit** (**ASIC**).

We will begin with a business case that justifies the need for an automated cashier stand and then proceed to gather the end user requirements. Subsequently we will come up with a high level technical design. We will then leverage the high level design to estimate the cost and duration of the project. Based on these preliminary efforts, we will put forth a realistic and competitive bid for the project.

Once our bid is successful, we will expand upon our high level technical design and start the unit design and implementation phase. This involves assigning engineers to the various modules identified by our high level design.

Finally we will integrate the modules and manufacture the ASIC before testing for compliance against the end user requirements used in the bidding phase.

At the conclusion of this module, the student will have a broad understanding of both the technical and non-technical areas of product development, while utilizing and building upon the technical knowledge gained previously.

Such an exercise, it is hoped, will allow the student to gain a finer appreciation for the skills that are required by the modern marketplace.

At a personal level, I must note that I am not an advocate of automation for the sake of increased profits. To deny the momentum of the modern trend however, would not allow me to equip the next generation with the skills required to survive in the world that my generation has left for them. To condone it on the other hand, would be to reveal my ignorance of the unsustainable effects of automation on human society. There is a delicate balance that I tread here and I hope my students will be forever conscious of that fact.

Acknowledgements

I am sincerely grateful to A. Walker for taking the time to review this document and for providing very valuable and constructive feedback.

1.0 – The Bidding Process

Assume you know of a grocer who employs five cashiers to accept payments from his customers at any given time. You find that he has difficulty hiring good cashiers and it appears even more difficult to keep the good ones – they seem interested to move on before too long. Every full time cashier costs the grocer an average of $30,000 a year for a total of $150,000 a year. If he can eliminate this cost, he could potentially relay the savings to his customers by lowering the prices of the merchandise that he sells and hence win more customers from his competition.

One way to eliminate the need for cashiers would be to automate the process using technology that will allow customers to pay for their purchases themselves. Since the store owner is not in the technology business, you encourage him to invite technology vendors like yourself, to present products and estimates on how much it would cost him to install such a self-service cashier machine. Based on the presentation from the different technology vendors, he gets a chance to decide on the product that best meets his needs and budget.

This process of getting input from multiple product vendors and then deciding on the product that best suits his requirements is referred to as the bidding process. It is often the starting point in the lifecycle of an engineering project. In this section we will briefly cover the main aspects of the bidding process.

1.1 – Business Justification

Although the grocer is fully aware of the difficulty he has in hiring and keeping his cashiers and the cost that they add to his business, he may not have thought through all the benefits of eliminating the cashier positions altogether. By quantifying the full scope of benefits, you help bring visibility to the advantages of your proposal and your product. It helps him justify buying your product for more reasons than he is already aware of. This form of advertising helps you win his business and hence it is a crucial component of your bid.

Let us first identify the different ways that automation of cashier stalls can impact your friend's grocery business.

1) Eliminate five people from his payroll for a total savings of $150,000 per year.
2) Eliminate the time and effort required to hire 1 new cashier every 3 months.
3) Eliminate the time and effort required to train 1 new cashier every 3 months.
4) Eliminate losses encountered due to human error during cash transactions.
5) Reduce extended line-ups when a trainee cashier is on duty.

Let us try and quantify these benefits into a dollar value in savings.

1) The savings in payroll is already quantified as $150,000.
2) Assuming on average your friend spends 8 hours advertising and conducting interviews to hire a cashier. Assume also that based on your friend's income, he earns about $100/hr. So the 8 hours he spends hiring a cashier is probably costing him $800 every 3 months for a total of $3,200 a year.
3) Training a new cashier takes 2 full days and 4 half days of your friend's time for a total of 4 full days. Assuming a full day involves 8 hours and again assuming your friend is worth $100/hr, cashier training is costing your friend $3,200 every 3 months for a total of $12,800 a year.
4) Assume on average each cashier losses $10 a day in human error. Assuming there are 5 cashiers working each day and that the grocery store is open for 260 days a year, this amounts to an annual loss of $13,000.
5) Customer frustration dealing with a trainee cashier is hard to quantify, but the grocery owner acknowledges that he has seen his customers leave the store, never to return under those circumstances. Let us be conservative and say that he losses 1 customer every time there is a new cashier. This in turn translates to 4 customers a year. On average a customer is worth $5000 a year in profits. The loss of 4 customers will then translate to $20,000 a year.

Let us now tabulate these figures to see the actual dollar benefit of automating the cashier stalls.

Benefit type	Savings
Payroll saving	$150,000
Hiring cost savings	$3,200
Training cost savings	$12,800
Human error savings	$13,000
Customer frustration savings	$20,000
Total Savings	**$199,000**

Not accounting for the capital investment required to buy the automated cashier machines, what the above table indicates is that even if the grocer needs to employ a technology specialist to maintain his automated cashier machines for $99,000 a year, he still shows a profit of $100,000 a year, if he bought your technology.

This business justification is a good starting point to get a feel for the feasibility of selling your product to a customer. If it takes less than five to ten years for the capital investment in a technology to pay for itself, in general, the business owner may be willing to make the investment. If it takes a lot longer, it will be harder to convince an entrepreneur of the merits of the investment, and you may be wise to save the cost of putting in a bid.

Professions in **Sales and Marketing** specialize in business justification analysis of this sort.

1.2 – Requirements Analysis

Once the preliminary business justification indicates that the prospect of using the technology is economically viable, the next step in the bidding process would involve getting a concrete understanding of the customer's requirements. This phase is particularly important because these are the requirements against which your final product will be tested for acceptability. If both you and the customer agree on these set of requirements and later you are unable to deliver a product that matches every one of the requirements, then the customer is at liberty to discard the contractual agreement and you could potentially lose all the effort and money you put into building the product.

In our example let us say that the following are the list of **End-User requirements** from the customer;

1) The Cashier machine must be able to scan a bar code.
2) The Cashier machine must be able to map a bar code to a price.
3) The mapping table used by the cashier machine to map a bar code to a price must be programmable so that the grocery store owner can alter prices as and when needed.
4) The mapping table shall not exceed 256 entries.
5) The Cashier machine must provide a sub-total after each item is scanned.
6) The Cashier machine will accept payments by credit cards only and will prompt the user to insert their credit card when they indicate that they have scanned all items.
7) Once a credit card is inserted, the Cashier machine will charge the current sub-total to the credit card and provide the customer with a receipt of the transaction.
8) The maximum price in any given session will not exceed $99.99. This limitation is added because the grocer is aware that he can save on the display unit costs by allowing this restriction.

We will now translate the end-user requirements into a list of **Technical Requirements** that will distinguish between the things that you will implement and things that you intend to get off-the-shelf (meaning purchased from other vendors), since they are either generic items that are cheaper to buy than make, or you don't have the expertise to make them.

Since you are in the business of designing ASICs, as part of the technical requirements you will split the requirements that will be satisfied by the ASIC and those that will be provided by other vendors.

Technology provided by 3rd party vendors.

1) Bar code scanners that interface with EPROMs (Erasable Programmable Read Only Memory).
2) EPROM programming interface.
3) Credit card processing machines that can interface with an ASIC.

Technology provided by your ASIC

1) Ability to interface with EPROMS to read cost of an item.
2) Ability to perform cumulative addition as each item is scanned.
3) Ability to display cumulative sub-totals on a 7-Segment LCD (Liquid Crystal Display) unit with 4 digits.
4) Ability to interface with a Credit card machine to make charges.

Professions in **Program and Product Management** specialize in this form of requirement gathering and analysis.

1.3 – High Level Design

Once the requirements are finalized, it is time to get a better feel for the extent of the engineering effort involved in building the product. A high-level design attempts to do this by identifying the various modules that need to be implemented and/or integrated into the final product. **Architects and other senior engineers** work with **Program Managers** to help identify all the building blocks and how they piece together. This will allow program managers to come with reasonably accurate time and cost estimates for the project.

Figure 1.3.1 shows a high-level block diagram for the Automated Cashier stand.

