Abstract

This report details the major design considerations investigated before the design and implementation of a 15-tap programmable Finite Impulse Response (FIR) Filter. Firstly, two adder structures are examined and compared in terms of area, speed and power consumption. The adder most suited for implementation in a parallel array multiplier was chosen and used to produce the structure. Following the construction of the multiplier, delay-balancing techniques were applied to the block to reduce power consumption further. The power consumption due to switching before and after delay balancing was compared to calculate the 35% power saving as a result of adopting this technique. A number of 16-bit adder configurations were designed and analysed to find a low-power fast adder for use in the filter. The solution was a 16-bit Rippled 4-bit Carry Look-ahead Adder; 70% faster than a 16-bit Ripple Adder with only 24.7% more power consumption. Finally a suitable number representation to store the FIR coefficients (specified as floating-point signed numbers) in 8-bit binary form was decided upon.

Table of Contents

Table of Figures and Tables

1 Introduction

This report investigates the power consumption of digital arithmetic circuits for use in the design and implementation of a 15-tap programmable Finite Impulse Response (FIR) filter. This section introduces the mathematical model of an FIR filter and discusses how this can be achieved in digital hardware. The report then investigates the power consumption for different architectures of the logical realisations of the mathematical components needed to implement the design. These results can then be used to produce a low-power solution.

1.1 FIR Filters

The mathematical structure of a 3-tap FIR filter is shown in Figure 1. The signal input is a number representing the magnitude of a sampled analogue signal. The z^{-1} blocks store their input and delay it by one sample period. The Co_x blocks contain the coefficients that shape the FIR filter frequency response. The design specification for the system to be designed details the coefficients for the test of the system as signed floating-point numbers.

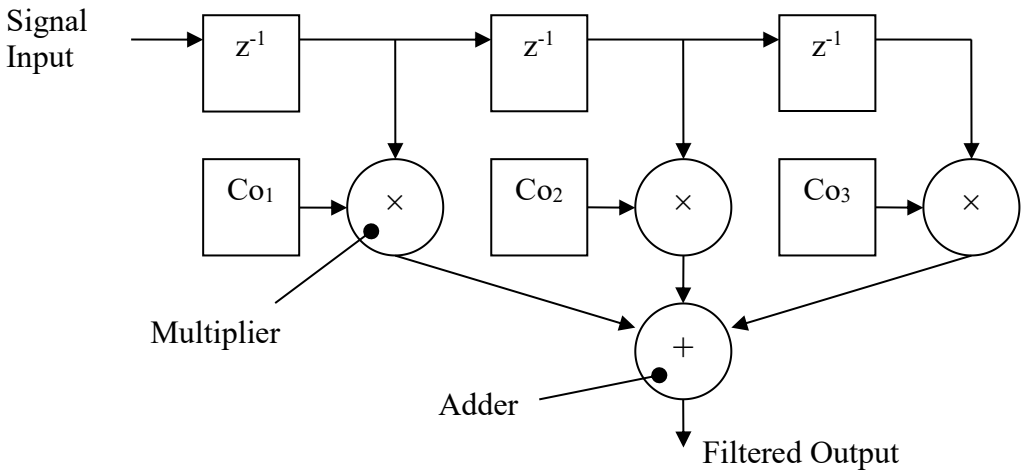


Figure 1: FIR Filter Structure

As can be seen from Figure 1, the arithmetic circuits needed for the design of a digital FIR filter are multipliers and adders, as well as storage elements. To keep the design of the system from becoming excessively complex, two input adders will be used in the system. This means that for the 3-tap filter above, the single adder shown would be created with two adders. The multipliers are to be parallel array multipliers, constructed with 1-bit full adders.

Each of the arithmetic parts needed to realize this structure as a digital system have been investigated within this report. Each part can be implemented in many different ways. This report details and compares the size, delay and power consumption results for different architectures of each component. These results enable the best solution of the optimization trade-offs to be achieved for the final design by consideration of the data contained within this report.

Another aspect of the design which has been considered within this report is the number representation scheme, and how the data is to be processed. The coefficients, which have been provided as signed floating point numbers, may not necessarily be

used as such within the digital system. The method of data conversion to a form more suited to simple and fast operation is investigated by this report. This enables the system designer to choose the appropriate arithmetic and control architecture for an efficient design.

Using the information gained from the analysis of each component of the FIR structure, an overall design has been achieved. This design uses power-reduction techniques without compromising functionality. The design, test, and detailed description of this system will be included in the second assessed report.

2 Data Processing

The specification for the FIR filter contains the 15 filter coefficients to be stored in the filter and used to carry out operations. These coefficients are shown below:

[-0.04557 0 0.06366 0 -0.1061 0 0.3183 0.5 0.3183 0 -0.1061 – 0.06366 0 -0.04557]

These numbers give the frequency response shown in Figure 2.

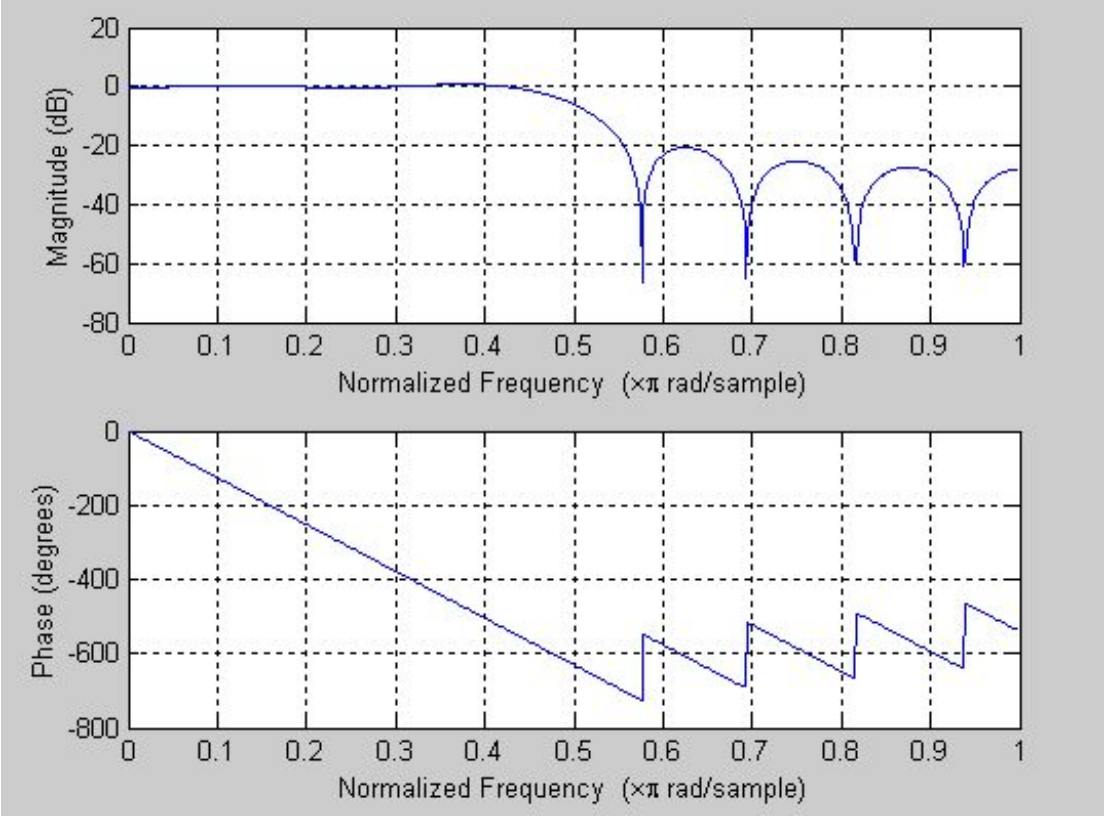


Figure 2: FIR Filter Frequency Response

These numbers are signed floating point numbers and must be converted into a form easily stored and operated upon in 8-bit binary. There are numerous possible ways to encode numbers using binary. Below is a summary of several possible formats that could be used to represent the above coefficients.

2.1 Radix-2

Radix-2 is the simplest possible encoding format. It allows positive integers in the range 0 to 2^n-1 to be encoded. In this format each bit of the binary number has a weight associated with it, as in the following example:

Weight:	128	64	32	16	8	4	2	1
Number:	1	0	1	1	0	0	1	0

128 + 32 + 16 + 2 = 178

Table 1: Radix-2 encoding

2.2 Sign Magnitude

Sign magnitude notation allows negative numbers to be encoded by allocating one bit to specify the sign of the number. If the bit is a logical '1' then the number is negative, otherwise it is positive. The remaining bits are then encoded using the radix-2 method explained above. This allows the system to store numbers in the range $-(2^n-1)$ to 2^n-1. Example:

Weight:	Sign	64	32	16	8	4	2	1
Number:	1	0	1	1	0	0	1	0

(32 + 16 + 2) = -50

Table 2: Sign Magnitude encoding

Sign magnitude has the disadvantage that there are two separate encodings for 0. In the case of an 8-bit number these are 10000000 and 00000000.

2.3 1s Complement

1s complement also allows numbers in the range $-(2^n-1)$ to 2^n-1 to be encoded. To encode a 1s complement the bit by bit complement of the binary number is taken. For example:

1s Complement	Decimal Equivalent
01111111	127
10000000	-127
11111001	-6
00000110	6

Table 3: 1s complement encoding

2.4 2s Complement

2s complement has become the standard method of storing signed binary integers. It allows the representation of numbers in the range $-(2^n)$ to 2^n-1, and has the major advantage of only having one encoding for 0. To perform 2s complement encoding the bits of the binary number are complemented, and then 1 is added. For example:

2s Complement	Decimal Equivalent
01111111	127
10000000	-128
11111010	-6
00000110	6

Table 4: 2s complement encoding

2.5 Mantissa Exponent

In order to store floating-point numbers, the standard method is to use a mantissa exponent representation. In this, a number is represented in scientific notation as

$$\text{mantissa} \times \text{radix}^{\text{sign} \times \text{exponent}}$$

Normally the radix is fixed and only the mantissa, sign and exponent are encoded. Using this method up to 2^n numbers can be encoded, although obviously with limited accuracy.
The IEEE standard 754 defines a standard for binary floating-point arithmetic using either 32 or 64 bit numbers. The 32-bit format is below:

S	Exponent (8 bits)	Mantissa (23 bits)

Table 5: Mantissa Exponent encoding

The range of numbers which can be stored using this standard is approximately 1.8×10^{-38} to 3.40×10^{38}. However, floating point arithmetic is not feasible for this project due to the complexity of the adders and multipliers which would be required. For example, a block diagram of an IEEE 754 adder is shown below in Figure 3:

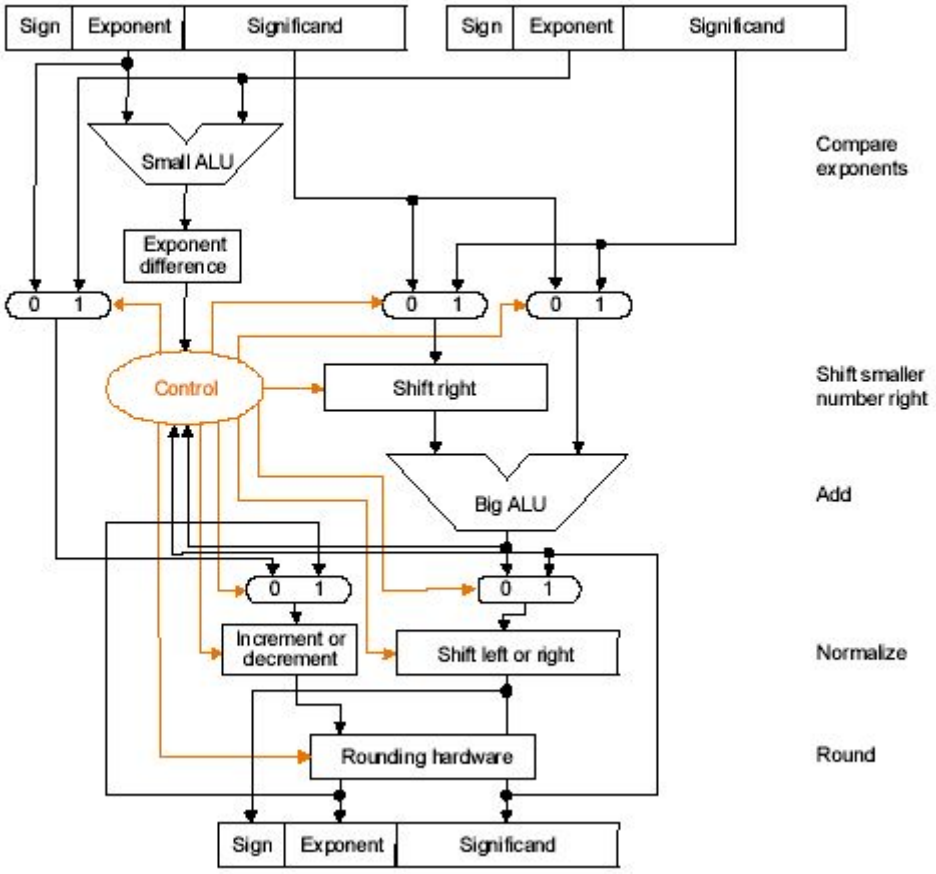


Figure 3: IEEE 754 Adder

2.6 Final Number Representation

Sign-magnitude representation of the coefficients has been chosen as the Number representation to be used to store the coefficients in this project. Mantissa Exponent encoding was rejected on the basis that the Multipliers and Adders involved would be too complex to build and apply power reducing techniques to. Radix-2 cannot represent negative numbers and so is not suitable for this filter. While 1s complement and 2s complement are both sound number representations, the Adders and Multipliers needed to implement operations with these numbers are more complicated than those needed for signed magnitude operations, and so would be harder to make more power efficient.

After sign-magnitude representation was chosen, the original coefficients were converted into a form easily stored in this encoding (i.e. positive or negative integers). To make this conversion, the following formula was used (except in the case of zero coefficients, which are left stored as zero, "00000000"):

$$Signed\_Magnitude\_Coeff = Round(Coeff * 128)$$

The coefficients after this conversion are shown below:

[-6 0 8 0 -14 0 41 65 41 0 -14 0 8 0 -6]

For example -6 is stored as 1000 0110. The frequency response of the new filter is shown in Figure 4.

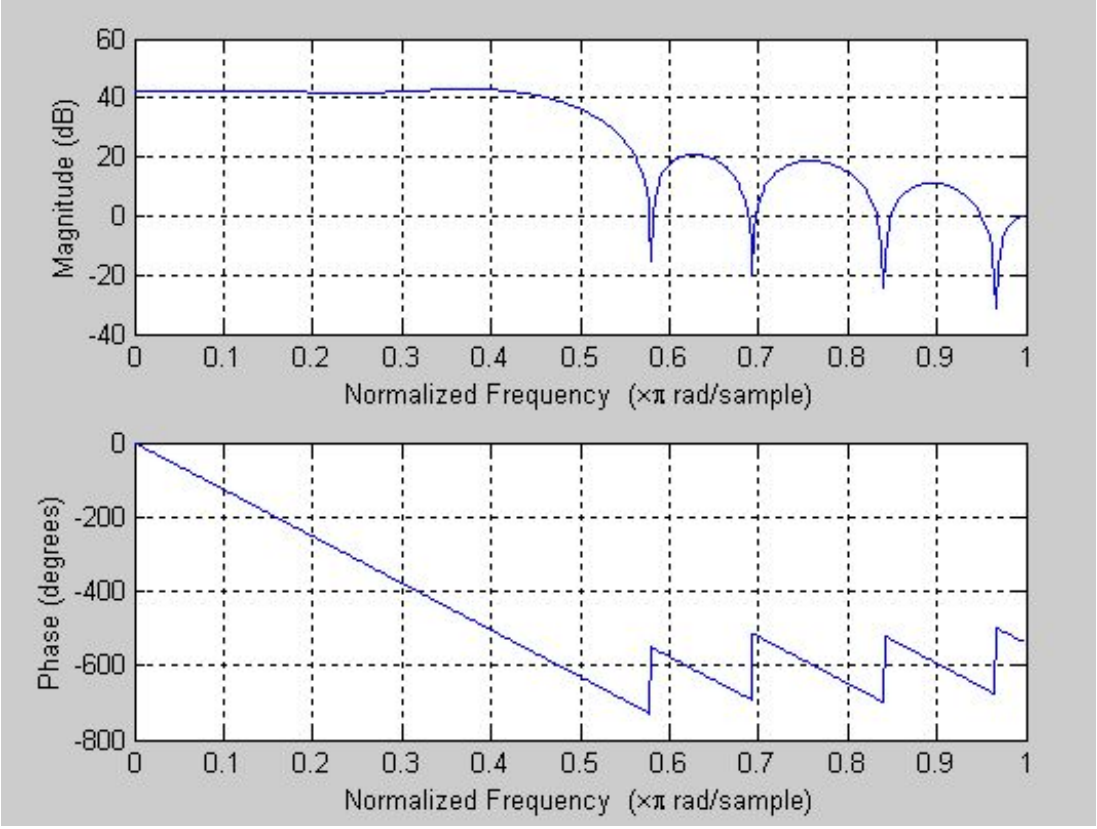


Figure 4: FIR Filter Frequency Response (after Number conversion)

If Figure 4 is compared with the original response in Figure 2, it can be seen that this number representation accurately recreates the FIR filter response, with the addition

of a 20dB gain factor, which can be easily removed after the filter by a simple analogue attenuator.

The final binary representation of the coefficients is shown in Table 6.

Coefficient Number	Original Value	Converted Value	Binary Representation
0	-0.04557	-6	10000110
1	0	0	00000000
2	0.06366	8	00001000
3	0	0	00000000
4	-0.1061	-14	10001110
5	0	0	00000000
6	0.3183	41	00101001
7	0.5	65	01000001
8	0.3183	41	00101001
9	0	0	00000000
10	-0.1061	-14	10001110
11	0	0	00000000
12	0.06366	8	00001000
13	0	0	00000000
14	-0.04557	-6	10000110

Table 6: Final Number Representation

3 Adder Comparison

In this section two adder designs; Ripple-through and Carry Look-ahead, are analysed in terms of their speed, area and power consumption. The aim of this investigation is to select the adder best suited for implementation in an array multiplier, to be used in the FIR Filter system. In this case, the multiplier must multiply two 8-bit numbers to give a 16-bit result. The role of the adder in the parallel array multiplier will be discussed later. An adder must also be selected to sum the 16-bit outputs of the 8-bit multiplication.