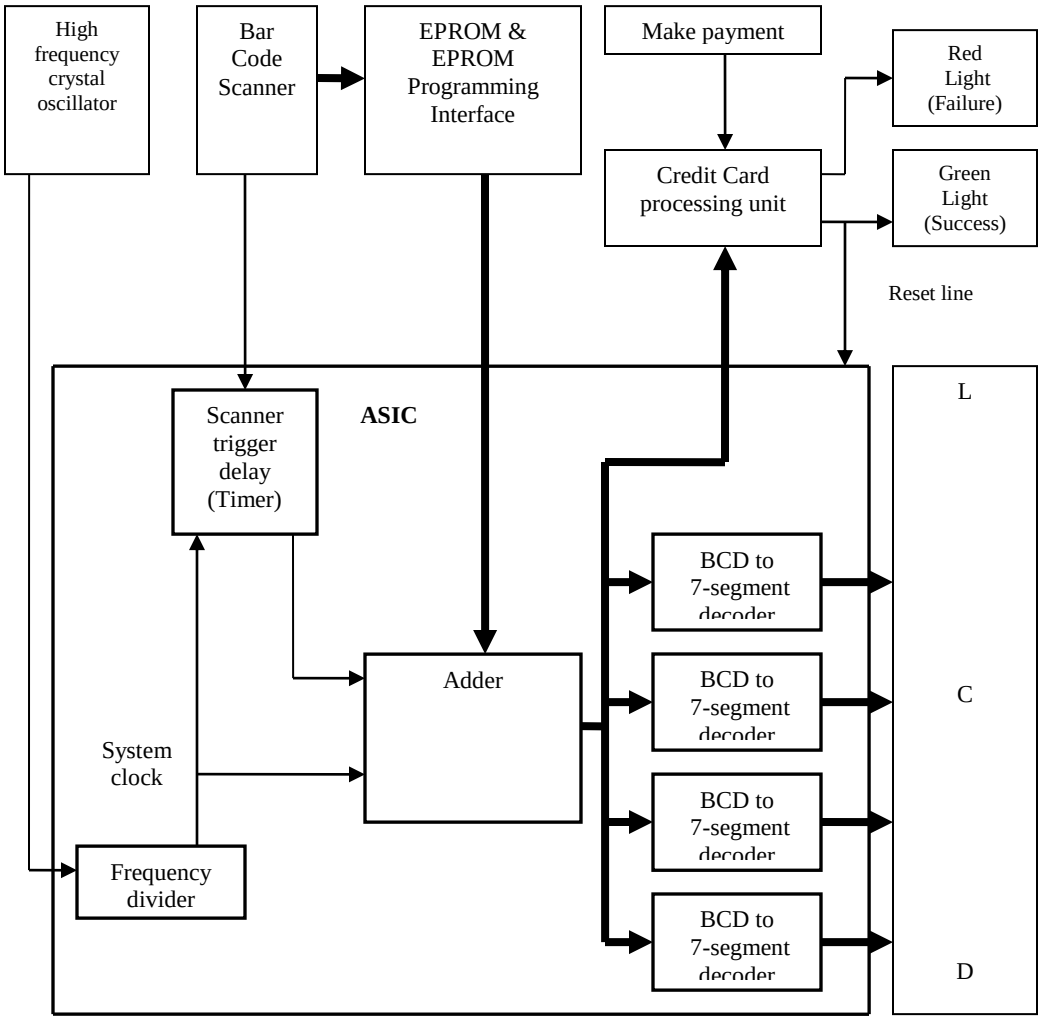


Fig 1.3.1: High Level Block Diagram

While the top level design does not necessarily go into the details of how a particular module is going to be designed or implemented, it does define the expectations of inputs, outputs and interfaces. This level of detail is sometimes referred to as a "**black box**" definition.

Peripherals:

We will start the analysis of this high level design by first studying the purpose of the **peripherals** (a part of the system that is external to the central core). Then we will study in the internal modules in the ASIC.

The first peripheral is a bar code scanner that supplies a code (over multiple output lines also referred to as a **bus** – shown as thick lines in Figure 1.3.1) to an EPROM. Simultaneously, the bar code scanner provides a trigger line to the ASIC to indicate that a customer has scanned an item. For simplicity, in our contrived illustrative example we will limit the number of items in the store to four bits (or 256 items).

The next peripheral is an EPROM that provides a programmable way for the grocer to map bar codes to prices. Think of the bar code as an address into a table in memory in the EPROM. Every time the address changes, the EPROM outputs the cost programmed into that row in memory. This output from the EPROM becomes input to the ASIC. Again for simplicity we will limit the cost of items to eight bits or 4 BCD (Binary Coded Decimal) values for a maximum price of 9999 cents per item.

The third peripheral is a credit card processor. When the user indicates that they have finished scanning, by pressing the “Make Payment” button, the ASIC will present the dollar value to the credit card processor over 16 lines (4 BCD values) for a maximum of $99.99. Once the transaction is completed by the credit card processor, it will indicate the status of completion as either a success or failure using a green light for success or a red light for failure. If the transaction is successful it will reset the ASIC, else a manual reset will be required, for which the store manager will be alerted.

The fourth peripheral is a 7-segment Liquid Crystal Display (LCD). This will be used to display the new cost after each scan.

As we have learnt previously, in any synchronous design, a system clock is required. We will use a 32MHz peripheral **crystal oscillator** (a device that provides a high frequency rectangular pulse) to get a clock input into the ASIC. We will then slow this clock for internal use, with the help of a frequency divider.

ASIC:

Now let us examine the modules that need to be implemented as part of the ASIC.

We have already indicated that the 32Mhz crystal oscillator is too high a frequency for our needs inside the ASIC. Hence we will need to implement a **Frequency Divider** that will output a clock that can be used system wide.

Next we need a way to delay the trigger coming from the scanner to allow time for the EPROM to process the input address change that occurred as part of the scan. If we read the output of the EPROM as soon as the bar code scanner indicates that an item has been scanned, there is a chance that we will catch the EPROM output in transition or in its previous state. This requires a **Timer** within the ASIC.

Then we need an **Adder** to do the cumulative addition required after each scan. The Timer module will provide the trigger for the addition process. When the adder is triggered by the Timer, it will read the output of the EPROM and add that to its current output (sub-total).

Finally we need four **BCD to 7-Segment** converters to take the output of the Adder and display it on the four LCD digits.

Based on these modules discussed, we can determine the ASIC parameters that will help us decide on what kind of ASIC we need. Key parameters of interest are things like the number of Input and Output (I/O) pins required and the number of gates required.

The number of I/O pins are easy to count. From fig 1.3.1, we need 18 input pins and 28 output pins for the LCD and 16 output pins for the credit card processor for a total of 44 output pins.

The gate count is harder to compute at a high level. But from previous experience, we can make a guess at the number of gates that will be used by each of the modules and make guesstimates.

1.4 – Scheduling and Cost Estimation

Project scheduling and cost estimation are perhaps the most critical components of the bidding process. Assuming all bids are on par in terms of functionality and design quality, the cost and the time to build the product is generally the key distinguishing factor in selecting the winning bid. On designs that you have previously implemented, it is often very easy to come up with accurate numbers for scheduling and cost. However, if this is the first time you are involved in a design of this nature, you will be wise to come up with your best guess and then pad it for unanticipated difficulties. The padding factor is something you choose based on previous experiences with guesstimates.

The key aspects of scheduling and cost estimation involve the following;

- Identify all the components that need to be built, bought, integrated and tested. We will leverage the effort we put into the high level design in the previous section for this purpose.
- Estimate the number of hours required to complete each of the tasks identified.
- Different tasks will require different skills and each skill type commands a certain hourly rate that is dictated by market forces. So we will make a note of the hourly rate for skill required for each task. In our example we will use 1 ASIC engineer at $70/hr, 7 purchasers at $30/hr, 1 PCB layout engineer at $40/hr. Look at the table below to identify the tasks for each of these people – you can identify them based on their hourly rate.
- We also need to identify any dependencies that a task may have on another task. This is essential because it signifies that the dependent task can only begin after its dependency is completed. Dependencies can be resource dependencies (same person has to work on both tasks) or task dependencies (one task requires another to be completed).
- Based on the above, we can make notes on the start and completion dates for each task.

The table below summarizes the above points for our automated cahier machine design.

Task #	Task Name	Hours	Hourly Rate	Cost	Dependency	Start Date	Completion Date
1	Purchase a Crystal Oscillator to be used as the high frequency clock	8	$30.00	$240.00	None	2nd Jan 2006	2nd Jan 2006
2	Purchase a Bar code scanner	24	$30.00	$720.00	None	2nd Jan 2006	4th Jan 2006
3	Purchase EPROM	24	$30.00	$720.00	None	2nd Jan 2006	4th Jan 2006
4	Purchase Credit Card Processing unit	40	$30.00	$1200.00	None	2nd Jan 2006	6th Jan 2006
5	Purchase Board to be used as the Printed Circuit Board (PCB) along with “Make Payment” switch and the “Red” and “Green” Light emitting Diodes (LEDs).	8	$30.00	$240.00	None	2nd Jan 2006	2nd Jan 2006
6	Purchase LCD	24	$30.00	$240.00	None	2nd Jan 2006	4th Jan 2006
7	Place order for the most appropriate ASIC chip to be used based on gate count and I/O pin count.	40	$30.00	$1200.00	None	2nd Jan 2006	6th Jan 2006
8	Design, simulate and test Frequency Divider module.	8	$70.00	$560.00	None	2nd Jan 2006	2nd Jan 2006
9	Design, simulate and test Timer module.	8	$70.00	$560.00	8	3rd Jan 2006	3rd Jan 2006
10	Design simulate and test BCD_to_7_Segment module.	8	$70.00	$560.00	9	4th Jan 2006	4th Jan 2006
11	Design simulate and test ADDER module.	40	$70.00	$2800.00	10	5th Jan 2006	12th Jan 2006
12	Integrate ASIC module and test with simulator.	16	$70.00	$1120.00	11	13th Jan 2006	16th Jan 2006
13	Design and etch PCB with layout information	16	$40.00	$640.00	None	2nd Jan 2006	3rd Jan 2006
14	Send ASIC net list for manufacturing and receive ASIC.	80	$100.00	$8000.00	12	17th Jan 2006	31st Jan 2006
15	Integrate peripherals with ASIC and test.	80	$70.00	$5600.00	14	1st Feb 2006	15thFeb 2006

Based on the table above, we know that as a rough estimate, the cost price for designing and implementing the automated cashier machine will cost us $24,400 and will require approximately 6 weeks to complete. To this cost we need to add our profit margins. Assuming a profit margin of 30%, the product will cost our client $31,720. Depending on the bids from our competition, we can tweak the profit margins to the extent necessary. Notice how competition forces producers to keep costs under check.