The analysis of the Adder designs was produced in OrCAD Capture CIS, a schematic capture program, which operates on the principle that circuits are built up by "dropping" gates into a design and connecting them with wires in an interface much like a commercial paint program. The advantage of working in the capture program for this analysis is that the circuits being produced are made from actual 74 series logic chips, as detailed in the specification. Therefore when the logic in larger adders becomes larger (e.g. ANDing of 5 or more terms is required) the level of logic increases, since the 74 series logic family does not feature gates of all sizes.

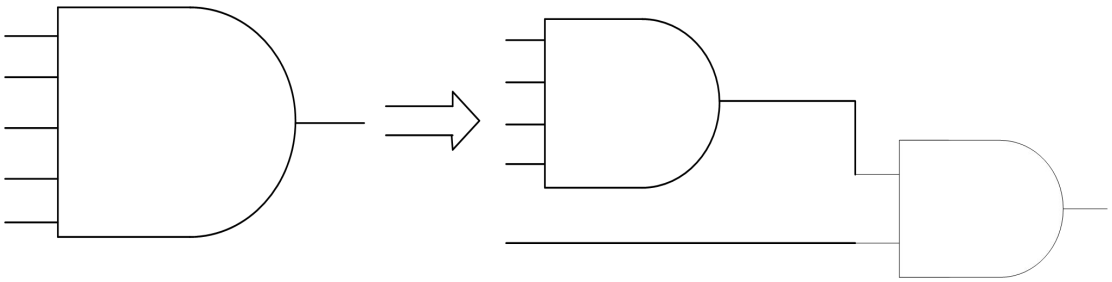


Figure 5: Logic modelling in Capture CIS

The benefit of using Capture CIS for the analysis of the Adders can be seen in Figure 5. Instead of using a 5-input AND gate, with one gate delay (as would be inferred from a VHDL model) the 5 input AND gate must be split into two available gates, a 4 input AND gate, and a 2-input AND gate. This is a more accurate model, as there are two gate delays for the function, and the potential for more power consuming transitions to be modelled (as will be discussed in more detail in section 3.4.1).

3.1 Ripple Adder Structure

This section details the circuit diagram of a Ripple carry adder and the structure of the adder as the number of bits increases. The fundamental principle behind a Ripple adder is that the first stage calculates the sum and carry-out values from the adder inputs, and the carry-out signal from this stage is then rippled into the next stage for the calculation of the next most significant bit. The structure of a 1-bit Ripple carry adder is shown in Figure 6. The inputs to the adder are A and B, one-bit inputs, and the carry-in input, Cin, from the previous stage. If this is the first stage the Cin input is tied to ground. The outputs of the system are the Sum output and the carry-out signal, Cout, to the next stage. If this is the last stage, and positive numbers are being added, the Cout signal is an indicator of overflow.

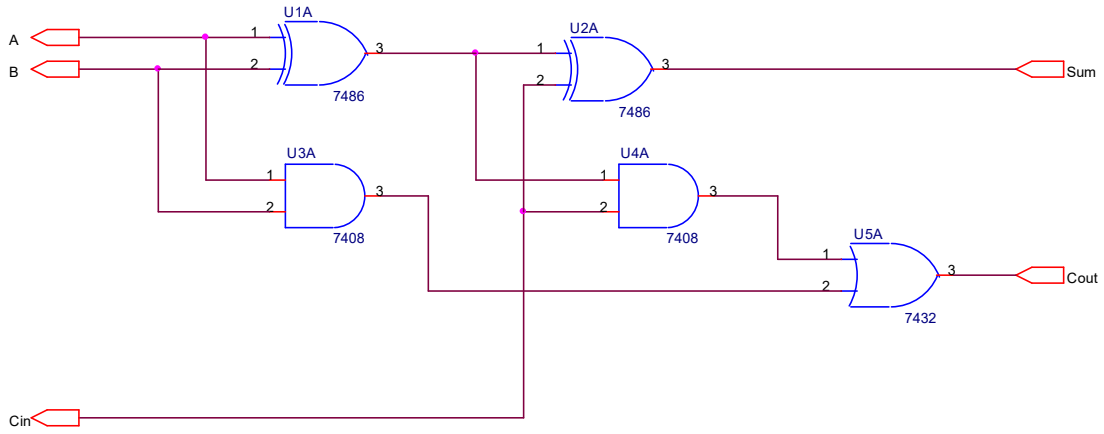


Figure 6: 1-bit Ripple Adder

Ripple adders are cascaded together to create a complete adder circuit for the required number of bits. The Cout signal from the previous stage is connected to the Cin of the next stage; the example of a four-bit adder shows this process and is given in Figure 7. The time for the addition to complete depends upon the time for the carry out signal to propagate from the first stage to the output at the last. This is known as the critical path, which is always the longest signal path in a section of combinational logic.

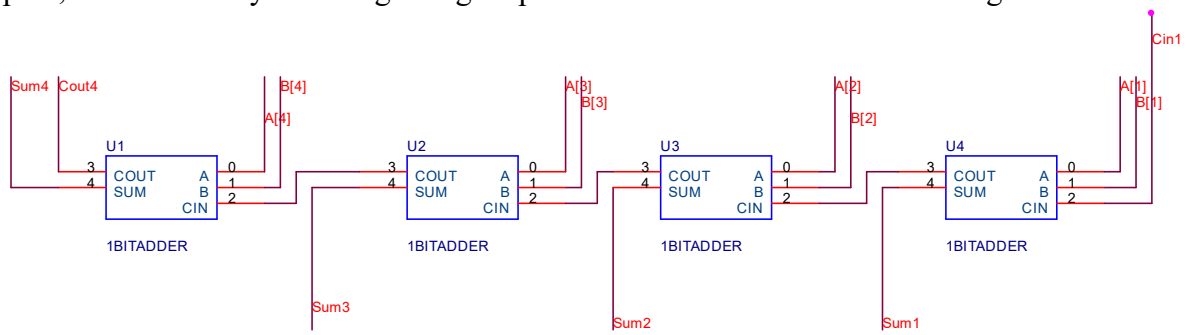


Figure 7: 4-bit Ripple Adder

3.2 Carry Look-ahead Adder structure

The Carry Look-ahead adder has a different structure to the Ripple adder. The Carry Look-ahead structure contains extra logic between the adder stages to predict the carry out signal. This makes the addition process faster and independent from the ripple of the carry signal through previous stages. The propagation time for the carry signal in the Ripple adder exceeds that of the Carry Look-ahead structure for adders of eight bits and above. The time saving of the Carry Look-ahead structure increases with the number of bits the adder operates on.

A Carry Look-ahead adder is made up of a series of Partial Full Adders (PFAs) and Carry Look-ahead logic. The PFA is very similar to a cut down 1-bit Ripple adder and is shown in Figure 8. The inputs to the adder are A, B and Cin as with the 1-bit full adder in the Ripple Adder. The outputs of the PFA are different however; P and G are the Propagate and Generate signals respectively, and S is the Sum output.

The P and G signals are used by the Carry Look-ahead logic which is external to the PFAs. This enables the system to predict the carry out signals for adders further down the line faster than the Ripple process would generate them.

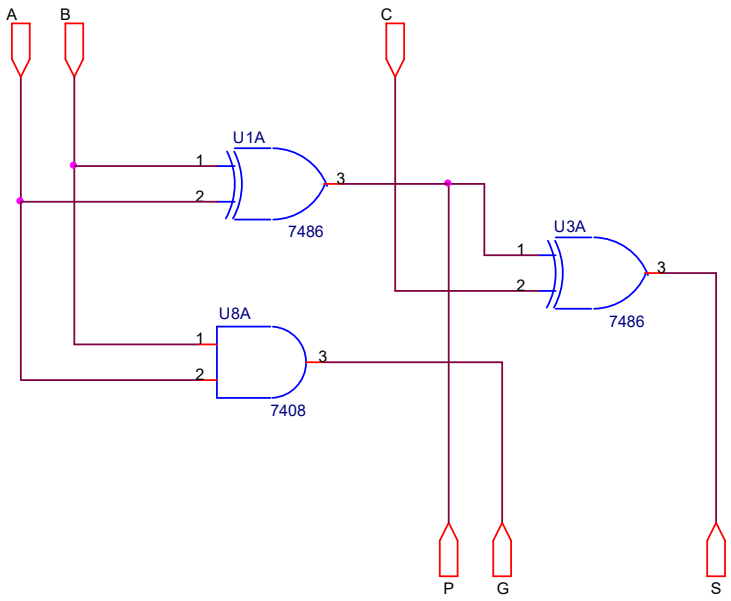


Figure 8: 1-bit Partial Full Adder (PFA)

The structure of a 4-bit Carry Look-ahead adder is shown in Figure 9. The PFA blocks are the rectangular structures at the top of the diagram. The carry logic takes the carry-in signal to the entire adder, as well as the propagate and generate signals to predict the carry-out as early as possible, without waiting for the carry signal to propagate through each stage.

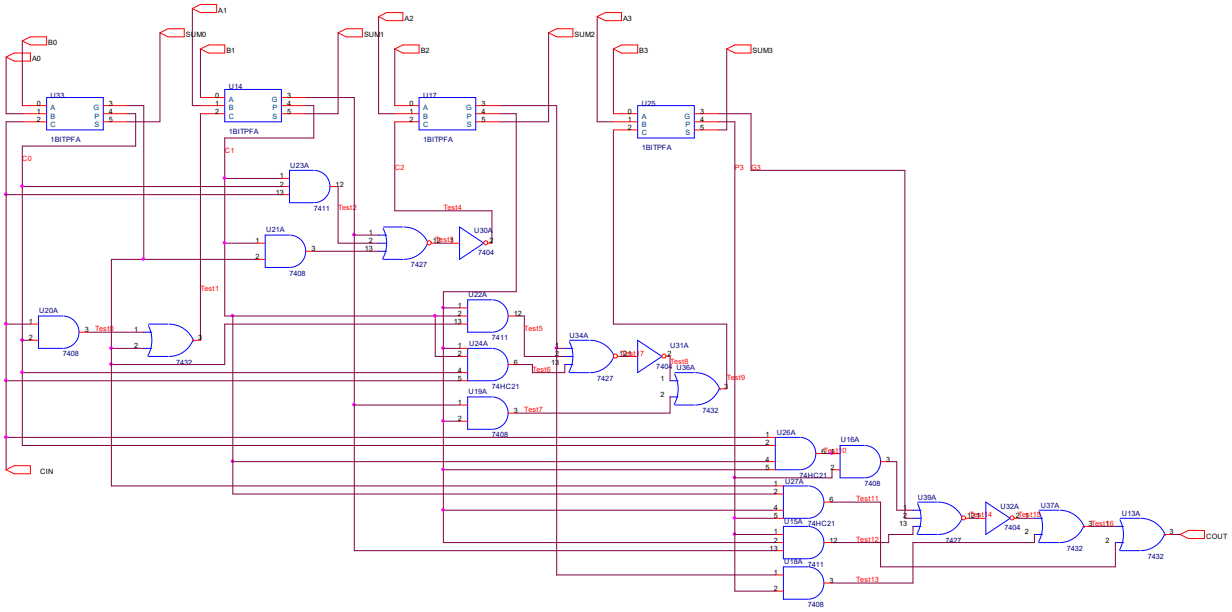


Figure 9: 4-bit Carry Look-ahead Adder

It is obvious from these early investigations that the Carry Look-ahead (CLA) adder will calculate the result of the addition faster than a Ripple adder, as a result of the extra carry prediction logic. However, this logic makes the CLA configuration considerably larger than the Ripple carry Adder.

3.3 Area Analysis

To analyse the area difference between the two adder configurations, the required number of gates in each adder were calculated. This figure was then used to find the

number of transistors needed to produce the each of the two Adders at various sizes. The results are shown in Table 7.

Adder Size	Ripple	Carry Look-Ahead
1-Bit	34	22
4-Bit	136	204
8-Bit	272	404
16-Bit	544	804

Table 7: Adder Transistor Count

To produce the larger CLA configurations, 4-bit Carry Look-ahead adders were chained together using master Propagate and Generate signals to produce the carry-in to the next 4-bit Carry Look-ahead adder.

The Boolean equations used to link one 4-bit adder to the next are as follows:

$$PG = P_3 \bullet P_2 \bullet P_1 \bullet P_0$$
$$GG = G_3 + P_3 \bullet G_2 + P_3 \bullet P_2 \bullet G_1 + P_3 \bullet P_2 \bullet P_1 \bullet G_0$$
$$Cout = GG + PG \bullet Cin$$

This is the only practical was to create CLA Adders of this size. As can be seen in Figure 9 the carry logic would become intolerably large for CLA Adders above 4-bits.

3.4 1-bit Multiplier Adders

The 8-bit multipliers included in the FIR filter design are parallel array multipliers, produced with 64 1-bit full adders. This section describes the power analysis completed to decide which adder structure to implement in the multipliers.

3.4.1 Power Analysis

There are three elements to power loss in digital circuits, as can be illustrated by the following equation [1]:

$$P = P_{switching} + P_{short-circuit} + P_{leakage-current}$$

The most dominant form of power loss in a CMOS circuit is the power consumed by the switching of gates, $P_{switching}$ in the equation above. The causes of this switching power are given by this equation [1]:

$$P_{switching} = \frac{1}{2} V_{dd}^{\ 2} f_{clock} C_L E_{sw} = \frac{1}{2} V_{dd}^{\ 2} f_{clock} C_L P_{0\to 1}$$

Where V_{dd} is the supply voltage, f_{clock} is the clock frequency, C_L is the load capacitance and E_{sw} is the switching activity. Since we cannot alter the supply voltage in this project, and the minimum clock frequency is constrained by the operation of the system, the only way to reduce power is by reducing the switching activity. In fact, power loss mostly occurs when a gate output switches from logic '0' to logic '1', so the switching activity can be replaced by the probability of a '0' to '1' transition, $P_{0\to 1}$.

This means that to analyse a circuit for power consumption, one can look at the number of 0 to 1 transitions in a sample behaviour. Estimating the amount of 0-1 gate transitions is difficult when considering large designs due to glitches and the complexity of the design. Unknown input statistics also pose a problem for analysis, and so when the behaviour of the input signals in normal operation are not known, a random input pattern is applied.

3.4.2 OrCAD

The two different adder structures were schematically designed in Capture CIS, which has the ability to produce a SPICE netlist. The designs were then stimulated and analysed within pSPICE, producing waveforms that can be inspected by the user. However pSPICE does not have the implicit capability to detect and count the number of 0-1 transitions on gates, especially those deep within the design hierarchy. It was suggested to use 4-bit binary counters to detect the 0-1 gate transitions. This was achieved by creating a part from a 4-bit binary counter and adding these within the design being examined. An example is shown in Figure 10. The output of every gate was fed into the clock input of an incrementing counter that is reset at the start of the stimulus. Every time a 0-1 transition occurs, the counter will increment once, producing a set of numbers from all the gates to sum at the end of the simulation. This provides an accurate representation of the switching activity, and therefore power consumption, of the design.

The test setup is reproduced in Figure 10. A typical test schematic is shown in Appendix 1.

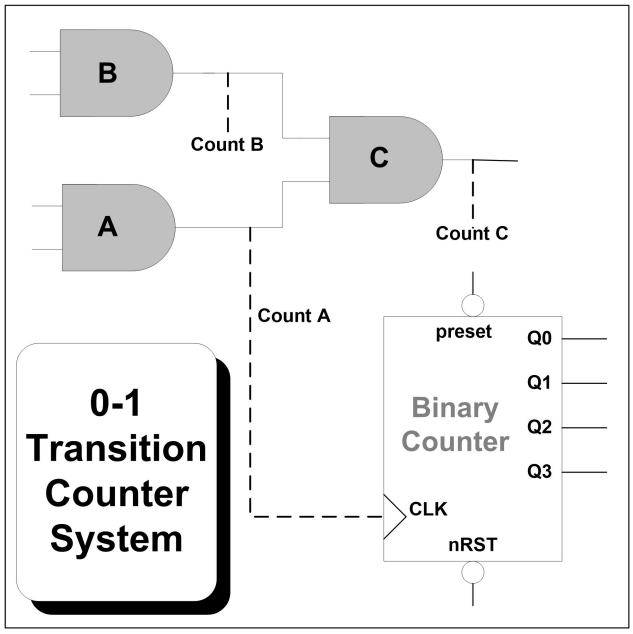


Figure 10: Transition monitoring setup

The stimuli for each test were applied using the Capture CIS digital stimulus part from the *Source.olb* library file.

There were three main limitations with this technique:

- The maximum number of stimuli that could be applied at any one input was 19. This is a limitation of pSPICE.
- If a glitch occurred, the binary counters' clock width could be violated, resulting in a persistent error.
- The technique was labour intensive; adding the necessary counters to the design was a time consuming, and error prone task.

By applying grey code to one of the adder inputs and keeping the other input constant it was possible to keep glitches to a reasonable level and good, usable results were

achieved. The detailed results for the alternative architectures can be seen in Appendix 2.

Power analysis of a 1-bit Ripple adder versus a 1-bit Carry Look-ahead adder would be trivial, in fact at 1-bit size, they are identical in structure. However to estimate the performance of the 1-bit adders in a large multiplier structure (where they are chained together) analysis of larger adders is completed. 1-bit, 4-bit and 8-bit adders of both designs were analysed, the overall results are shown in Table 8 , and a more detailed tabulation of the results can be found in Appendix 2.

Adder Size	Ripple-Through	Carry Look-Ahead
1-Bit	5	5
4-Bit	26.55	31.33
8-Bit	26.2	41.8

Table 8: Average 0-1 transitions per test

To generate stimulus, one input to the adder was kept constant, while several possible values were applied to the other input. The same tests were performed on each of the designs.

Comparing these results, and then extrapolating for 16-Bit and 32-Bit adders it is obvious that the power consumption of the Carry Look-ahead will be considerably more than that of the Ripple adder, an estimated 49% difference when considering a 16-Bit Adder.