We may also want to allow for unanticipated delays in manufacturing of the ASIC or delays caused by malfunctioning ASICs that need to be resent to the manufacturing lab. If we add 2 weeks for such eventualities, we will be able to promise the product in 8 weeks to our client.

Scheduling and cost estimation is usually conducted by **product or program managers**. There are many software tools to aid in building and visualizing the table we constructed above. These tools have now become common place in project management. While they aid the process of constructing and policing a schedule, the essence of the planning exercise we engaged in above, remains unchanged irrespective of the tools that are used.

2.0 – Design and Implementation

In the previous section we looked at an end-user problem and broke it down into many separate tasks which when put together achieve the end-user goal. The techniques we employed in this breakdown were mostly a **top-down** approach. By that I mean that we almost always looked at things at a high level and then tried to see how we could break it down to a lower level. This way of viewing and analyzing issues is a trait that is common to managers. It lends itself well to delegation of responsibilities.

Scientists usually look at issues from a **bottom-up** approach. They tend to study low level fundamentals like the structure of atoms and then based on their understanding at that layer they try to apply that knowledge to predict interactions at a higher layer such as semiconductor behavior under the effect of doping agents.

Engineers often need to use both these techniques depending on the phase of the product cycle. In the bidding phase the top-down approach is very efficient. However in the design and implementation phase the bottom-up approach helps build early confidence.

In this section we will use the bottom-up approach to design each of the modules identified by the high level design in the previous section.

Note that I will not cover the design of the BCD_TO_7_Segment design, since it was already discussed in the prerequisite notes.

2.1 – ASIC Design Considerations

Use only high level logic devices

When we studied the design of logic gates using CMOS technologies in the course on Digital Design Fundamentals, we discussed NOT, NAND, NOR, Transmission Gates and D-Flip-Flops. However, in that section we did not discuss the delays (also known as **gate latencies**) involved in the outputs reflecting the changes to the inputs.

Yet another issue that we omitted in that discussion was the threshold input voltages required for recognizing a change in input state. Most of our discussion assumed that when we change an input from a "1" to a "0" or vice-versa the voltage associated with these logic states would change instantly (or in zero time). In reality however a voltage transition is never a "step" function but rather a "curve". At some point during this curve the transistor registers the change in state. This point is referred to as the **threshold voltage**.

When using gates in an ASIC design, parameters such as gate latencies and threshold voltages have to be well defined with only the most minimal variation allowances. If this were not the case, it would be impossible to predict the delays in rippling change through different parts of an ASIC design.

Most high level gates like NOT, NAND, NOR and D-Flip-Flops can be accurately parameterized using the worst case performance characteristics of the underlying transistors. However low level devices, such as Transmission Gates are harder to parameterize, since their performance characteristics are often dependent on the context in which they are used. In other words, it may be easy to parameterize a D-Flip-Flop that uses a Transmission Gate but not easy to parameterize a Transmission Gate on its own.

For reasons such as these, the use of low level logic gates are either forbidden or at least avoided in ASIC designs.

Avoid autonomously timed outputs driving asynchronous lines

Asynchronous lines refer to inputs that will cause a change in the output as quickly as the logic gates will allow. In other words, there is no dependency on any trigger (often a periodic clock) for the output to reflect the changes in the input values.

An Autonomously timed output generally refers to output lines that are not necessarily synchronized or free from glitches.

When an Autonomously timed output drives an Asynchronous input, there is a high likelihood of glitches on the output line being registered as a valid change in input state causing instability and unanticipated behaviors.

As an example let us study the circuit in Fig 2.2.1. Depending on the design of the counter, the transition from state "3" (011b) to state "4" (100b) may involve a glitch where momentarily the state of the output pins represent state "7" (111b). If this transition time exceeds the delay through the AND gate, the design that was intended to be a 7-state machine (states "0" to "6") would in fact end up being a 4-state machine (states "0" to "3") because the RESET line will force to output to "0" (000b) every time we go from state "3" to "4".

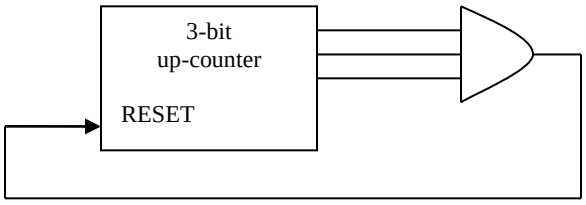


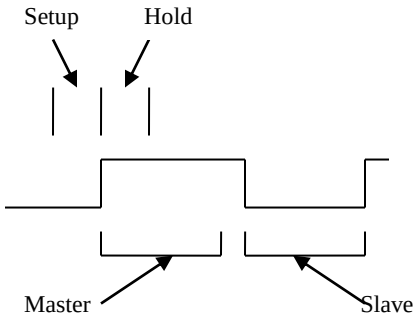
Fig 2.2.1: Autonomously Timed Outputs

Do not degrade clock edges

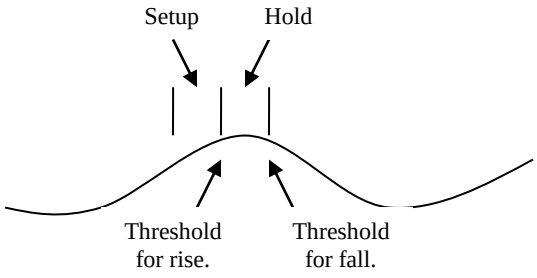
One of the advantages of the CMOS technology is the decoupling of the input and output wave forms. In other words, irrespective of the rise or fall times (time taken to reach the threshold voltage during state change) at the input of a CMOS transistor the output will have a constant rise or fall time. Thus a signal may pass through several gates without being degraded in CMOS. The reason for this should be clear once we understand that the output signal in CMOS is shielded from the input by the oxide layer. Note however, that as a signal passes through several gates, it will accumulate the gate latencies of each gate.

A node with a large **fan-out** (number of taps on a line) is however susceptible to degradation of the edges due to the variation of resistances on the different lines. A clock line is a good example of one with a large fanout. In synchronous designs, this line traces to almost all corners of the die leading to substantial degradation of edges (consequently poor rise and fall times). A poor clock edge can contribute to the malfunctioning of many synchronous devices.

As an example let as look at the impact of a poor clock edge on the functioning of a D-Flip-Flop. Fig 2.3.1 shows a comparison between a perfect clock edge and a severely degraded one. For a transistor to recognize a change in input state, it requires that **pre-threshold** voltage and **post-threshold** voltages be maintained for a period known as the "**setup**" and "**hold**" times respectively. If the time between the threshold for the rising edge and the threshold for the falling edge is less than or equal to the hold time for the D-Flip-Flop, it will behave as a transparent latch, where the input value is reflected at the output as soon as the rising clock edge is received.



D-Flip-Flop operating with an ideal clock



The effect of a degraded clock edge.

Makes a D-Flip-Flop transparent

Fig 2.3.1: Degraded clock edges

One way to remedy this situation is to exploit the CMOS decoupling effect by passing each fan-out of the clock through two CMOS NOT gates in series. This technique is referred to as "**clock buffering**". However when doing this you introduce a delay in the clock. This can lead to a "**clock skew**" where the time when the clock's rising edge arrives, varies in different parts of the die. Fortunately the software performing the layout of a design on a die, attempts to place all fan-outs at equidistance so that the delays on each fan-out is about the same.

Avoid race conditions

If combinational blocks link sequential blocks (refer to notes on Fundamentals of Digital Design for definitions), it is essential to ensure that the delays through the combinational blocks are less than the clock period. Although this flaw in design should be picked up by simulators, they can lead to errors in the actual chip if either the delays are fairly close to the clock period or if the simulator assumes delays for a different technology.

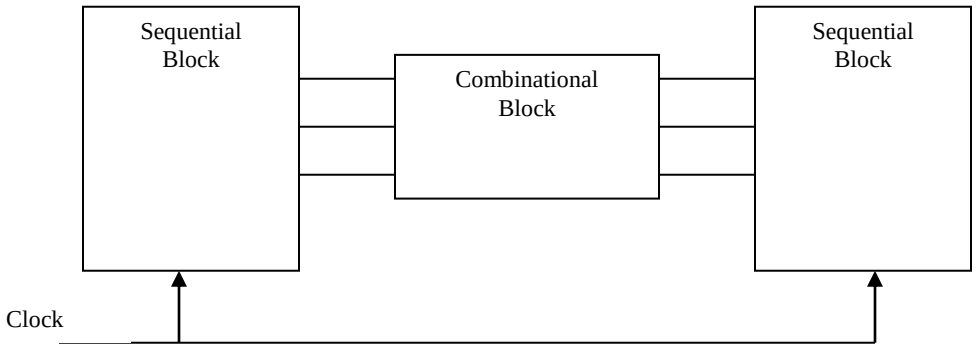


Fig 2.4.1: Potential race condition

Design for testability

In integrated circuits, it is essential to confirm the complete functionality of a chip in the shortest possible time. In the implementation of a prototype it might also be essential to isolate malfunctioning modules to identify the problem. Testing a chip by providing every combination of input can be very laborious and in most cases impractical. Hence the design should provide for a simple set of test vectors (set of input values) that would test for any anticipated failures. These anticipated failures dictate the fault model.