Adder Size	Ripple	Carry	Percentage Difference
1 Bit	5.00	5	0%
4 Bit	26.55	31.33	15.25%
8 Bit	26.20	41.80	37.32%
16 Bit*	26	51	49%
32 Bit*	26	61	57.38%

Table 9: Adder Transition Test Comparison (*Estimates)

These results, although displaying the trends expected of the two adder designs, are flawed by the small number of stimulus (and the data dependence intrinsic to the process) that were applied to the adders. Secondly, pSPICE has several limitations with regards to simulation of this type. As a result, the OrCAD layout was used to generate a VHDL netlist, to be simulated with VHDL testbenches in Modelsim to give more comprehensive and representative results.

3.4.3 VHDL

As a result of the limitations experienced using the OrCAD simulation suite, it was decided to utilise the increased flexibility and capabilities of a VHDL testbench simulation. It must be stressed that the simulations were carried out on the VHDL netlist extracted from OrCAD, and so this method was still testing the actual logical structure of the Adders, and not an approximation of it. For example, the logic for the 4-bit CLA was exactly as shown in Figure 9. The VHDL language also enabled the simple implementation of a pseudo-random input sequence. This reduced any dependency on the input data, which may have existed in the OrCAD testing. To generate this pseudo random sequence a Linear Feedback Shift Register (LFSR) was used [2].

The LFSR uses XOR feedback to generate a series of sequential states which appear random. The LFSR was made wide enough to give an effective number of pseudo random inputs for a long period of time in order to get useful data.

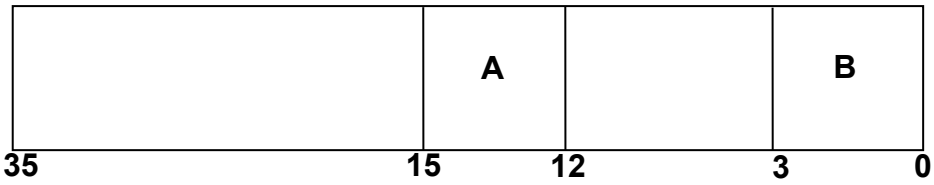


Figure 11: 4-bit number placement in 36-bit LFSR

Figure 11 shows how the two four bit numbers used to stimulate the 4-bit adder designs were located in the LFSR output pattern. Since the pattern has $2^n - 1 = 2^{36} - 1 = 68{,}719{,}476{,}735$ states, this setup gives a more than adequate set of stimuli for the designs to add while the transitions of the system are being examined.

The transitions of the system are counted in a similar way to the counters employed in the OrCAD testing. All outputs of the gates in the design are labelled as signals and the 0-1 transitions are counted by a set of counters placed in the VHDL file after netlisting. The counters all have the form:

```
process(Y(0))is
begin
  if(Y(0)'event and Y(0) = '1') then
     Count0 <= Count0 + 1;
  end if;
end process;
```

This code fragment shows the Counter "Count0" that is incremented on every 0-1 event on the signal Y(0). A master Sum signal adds together all the Count values to give a running total during simulation. All Counters are initialised to have a Count of zero at the start of a simulation run. Simulation of a 0-1 transition on Y(0) and the resulting increment of Count0 is shown in Appendix 3.

The same pseudo-random sequence was applied to both the 4-bit CLA and 4-bit Ripple Adders for 1ms and 5ms. The total detected 0-1 transitions are shown in Table 10.

	4-bit Ripple Adder	4-bit CLA	Percentage difference
Transitions after 1ms	**1,345**	**2,423**	Ripple 44.5% less
Transitions after 5ms	**9,090**	**16,857**	Ripple 46.1% less

Table 10: 4-bit CLA results

The Ripple Carry adder displays lower power over both times, as is consistent with the results of the OrCAD analysis and also first principle calculations. The percentage difference in transitions is shown in the final column, illustrating that the Ripple Adder is approximately 45% more power efficient than the larger Carry Look-ahead structure. The main advantage of the Carry Look-ahead adder lies in its speed. For

adder designs of 8-bits and above, the Ripple configuration takes far too long to generate carry-in signals at the higher order bits that the design becomes unusable for our system.

From the previous analysis the obvious choice for the 1-bit adder for use in the 8-bit multiplier is the Ripple adder. The 8-bit multipliers in the system will each contain 64 1-bit adders and the aim is to keep the power consumption of the multiplication operation as low as possible, this will be discussed in section 4. For this reason, the Ripple adder is chosen to be used in the parallel array multiplier, since it is fast enough for 1-bit additions and provides the lowest power solution.

3.5 16-bit Summing Adders

A 16-bit adder is needed in the design for the addition of the outputs from the multiplications (the 8-bit coefficient multiplied by the 8-bit sample gives a 16-bit result). These multiplications give 15 16-bit numbers (when the filter is in 15-tap mode) and these need to be summed to give the filter output for that state. It is important therefore that the 16-bit Adder be fast (there are 15 consecutive additions at worst case) but also as low-power as possible.

3.5.1 16-bit Ripple Adder

The first 16-bit adder to be considered was the 16-bit Ripple adder, as shown in Figure 12. The rectangular blocks are Full Adder cells, previously shown in Figure 6. This Ripple adder was produced in OrCAD and extracted for simulation in Modelsim with a VHDL testbench.

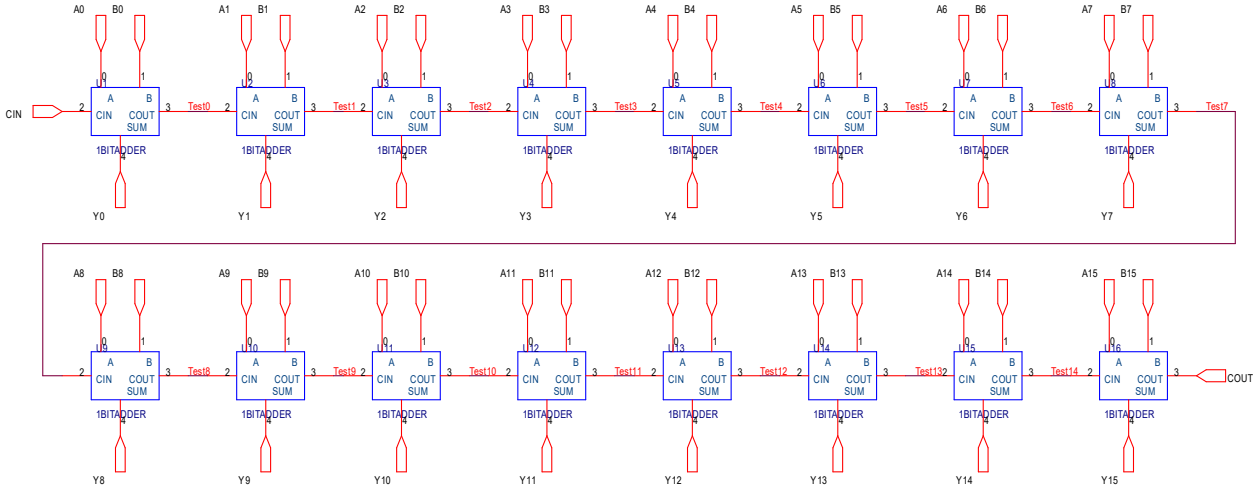


Figure 12: 16-bit Ripple Adder

As with previous power analysis, signals inside the system were labelled and counters were instantiated in VHDL to measure transitions on the internal nets of the structure, as well as the outputs. The worst case propagation delay through the adder was also measured. A test setup identical to the VHDL testing on the 4-bit adders described in section 3.4.3 was used. A 36-bit LFSR was used to create a pseudo random sequence of numbers 36 bits wide, from which two 16-bit vectors were connected to the inputs of the adder. The bit connections are shown in Figure 13.

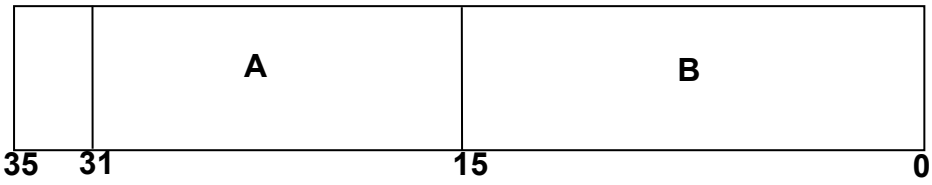


Figure 13: 16-bit number placement in 36-bit LFSR

The simulation was run for 1ms and 5ms. The results are summarised below:

	Transitions after 1ms	Transitions after 5ms	Propagation delay (ns)
16-bit Ripple Adder	**9,905**	**62,877**	**760**

Table 11: 16-bit Ripple adder simulation results

Some simulation plots from the power analysis are reproduced in Appendix 4.

3.5.2 16-bit Full Carry Look-ahead Adder

Following simulation of the 16-bit Ripple adder, a 16-bit Full CLA adder was constructed in OrCAD and a VHDL netlist was extracted. The structure of the 16-bit Full CLA adder is shown in Figure 14. The large regular blocks are 4-bit Carry Look-ahead adders (see Figure 9) and the surrounding logic is predicting the carry into each 4-bit block depending upon the Cin input, and the master propagate and generate signals.

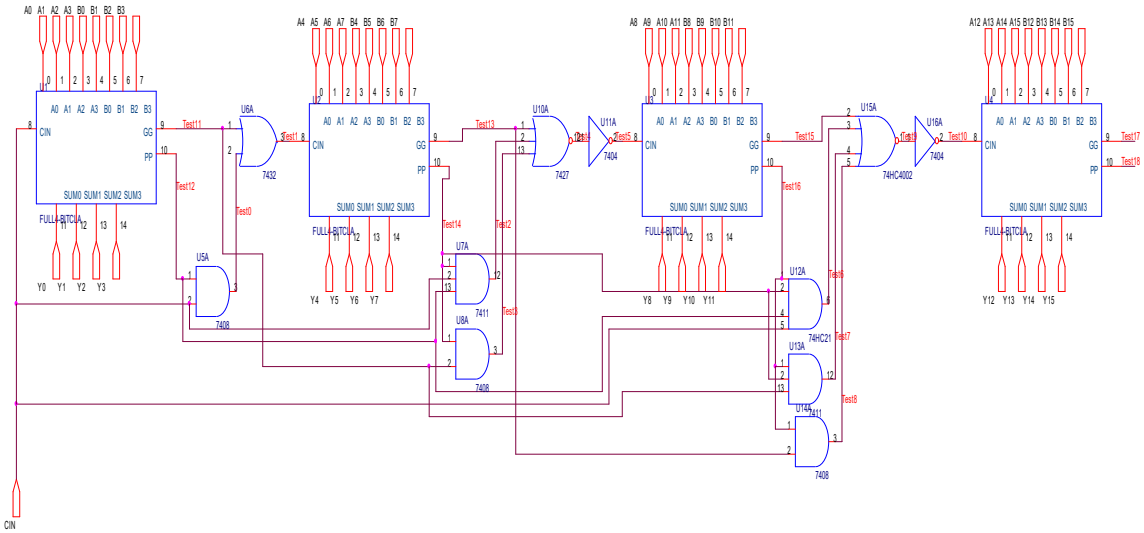


Figure 14: 16-bit Full Carry Look-ahead Adder

This adder configuration was simulated to extract power consumption and delay data through the Full CLA structure. The results are shown in Table 12.

	Transitions after 1ms	Transitions after 5ms	Propagation Delay (ns)
16-bit Full Carry Look-ahead Adder	**13,087** (32% more than Ripple)	**88,976** (41.5% more than Ripple)	**207** (360% times faster than Ripple)

Table 12: 16-bit Full CLA Adder results

The results indicate that although the CLA adder is much faster than the Ripple structure, it consumes a large amount of power in comparison to the simpler 16-bit Ripple structure.

It was decided to investigate whether there were any modifications which could be made to the CLA adder to preserve some of its speed advantage but also reduce the power consumption at large size additions. The result was the 16-bit Rippled 4-bit Carry Look-ahead adder.

3.5.3 16-bit Rippled 4-bit Carry Look-ahead Adder

As a result of looking at the power consumption through switching of a full 16-bit CLA structure (made up of four 4-bit CLA adders and Carry Look-ahead logic), it was decided to see if the power could be reduced somewhat by rippling 4-bit CLAs together instead of generating the predictive carry signals between each 4-bit block. This choice was made as an attempt to reduce power loss between the stages while still producing a faster solution than the 16-bit Ripple adder.

The structure of the Rippled 4-bit Carry Look-ahead adders which produce a 16-bit adder is shown below in Figure 15. The carry-out from one 4-bit stage is rippled into the carry-in of the next 4-bit stage, removing the need for Carry Look-ahead logic between 4-bit stages and so reducing switching and power consumption during an addition.

The adder was simulated to check the functionality was correct. Analysis was then performed on the structure to calculate the switching activity and therefore the power consumption of this 16-bit adder configuration. The propagation delay was also measured to provide an indication of the speed of the design. This enabled comparisons between the alternative adder architectures to be drawn. The results were as illustrated in Table 13.

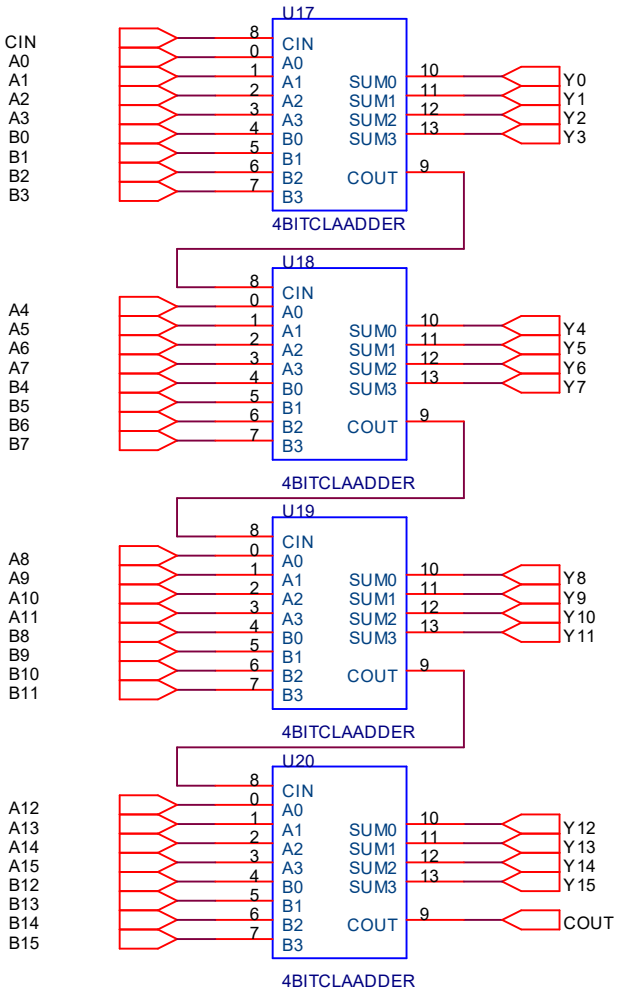


Figure 15: 16-bit Adder (4 Rippled 4-bit CLAs)

	Transitions after 1ms	Transitions after 5ms	Propagation Delay (ns)
16-bit Rippled 4-bit CLA Adder	**13,064** (31.8% more than Ripple, 1% less than Full CLA)	**86,977** (38% more than Ripple, 2% less than Full CLA)	**450** (70% faster than Ripple)

Table 13: 16-bit Rippled 4-bit CLA Adder results

At first it was thought that these results were not encouraging; the power saving is minimal compared with the loss of speed from 207 ns to 450 ns propagation time. However after some investigation into the simulation results of the Modelsim runs, it was discovered that much of the power loss was due to the rippling through of new transitions after the carry-in from a previous stage arrived. Each 4-bit adder had already settled down to an output state before the carry-out from the previous stage arrived, when it did, the adders performed the additions again, causing more transitions, and more power loss.

To decrease this loss, it was decided to implement delay balancing on the 16-bit adder to reduce the number of transitions.

3.5.4 Delay Balanced 16-bit Rippled 4-bit CLA Adder

The function of the delay balancing technique is to ensure that all inputs to a system arrive at the same time. This removes unneeded transitions caused when the system settles, only to be switched again by a late arriving input signal. In the case of the Rippled 4-bit adders, it is a case of balancing the arrival of the inputs to each 4-bit CLA Adder to coincide with the arrival of the carry-in signal from the previous 4-bit CLA. For example, the first CLA does not need any delay balancing because the carry-in signal and the addition inputs are available at the same time. However, the addition inputs A and B of the second 4-bit CLA must be delayed by the time it takes for the carry-out signal to be generated by the first 4-bit CLA.

The carry-out propagation time was measured by pSPICE simulation and created by making a delay block of buffers, as shown in Figure 16.

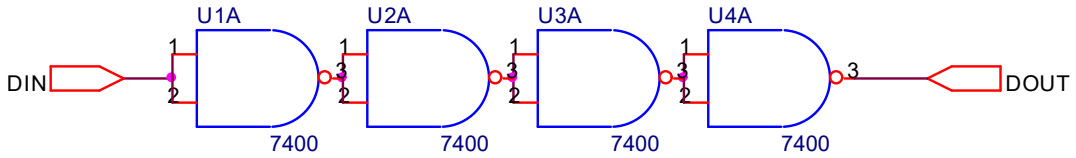


Figure 16: Delay block for CLA delay balancing

The delay balanced 16-bit Rippled 4-bit CLA adder is shown in Figure 17. The large rectangular blocks on the right of the figure are the 4-bit CLA blocks as shown in Figure 9. The smaller blocks on the left are the delay blocks as seen above in Figure 16. The first 4-bit CLA adder in the chain needs no delays as the adder inputs and the carry-in signal all arrive at the same time. The adder inputs to the second block however are delayed by the time taken for the first adder to generate the carry-out signal. The third adder inputs are delayed by two delay blocks (the time for the first adder to generate the carry-out to the second adder, added to the time taken for the second adder to generate carry-out with this carry and its adder inputs). By the same token the final adder has three delay blocks placed on its adder inputs.

Power analysis was performed on the delay balanced 16-bit Rippled 4-bit CLA adder to calculate the power saving due to the application of the delay balancing power reduction technique. The propagation delay through the adder was also measured.