One of the more common fabrication errors is the "**stuck at low-high**". This refers to gates that are always stuck in one state and refuse to change state. Hence a common fault model allows for the toggling of all nodes in a design either directly or indirectly. This requires the design engineer to accommodate test circuitry in each module that will confirm, with a minimum set of input vectors, that all nodes in that module can toggle between the on and off values.

2.2 – Frequency Divider

The high level design in Fig 1.3.1 identified a high frequency crystal oscillator as the input clock. For our ASIC design a low frequency clock will be adequate. Let us assume that a 1KHz clock will meet our requirements. Further assume that our high frequency crystal oscillator generates a 32KHz clock. This will mean that we need to step this down to 1KHz to generate the internal clock signal that can be used by all the other modules within the ASIC.

It turns out that a frequency divider is essentially a counter design much like what was discussed in section 4.4 of the notes on Digital design. If you did the timing diagram exercise in that section, you will notice that each D-Flip-Flop divides the input frequency by 2. Hence to go from 32KHz to 1KHz, you need five D-Flip-Flops as shown in Fig 2.2.1.

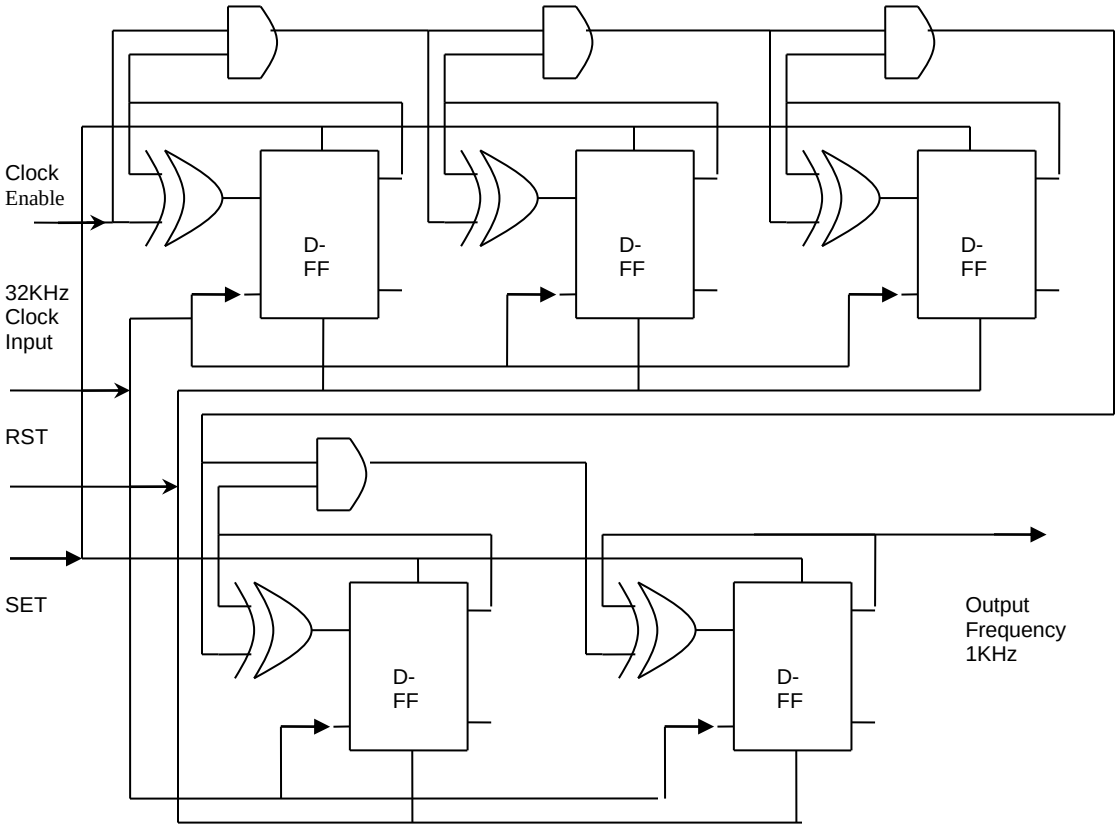


Fig 2.2.1: 5-Step Frequency Divider

Notice how we have added a "SET" line to the D-Flip-Flops. The SET line sets the output of each D-Flip-Flop to a "1". Based on the timing diagram for this counter you will see that once all the D-Flip-Flops are set to "1", it will take a single clock cycle to flip the output of all the D-Flip-Flops to a "0". This will confirm if we have a "stuck-at" problem with any of the D-Flip-Flops.

Fig 2.2.2 shows the additional test circuit that will check for stuck at problems. A , B, C, D and E represent the states of each of the D-Flip-Flops. After the SET line is toggled, A through E should be set to "1" and F should be ON. The very next clock cycle should turn G ON. This will confirm that we don't have any "stuck-at" issues in the frequency divider module.

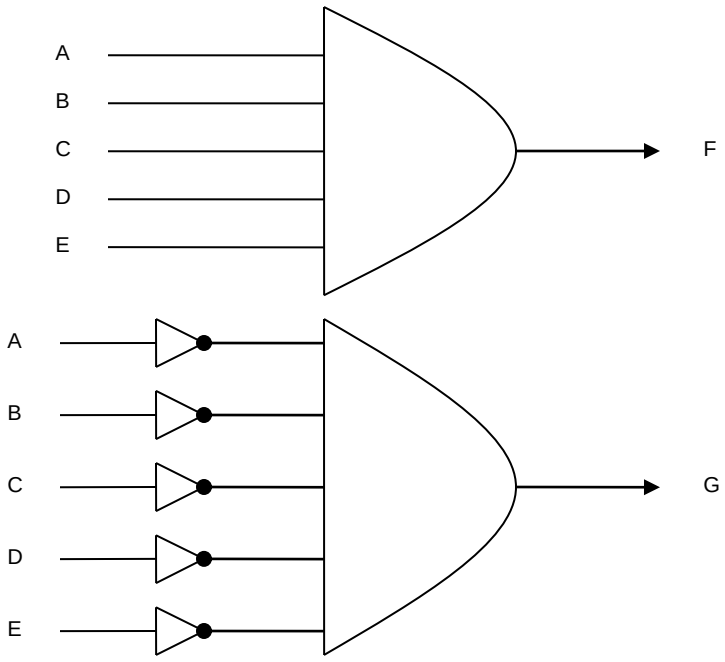


Fig 2.2.2: Frequency Divider Test Circuit

2.3 – Timer

The high level design in Fig 1.3.1, identified a need to delay the pulse, coming from the scanner, indicating that an item was scanned. This delay was to account for the time taken by the EPROM to output the cost associated with the item that was scanned. The “ADDER” module uses this delayed signal to perform a cumulative addition. The actual value of this delay will depend upon the specifications of the EPROM that we choose. Here we will assume that the delay required is a minimum of 2ms. This means that a 2-bit counter (4 states or 4ms maximum delay at 1KHz clock) will meet the requirements. Assume that the pulse coming from the scanner will remain high for 5ms.

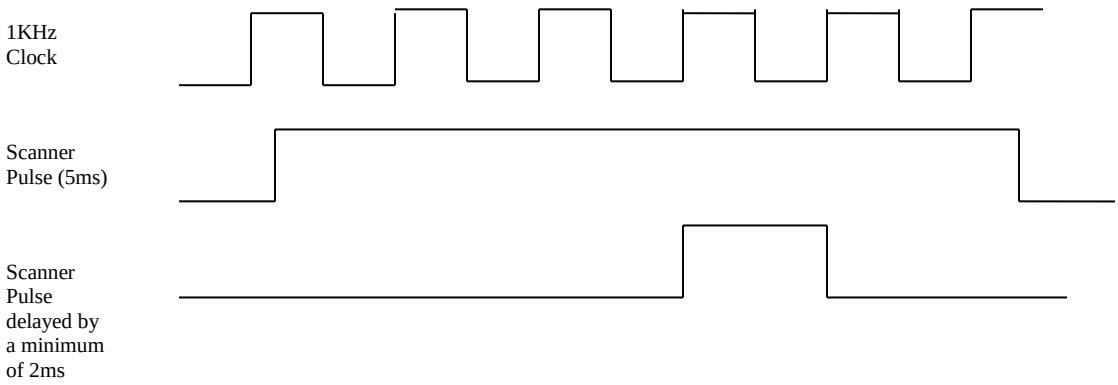


Fig 2.3.1: Timing diagram showing delayed pulse

Fig 2.3.1 illustrates the timing requirements. Our 2-bit counter will start counting at the first rising clock edge after the scanner pulse goes high. To do this we will use the scanner pulse as the clock enable line for our 2-bit counter. In 2ms, we can have 3 rising edges and so we can go from state 0 to 3 anywhere between 2 to 3 ms, depending on when the scanner pulse goes high. To ensure that the counter reverts to state 0 before the next scanner pulse, we will tie the inverse of the scanner pulse to the reset line on the 2-bit counter.