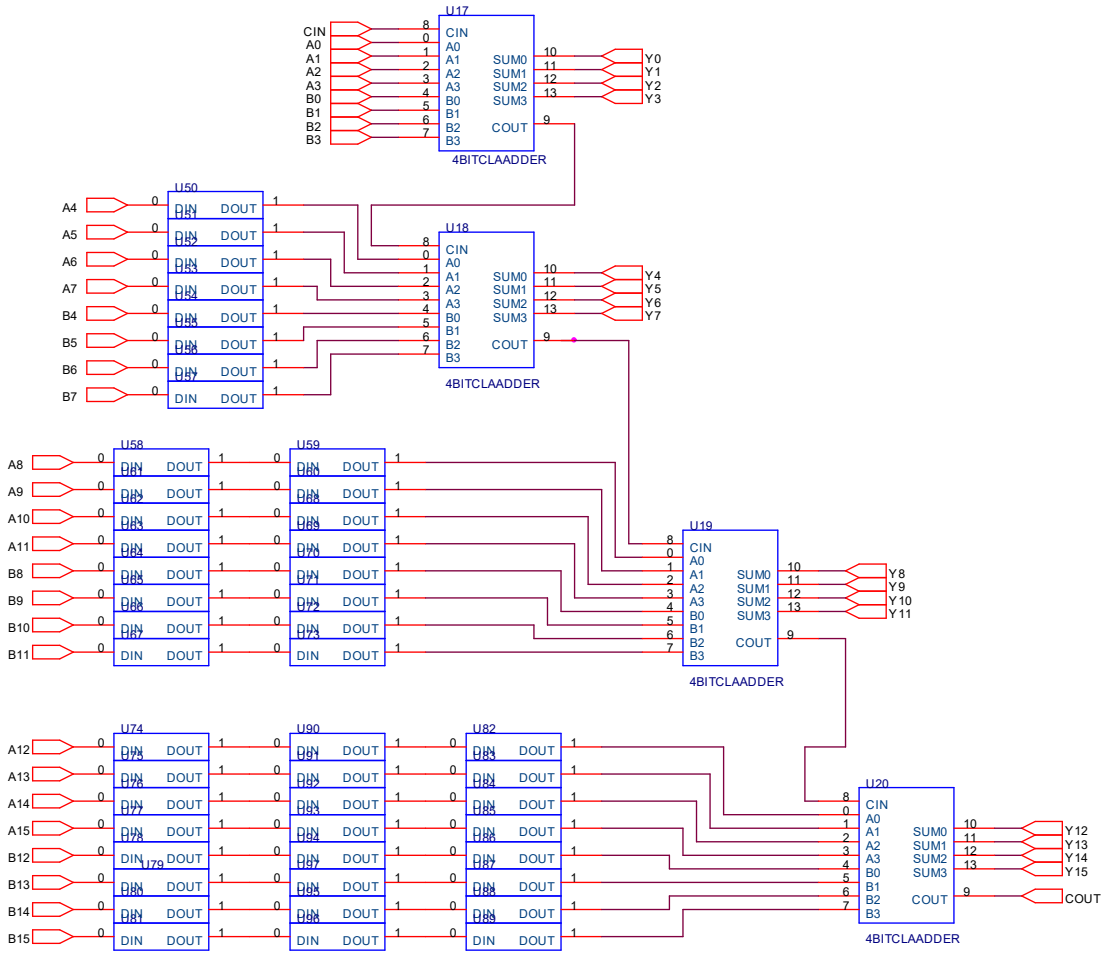


Figure 17: Delay Balanced 16-bit Rippled 4-bit CLA Adder

	Transitions after 1ms	Transitions after 5ms	Propagation Delay (ns)
Delay balanced 16-bit Rippled 4-bit CLA Adder	**12,231** (7% less than Rippled CLA, 7% less than Full CLA, 23% more than Ripple)	**78,398** (10% less than Rippled CLA, 12% less than Full CLA, 24.7% more than Ripple)	**450** (same as Rippled CLA, 70% faster than Ripple)

Table 14: Delay balanced 16-bit Rippled 4-bit CLA adder results

It can be seen from Table 14 that the application of delay balancing to the 16-bit Rippled 4-bit CLA adder has made this configuration much more attractive. The result is a fast adder with power consumption less than a Full CLA adder structure. The adder is 70% faster than a 16-bit Ripple adder and only consumes 24.7% more power. Since there will be 15 16-bit adders cascaded to form the final parallel FIR structure, a balance of speed and low-power is crucial. It is believed that this is achieved by the 16-bit Rippled 4-bit CLA adder.

4 Parallel Array Multiplier

4.1 Unbalanced Parallel Array Multiplier

The Parallel Array Multiplier to be constructed for the FIR filter design is an 8-bit multiplier made up of 64 MCells (Multiplier cells) in a Matrix-style structure. MCells contain a 1-bit full adder (Ripple configuration) and an AND gate. The MCell is shown in Figure 18.

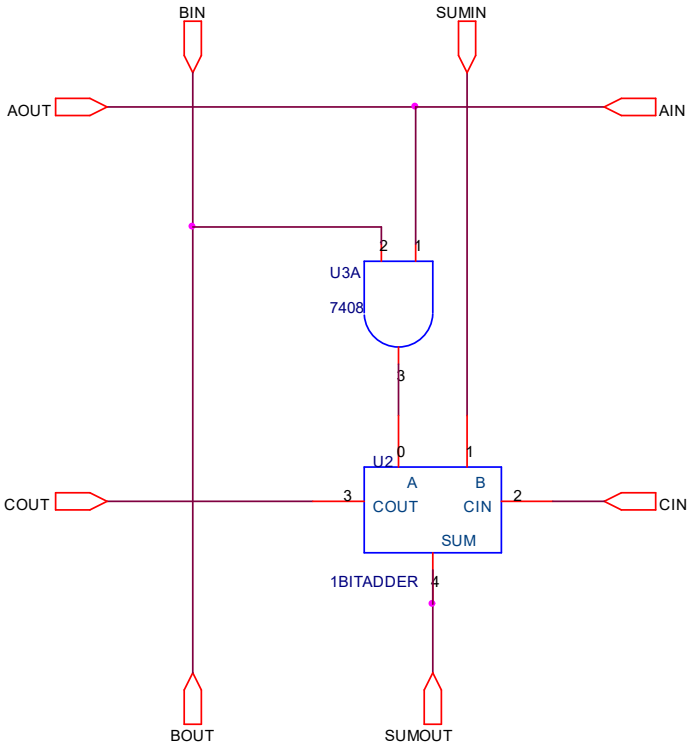


Figure 18: Multiplier MCell structure

The MCells are connected together to form a multiplier. A 2-bit multiplier is shown in Figure 19. The inputs are connected to the system from the right edge and top and the MCells are arranged in the structure shown to give the multiplication function. An 8-bit multiplier was constructed using this system and a VHDL netlist was extracted from the OrCAD schematic layout. A VHDL testbench was created to test the functionality of the multiplier. The simulation results from the functional test of the multiplier can be seen in Appendix 5. Following the functional testing of the block, the multiplier was analyzed for power consumption from 0-1 transitions on the internal nodes and on the outputs.

The VHDL netlist of the multiplier was edited in the same way as the Adders examined in section 3.4 and 3.5 to include counters, triggered by 0-1 transitions on the labelled nodes in the circuit. The top-level testbench contained a 36-bit LFSR to generate pseudo-random inputs to the multiplier, and also counters to monitor transitions on the output ports. The simulation was run for 1, 5 and 10ms and the transitions on all the nodes and outputs were summed at each time to give a view of the switching activity inside the multiplier.

The results of the VHDL power simulation are reproduced in Table 15.