Fig 2.3.2 shows the design of this circuit. Notice how we pass the scanner pulse through a D-Flip-Flop that is clocked by an inverted clock line. This serves two purposes. First the flip-flop makes the scanner pulse synchronous and we are thus immune to glitches on this line. Second, by using an inverted clock line, we ensure that the clock enable line for our counter satisfies the signal hold time requirements very comfortably since it has an entire half clock cycle before the counter is going to change its state. Such an inverted clock line is also referred to as a 90 degree phase shifted clock.

Since we have an active low reset line, we can conveniently use this latched scanner pulse to clear our counter to ensure it is at state 0, at the time the counter clock enable line is set.

At the output, we again use the 90 degree clock phase shift technique to ensure that our delayed scanner pulse satisfies the setup and hold time requirements of the ADDER module. Notice also that we pass the output of the AND gate (that detects the state 3 on the counter) through a flip-flop to ensure that no glitches on this line ripple into the ADDER module.

As an exercise, you can now redraw the timing diagram of Fig 2.3.1 to take account of the clock phase shifts being employed at the input and output ends of this design. Also re-design the circuit to detect “stuck-at” issues in the minimum amount of clock cycles.

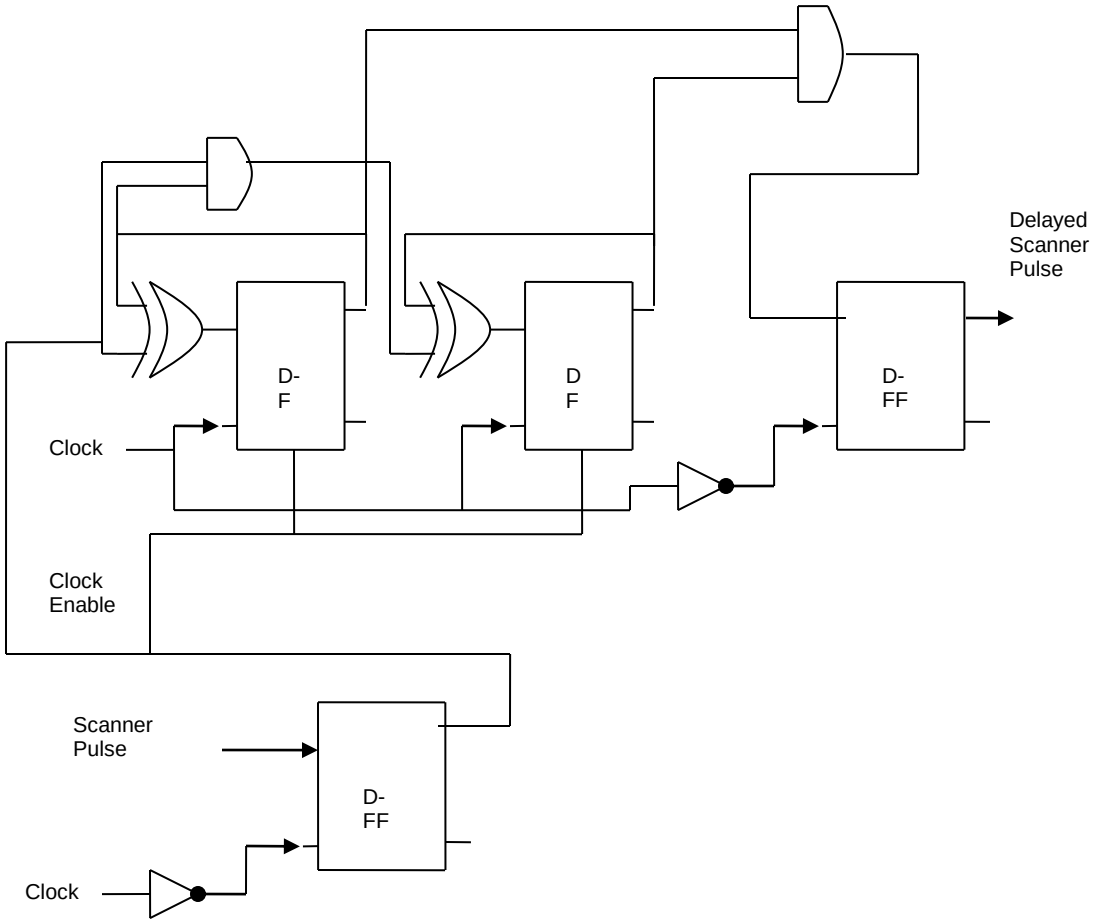


Fig 2.3.2: Timer Delay

2.4 – Adder

The “ADDER” module is going to be, by far, the most complicated module in our ASIC design. We will start the design by applying the top-down design technique that will help us breakdown all the tasks that need to be done to achieve the addition operation. Fig 2.4.1 is a high level design that identifies the building blocks that will be used in the design of the “ADDER”.

First we will design a 4-bit binary adder. It will take as input, 2 sets of 4-bits along with a carry-in, and offer 4-bits as output along with a carry-out. One set of 4-bit input will be tied to the EPROM output that gives us the cost of the grocery item that was just scanned. The other set of 4-bit input will be a feedback representing the current cumulative total.

Since we are adding the cost of groceries, we would want to perform this addition in Base 10 arithmetic. The B_To_D converter module will convert the sum of the 4-bit binary addition into a decimal addition. For example, if the sum of the addition is 0x0B (decimal 11), it will convert this to a decimal “1” with a carry of “1”.

The 4-bit Latch is what synchronizes the addition operation with the scanning operation. The “Transmit” line in the input of the Latch will be tied to the delayed scanner pulse and this line will be used to “enable” the latch. This means that when the “Transmit” line is set, at the next rising edge of the clock, what is at the input of the latch will be transmitted to the output and this output will get displayed on an LED as well as sent back to the adder in a feedback loop. Notice that the “Transmit” line has to coincide with a rising clock edge for the signal to be transmitted. This ensures that any glitches in the “Transmit” line don’t affect the “ADDER” module.

Finally we will place four sets of these “ADDER” building blocks in parallel to have the 4 decimal digits as stipulated in the requirements.

What we have done in Fig 2.4.1 is to design using a top-down approach. This has helped us isolate the tasks into individual modules, much like the technique we employed when we come up with the high-level design. Once the tasks are identified the bottom up approach is more efficient.

To design a 4-bit binary adder, we first need to design a 1-bit binary adder. A 4-bit binary adder is just a combination of four 1-bit adders. The 1-bit (and hence the 4-bit) adder is a simple combinational logic circuit.

Following this, we will design a B_To_D converter module, with a simple combinational logic circuit.

Before we design the 4-bit Latch, I will discuss and design another fundamental digital circuit building block known as a multiplexor and a de-multiplexor.

And finally we will design the latch that uses multiplexors along with all the other building blocks we have used previously. This will be a sequential circuit.

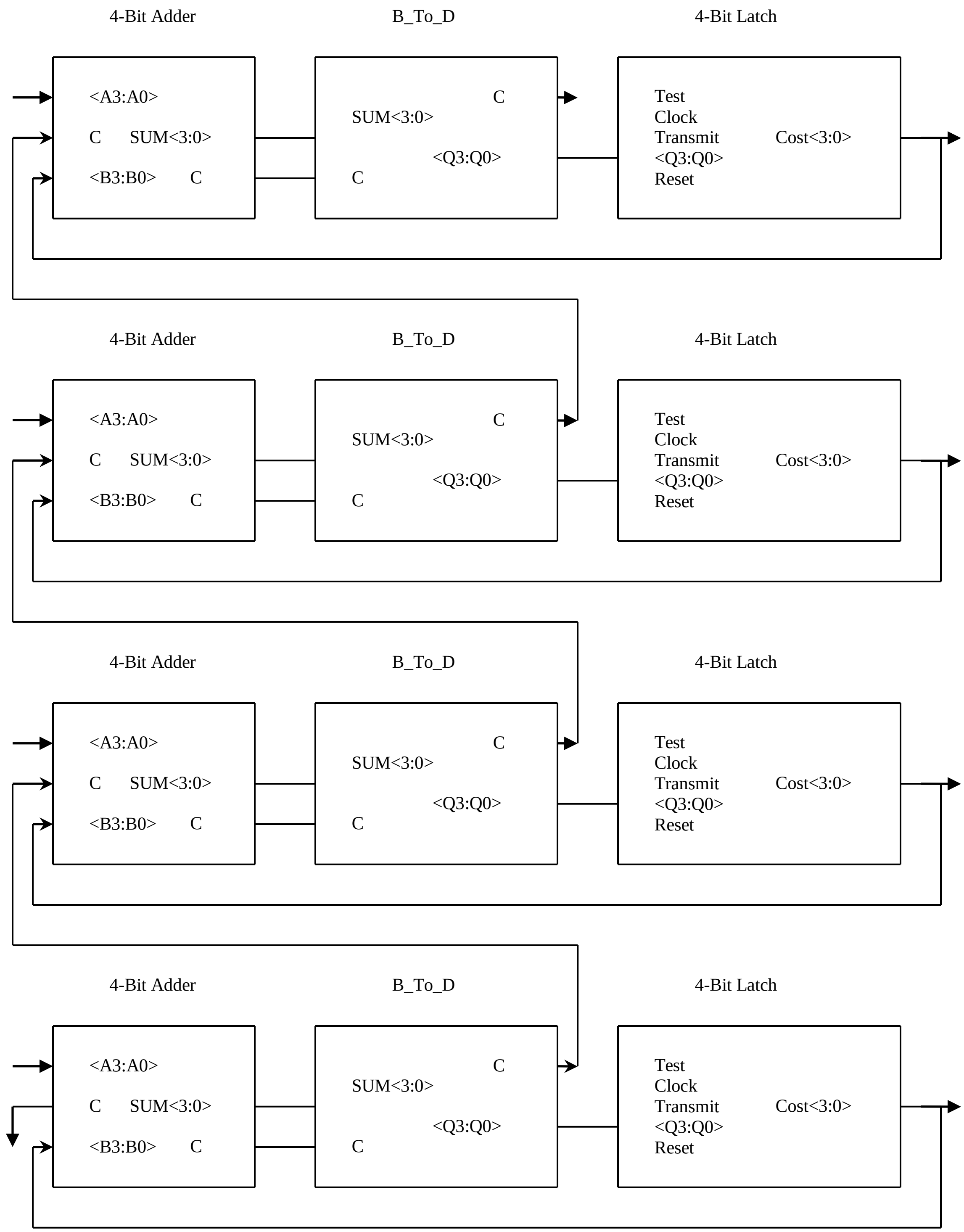


Fig 2.4.1: High level Adder Design

1-Bit Adder

A 1-bit adder accepts 2 1-bit inputs along with a Carry-In and outputs a 1-bit SUM along with a Carry-out.

The truth table for this circuit is given in Fig 2.4.2.

Input A	Input B	Carry-In	SUM	Carry-Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig 2.4.2: Truth Table for a 1-bit Adder

Using the Karnaugh map design technique we learnt previously, we will come up with the circuit in Fig 2.4.3 to implement the truth table in Fig 2.4.2.

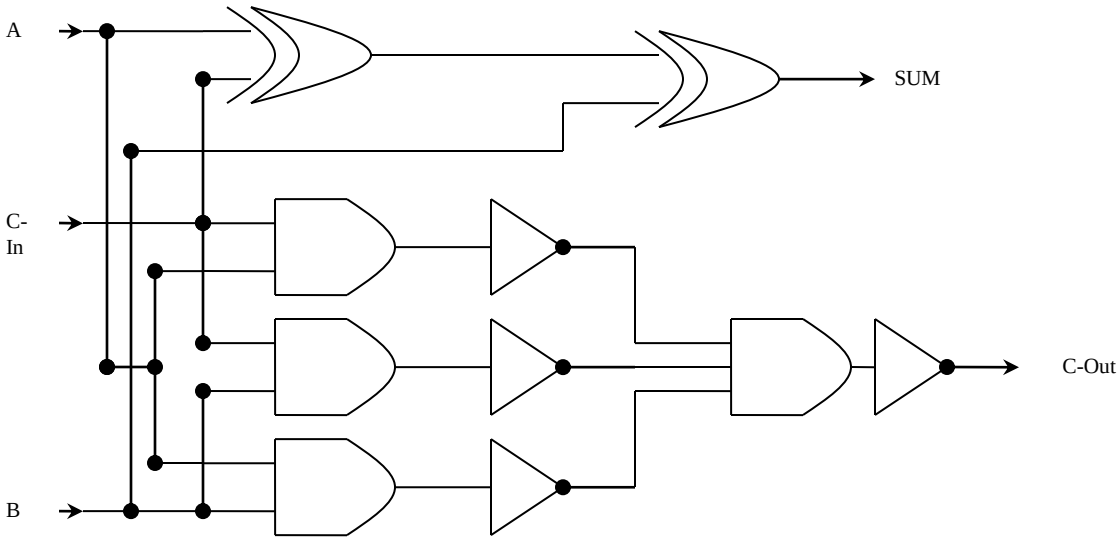


Fig 2.4.3: Design for a 1-bit Adder

4-Bit Adder

A 4-bit adder is essentially a combination of four 1-bit adders in parallel as shown in Fig 2.3.4. The inputs are A<0:3>, B<0:3> and a Carry-In. The outputs are SUM<0:3> and a Carry-Out. Note that we feed the Carry-Out of each bit into the Carry-In of the next more significant 1-bit adder. Also note that this is a purely combinational circuit and that we can expect glitches in the output. However we can accommodate these glitches because of the 4-bit Latch that we have accounted for in our overall ADDER design.

The Carry-Out line from the 4-bit adder will be fed into the Carry-In line of the next more significant 4-bit adder. The Carry-Out line from the most significant 4-bit adder will be used to signal an overflow in addition. Recall that the requirements stipulated a maximum price of $99.99.

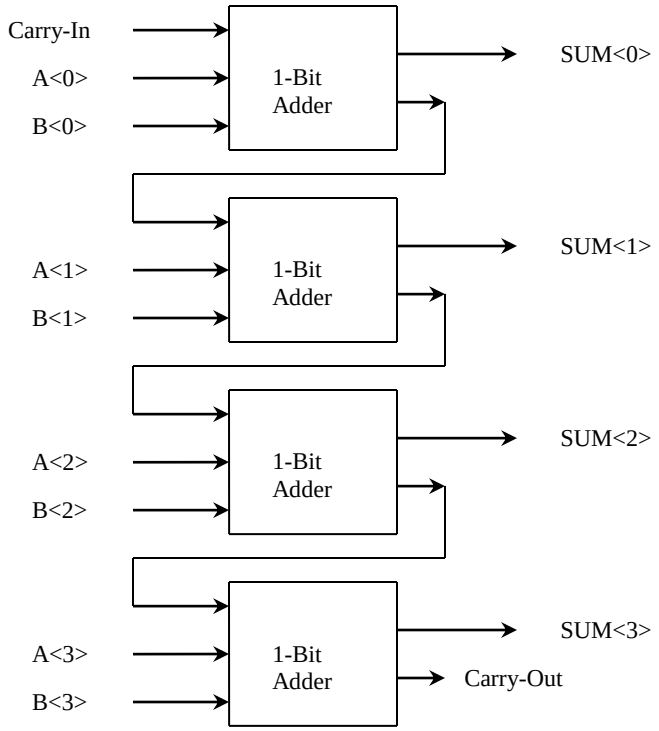


Fig 2.3.4: 4-bit Adder

B_To_D Converter

Since the end user of our design is going to expect the cost to be displayed in base 10 arithmetic, we will have to convert our 4-bit binary addition into a decimal number. This conversion will also impact the carry-out line that will be fed into the next more significant 4-bit adder. For example, if the 4-bit binary addition results in a 0xF, the carry-out line would have been a zero. However, once the sum is converted to decimal, the sum would change to “5” and the carry-out would be set to “1”.

The Truth Table for the B_To_D converter is shown in Fig 2.3.5 below. The input in this truth table are prefixed with “B_” (for binary) and the outputs are prefixed with “D_” for decimal.

B_S<3>	B_S<2>	B_S<1>	B_S<0>	B_C	D_S<3>	D_S<2>	D_S<1>	D_S<0>	D_C	#
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1	0	1
0	0	1	0	0	0	0	1	0	0	2
0	0	1	1	0	0	0	1	1	0	3
0	1	0	0	0	0	1	0	0	0	4
0	1	0	1	0	0	1	0	1	0	5
0	1	1	0	0	0	1	1	0	0	6
0	1	1	1	0	0	1	1	1	0	7
1	0	0	0	0	1	0	0	0	0	8
1	0	0	1	0	1	0	0	1	0	9
1	0	1	0	0	0	0	0	0	1	10
1	0	1	1	0	0	0	0	1	1	11
1	1	0	0	0	0	0	1	0	1	12
1	1	0	1	0	0	0	1	1	1	13
1	1	1	0	0	0	1	0	0	1	14
1	1	1	1	0	0	1	0	1	1	15
0	0	0	0	1	0	1	1	0	1	16
0	0	0	1	1	0	1	1	1	1	17
0	0	1	0	1	1	0	0	0	1	18
0	0	1	1	1	1	0	0	1	1	19

Fig 2.3.5: B_To_D Converter Truth Table

Note that I have only accounted for states 0 through 19. This is because the prices that are going to be input into the adder are going to be decimal numbers. The maximum decimal number we use in any 4-bit adder is “9”. So the highest output we expect will be when we add 2 nines with a carry-in. This will result in a sum of “19” and that is highest state in our Truth table. While it is useful to add all the possible states and display errors if these unanticipated states manifest, it would cost us a lot more gates to implement them.

I will leave the conversion of this Truth Table into a gate design as an exercise. Note that this easily done using freely available design software on the internet. But it can also be done manually using Karnaugh maps.

Multiplexors (MUX) and De-Multiplexors(DE-MUX)

Before designing the latch circuitry, I will introduce yet another very common building block used in digital design, much like gates and flip-flops. They are known as multiplexors and de-multiplexors (or MUX and DEMUX for short).

A MUX takes multiple inputs but has only one output. The output will be one of the inputs into the MUX. Which input is chosen to be the output will depend on a special input known as a "selector". For example, for a 2-input MUX, you need a 1-bit selector. When the selector is low, one of the 2 inputs will pass through to the output and when the selector is high, the other input will pass through to the output. Fig 2.3.6 shows the Truth Table for a 2-input MUX. Notice that when the select line is low, the output is always the same as the Input A. When the select line is high, the output is always the same as the Input B.

Input A	Input B	Select	Output
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Fig 2.3.6: Truth Table for a 2-input MUX

I will leave it as an exercise to translate the above truth table into a gate design. For our purposes here, we will from now on use the block diagram in Fig 2.3.7 to represent a MUX.

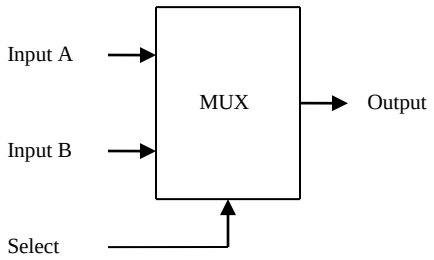


Fig 2.3.7: MUX Block Diagram

A DE-MUX is the inverse of a MUX. It takes one input and has multiple output lines. The input line is reflected in one of the output lines based on the value of the selector line. As an exercise you can create a Truth Table for a DE-MUX and subsequently convert your Truth Table into a gate design.

4-Bit Latch

Fig 2.3.8 shows the design for a synchronous 4-bit Latch.

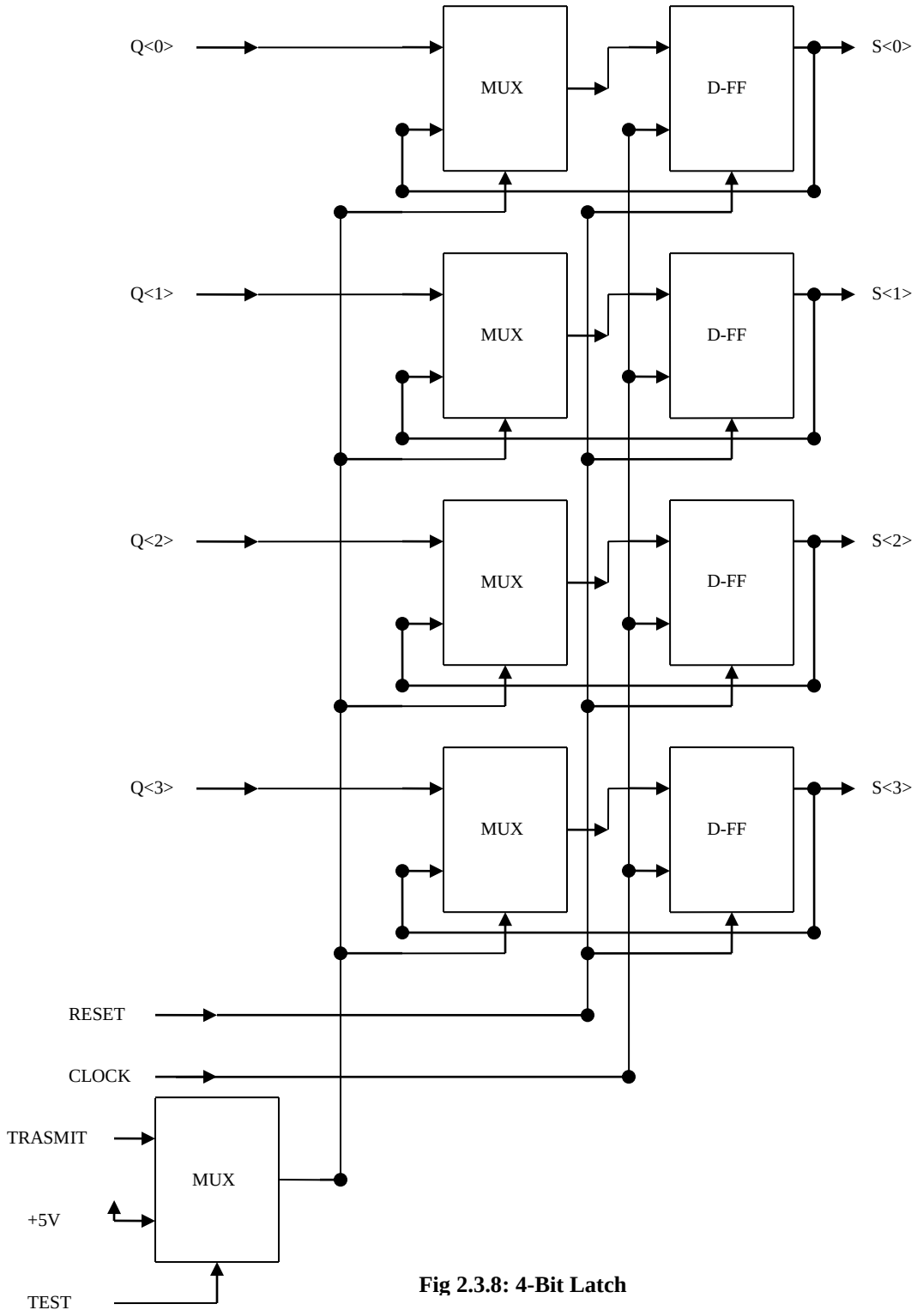


Fig 2.3.8: 4-Bit Latch

The 4-bit input into the latch is fed into 4 MUXs that is selected by the “TRANSMIT” line. The “TRANSMIT” line will be tied to the delayed scanner pulse that we designed earlier. When the “TRANSMIT” line is high the 4-bit input into the latch is the output of the MUXs. When the “TRANSMIT” line is low, the output of the MUXs are the same as the output of the D-FF, since the signal is fed back.

Note that when we are in “TEST” mode, the “TRANSMIT” line is always high because the “TEST” line is the selector used to choose between the incoming “TRANSMIT” and a constant high signal level (+5V). This will help test the circuitry for stuck-at problems without being dependent on an incoming “TRANSMIT” line.

Note that this is a synchronous circuit since the output of the D-FF is only set on the rising clock edge. Because of the 90 degree phase shift technique we employed in the design of the delayed scanner pulse, the TRANSMIT line is guaranteed to withstand the “set up” and “hold” time constraints at the time the rising clock edge occurs.

With the design of the 4-bit Latch we have completed the design of the ADDER by addressing all the modules identified in the high level ADDER design of Fig 2.4.1.

3.0 – Integration and Testing

Once the module design phase is completed, we move into the integration and testing phase. Here the input and output interfaces to each module is tied to other interdependent modules and then the integration engineer would simulate the functionality of the entire system (in software) to confirm that the overall design lives up to the original statement of requirements.

Once the simulation results prove satisfactory, the design in the form of a “net list” will be sent to an ASIC manufacturing plant. During the manufacturing process, the “net list” will be translated to metal contacts in a pre-build gate array of transistors on a silicon die.

The chip is then received from the manufacturing plant and then tested for stuck-at issues and functionality.

In this third and final section we will cover some of the details of these aspects of integration and testing. We will limit the discussion to modules within the ASIC. Integration issues with the 3rd party modules outside the ASIC will be considered beyond the scope of this set of notes.

3.1 – Putting the pieces together

Fig 3.1.1 shows all the ASIC modules integrated with all input pins to the left and output pins to the right.

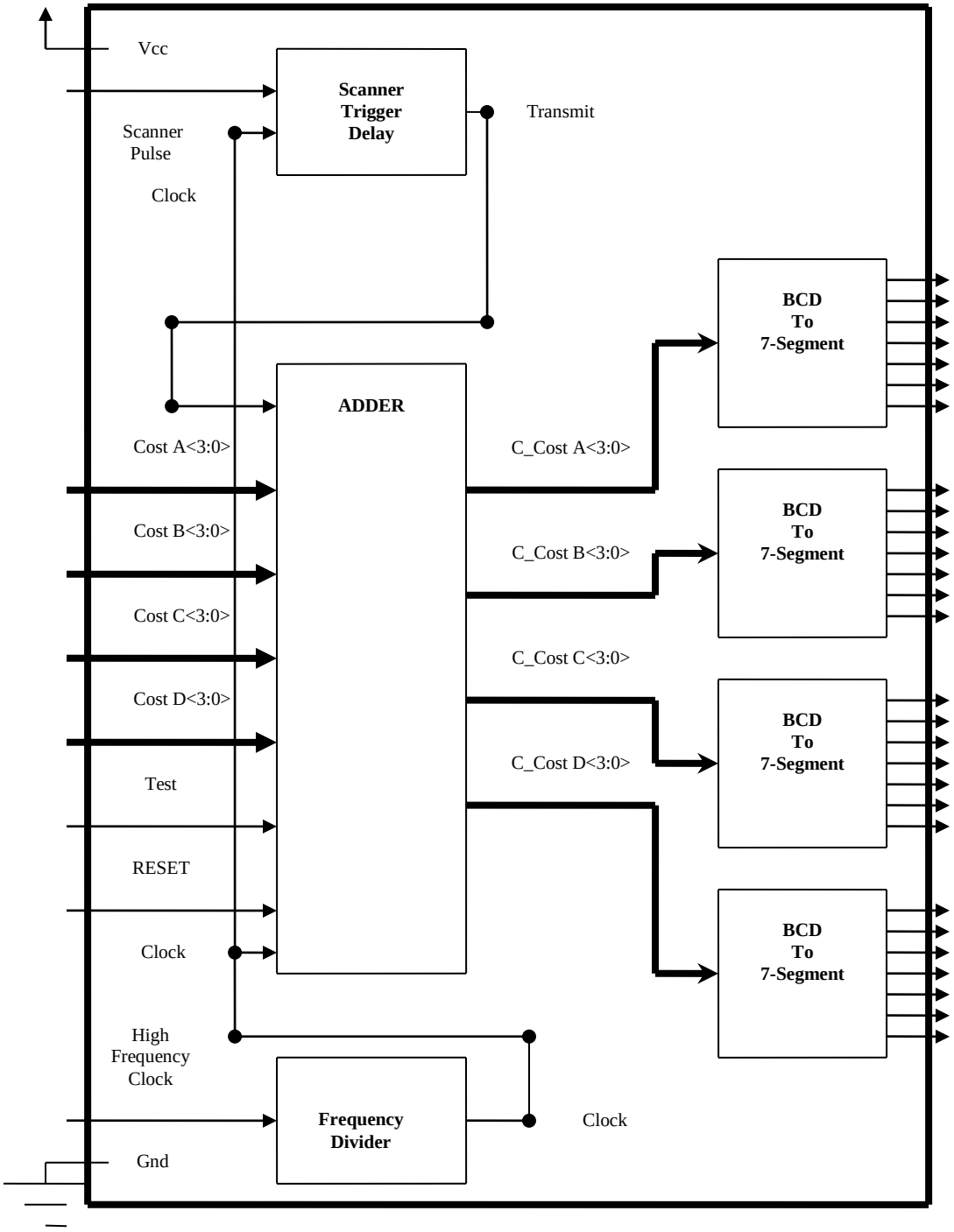


Fig 3.1.1: ASIC Modules integrated

Note that to add test circuitry to the Frequency Divider, Scanner Trigger Delay and the BCD To 7-Segment modules would consume additional I/O pins and gates. Hence I have avoided them here.

The Adder module, however, provides for a “Test” line. This line effectively simulates a “Transmit” line active behavior. This will enable us to test the Adder module, without depending on a pulse from the scanner. By manually pulsing the “High Frequency Clock” input, we can receive clock rising edges on demand. This capability combined with the “Test” line and input cost lines, allows us to test the Adder independent of the other modules.

Note that we have used 2 Input pins for power and ground. This brings the total number of input pin requirements to 21. And the total number of output pin requirements is 28.

3.2 – Chip manufacturing process

Way back in 1916, a Polish Chemist by the name of **Jan Czochralski**, accidentally dropped a pen in a container of molten tin. He quickly removed the pen and found a thin thread of solidified metal hanging from its tip. He then replaced his pen with a capillary and found that the thin thread was a single crystal.

The Czochralski method of creating a single crystal is the starting point in the manufacture of semi-conductors. The process begins by dipping a "seed" semi-conductor crystal, mounted on a rod, into a container filled with molten semi-conductor (usually silicon). When the rod is pulled upward and rotated simultaneously, a cylindrical piece of silicon crystal is formed. This is known as an **ingot**. Ingots vary in dimensions but usually range from one to two meters in length and about 300 millimeters in diameter.

The ingots are sliced into thin **wafers** that are typically in the range of 0.75 millimeters. These wafers are polished into a very flat surface and referred to as **"virgin wafers"**.

The virgin wafers are then **oxidized** by exposing the wafer to high heat and gases. This causes a growth on the virgin wafer. The process is similar to building rust on a metal. Then the oxidized wafer is coated with a **photo-resist** material (material that is sensitive to light). Then **masks** representing the digital design are used to carve out the design by shining light to the photo-resist through the mask. The exposed photo-resist and the underlying oxidized layer are then **etched** away to reveal the pattern of the mask on the oxidized wafer.

An insulation layer is then applied and another coat of oxidized layer is formed. The process of masking and etching is repeated with the next mask representing the design. The two layers are then bombed with P-type or N-type ions to achieve doping. Then a **metal** is dropped to fill holes that were left open between the 2 layers to make electrical connections between the layers.

The process is then repeated for as many times as there are masks. Note that each wafer can accommodate hundreds of chips. The chips are cut from the wafer once the above process is completed, along the crystal cleavage to form a **die**. The die's I/O pins are connected to chip package pins and then the chips are ready for shipping.

Note that chip manufacturing laboratories have to meet certain minimum "**Clean Room**" standards since even tiny particles of dust can negatively impact the manufacturing process. The **International Standards Organization** currently defines 6 classes of clean rooms – Class 1, Class 10, Class 100, Class 1000, Class 10,000 and Class 100,000. These classes refer to the maximum number of particles bigger than half a micron in 1 cubic feet of space.

In **ASIC** design there are generally three manufacturing possibilities – Standard cell, Gate Array or Full Custom.

In the **Standard cell** approach, the designer uses a high-level design language to describe the module level functionality. Software then converts this to high-level design such that it uses well known standard cells whose electrical characteristics are well established. These standard cells are then placed as they would appear on the chip and routed to connect all interdependent modules. This is then transformed into the masks required for fabrication.

In the **Gate array** approach, the designer uses a generic predefined wafer which has transistors already formed but not yet connected using the metal. The design defined the metallization required to achieve the final implementation. While the manufacturing costs may be lower using the Gate Array method, the transistor utilization is also lower because of placement and routing limitations. Gate arrays have mostly been replaced with Field Programmable Gate Arrays (FPGA). These are software programmable and hence reduce implementation costs considerably.

The **Full custom** design allows one all the flexibility on gates, placement and routing. This approach can be used by ASIC as well as generic products. However it can be expensive to manufacture and will require considerably larger amounts of testing resources.

3.3 – Testing the ASIC for stuck-at problems

As discussed previously, one of the most common errors in chip manufacturing is that a transistor can get “stuck” in a state – either low or a high. To catch this early in the test cycle, one of the first things that is done when a chip is received for the chip manufacturer is to quickly see if we can toggle as many transistors as possible.

In our design, we allowed for a “Test” line into the Adder module. By turning this “Test” line on, we can feed various input costs and confirm that all our latches are toggling at the output. We can manually clock 32 pulses at the “High frequency Clock” input to get a rising clock edge into the Adder module.

Feeding the Adder with an input of “7777” in the first clock cycle and then feeding it with a “2223”, should confirm if 3 of the 4 bits in the 4 bit adder are capable of toggling. This test case (also known as a **test vector**) forces the lines to go from a “1” to “0”.

You can identify similar test vectors that test as many nodes as possible within each module. Even though we don’t have a “Test” line into the Frequency divider or the Scanner pulse delay modules, you can indirectly confirm the toggling of nodes within these modules by confirming that the Adder is behaving the way it would be expected while acknowledging signals from the other modules.

Note that if there are any stuck-at issues, the design is sent back to the manufacturing lab and further test resources are not expended at this juncture.

3.4 – Functional Testing of the ASIC

In Section 1.2, we outlined the four requirements that will be provided by the ASIC design as follows;

- Ability to interface with EPROMS to read cost of an item.
- Ability to perform cumulative addition as each item is scanned.
- Ability to display cumulative sub-totals on a 7-Segment LCD (Liquid Crystal Display) unit with 4 digits.
- Ability to interface with a Credit card machine to make charges.

For each requirement, the test engineer will come up with a "**Test Case**" to validate that the requirement is satisfied.

For the first requirement, our design allows for 16 input pins to interface with an EPROM. The test engineer can tie these pins to de-bounced switches (switches that can generate a "clean" signal) and verify that a manually generated clock pulse, causes the input presented by the de-bounced switches are reflected at the output of the ASIC.

The second requirement involves testing the addition functionality. The test engineer will have to come up with a minimum set of cost values that best covers all boundary conditions and confirm that the output of the ASIC reflects the sum of the inputs provided thus far.

The third requirement is partially tested in the previous tests as the LCD was our view into the output of the ASIC. However, for completeness, this test case must cover the unique set of input vectors that will cause each of the LCD digits to display the digits 0 through 9.

The fourth requirement is covered by the RESET line into the ASIC. In our high level design the peripheral Credit Card processor, would pulse this line once the processing is successful. This will cause the ADDER output to be reset. This can be manually tested by providing a pulse to the RESET line and ensure that the output of the ASIC is then set back to zero.

4.0 – Conclusion

In this set of notes on ASIC design we have covered adequate ground on a full product development cycle to familiarize the student with both technical and business aspects of product development. We have also applied the knowledge gained previously in the notes on Digital Design to accomplish practical modules, such as Adders and Frequency Dividers and detailed how such individual modules can be integrated in a way that they accomplish a set of end user requirements for a product. We have discussed the two common design approaches namely the top-down and the bottom-up designs and when a particular approach is warranted. In passing, we have also mentioned the various job titles associated with the various tasks that were identified in product development, so as to give the student an appreciation for various job descriptions.

This sets the stage to introduce the student to the general purpose microprocessor and the application of software to realize end-user requirements. Future modules in the Accelerated Learning Series will cover topics in these areas.