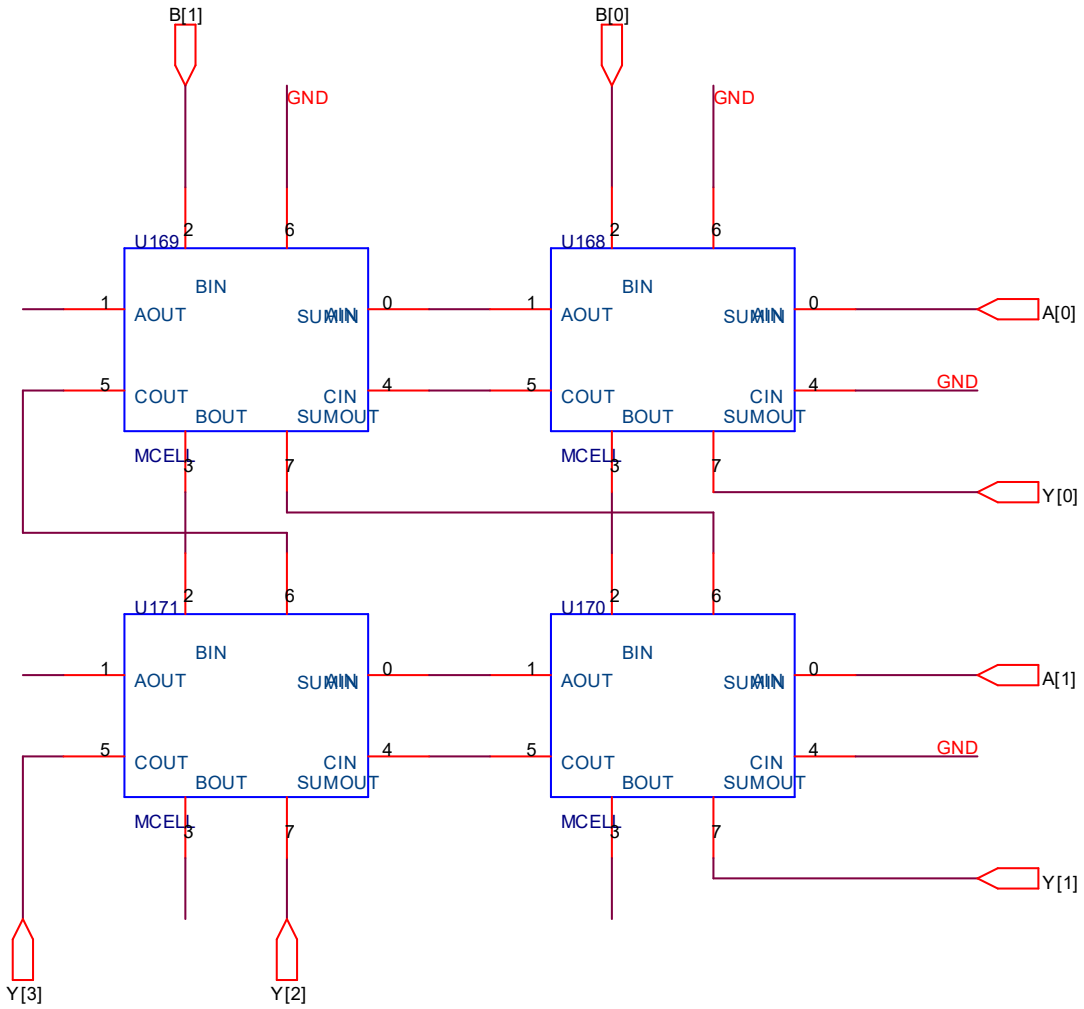


Figure 19: 2-bit Parallel Array Multiplier

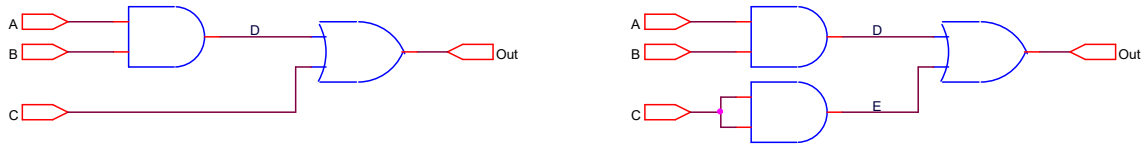
	Transitions after 1ms	Transitions after 5ms	Transitions after 10 ms
8-bit Parallel Array Multiplier	**104,506**	**576,104**	**1,130,731**

Table 15: 8-bit Parallel Array Multiplier results

The results display the switching activity inside and on the output ports of the 8-bit Multiplier. To attempt to reduce the power consumption of the multiplier, delay balancing was applied to the structure to reduce the number of transitions. Some simulation plots from the power analysis are reproduced in Appendix 6.

4.2 Delay Balanced Parallel Array Multiplier

Delay balancing reduces the power consumption of a system by removing all unnecessary logic transitions. This means that when the inputs to a logical block change the logic components within this block either remain static, or alter their state only once. Consider the logical arrangement shown in Figure 20 a). This circuit, when stimulated by certain input stimuli, contains unnecessary transitions.



a) Unbalanced logical circuit b) Balanced logical circuit

Figure 20: Delay Balancing Logical Circuitry

The transitions, which occur within circuit a) in Figure 20, are shown in Table 16 below. The circuit's inputs start begin as A = '1', B = '0' and C = '1', as shown in the first row of the table. This provides a '1' at the output of the logical block. When the inputs B and C change, as given by the next row, the output from the OR gate changes to a '0' before moving to the correct output of '1'. This is due the propagation time it takes for the change of B to reach the internal signal D. Once this occurs, the output from the OR gate reverts back to the '1' state, thus the system has provided unnecessary transitions.

A	B	C	D	Out
1	0	1	0	1
1	1	0	0	0
1	1	0	1	1

Table 16: Transition Table for Example Unbalanced Logic Circuit

Once delay balancing has been applied to circuit a), a configuration similar to that of circuit b) in Figure 20 is produced. The AND gate, producing the internal signal E, delays the second input to the OR gate so that both inputs, D and E arrive at very similar times. The AND gate could be replaced with another method of delay, such as a buffer, or an OR gate. The method of delay balancing a logical circuit to reduce its power consumption only applies to synchronous circuits, where inputs to logical blocks are to change at the same time.

When the same stimulus, as applied to the unbalanced circuit, is applied to the balanced circuit, the transitions given in Table 17 are observed. From this table, it can be seen that the inputs to the OR gate, D and E change their level at the same time, therefore stopping the output from falling to '0' before rising back to '1'.

A	B	C	D	E	Out
1	0	1	0	1	1
1	1	0	0	1	1
1	1	0	1	0	1

Table 17: Transition Table for Example Balanced Logic Circuit

Analysing the number of transitions within these example circuits shows that, for the stimuli applied, both circuits contain two 0-1 transitions, therefore not reducing the power consumption of this circuit. The power reduction becomes apparent in the external circuitry of which the output, Out, is an input. For circuit a) a 1-0-1 pulse is seen. This may propagate through the external circuitry consuming excess power.

Circuit b) stops the propagation of this pulse. The real saving of power using this method becomes more apparent in complex circuits.

The delay balancing applied to the multiplier happens in the MCells themselves. The aim is that all the inputs to each MCell arrive at the same time, so that the switching activity of the block is reduced.

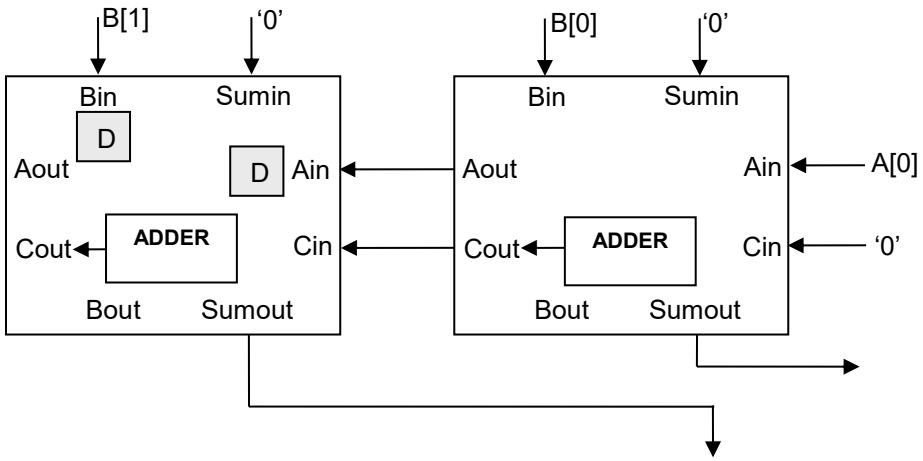


Figure 21: Delay Balancing the Parallel Array Multiplier

Figure 21 shows two MCells, performing the least significant operations in the 8-bit Parallel Array Multiplier, to illustrate the Delay Balancing process. The aim is that all inputs to an MCell arrive simultaneously to reduce transitions in the system. It can be seen that the Cout signal from the first MCell is an input to the second MCell and will not be valid until it is generated by the adder in the first MCell. The grey blocks in the second MCell are delay blocks which delay the input signal to the block by the length of one addition operation. Therefore it can be seen that the Bin and Ain signals will be delayed by 1 addition time and arrive at the same time as the Cin signal (Cout from the previous MCell), thus reducing transitions. When this technique is applied over the entire multiplier, a large power saving is observed.

The delays applied to each MCell are shown in the below matrix diagram (Figure 22). The first number in each cell is the delay applied to the Bin signal, and the second is the delay applied to the Ain signal (e.g. 7_1 means the Bin signal is delayed by 7 additions and the Ain signal by 1). In the case of the MCells in the left hand column, the Sumin signal is also delayed by 1 addition to reduce transitions. The diagram refers to the positions of the MCells in the structure, and can be compared with the diagram in Appendix 7.

7_1	6_1	5_1	4_1	3_1	2_1	1_1	0_0
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_2
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_4
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_6
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_8
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_10
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_12
2_1_1	2_1	2_1	2_1	2_1	2_1	2_1	2_14

Figure 22: Delay balancing applied to MCells

The delay balanced multiplier was subjected to an identical test setup as the unbalanced multiplier to assess the reduction in power as a result of adopting the technique. The results are shown in Table 18.

	Transitions after 1ms	Transitions after 5ms	Transition after 10ms
Delay Balanced 8-bit Parallel Array Multiplier	**67,870** (35.1% less than Unbalanced Multiplier)	**374,372** (35.1% less than Unbalanced Multiplier)	**736,268** (34.5% less than Unbalanced Multiplier)

Table 18: Delay Balanced 8-bit Parallel Array Multiplier results

The results show that a power saving of around 35% was observed as a result of adopting the power reduction technique. This is consistent with research results quoting the power saving as 36% [3]. The delay balanced multiplier will be used in the FIR filter design to complete the 8-bit multiplications.

5 Conclusion

This report has described the analysis performed to select adders for use in the 8-bit Parallel Array Multiplier and 16-bit summing adders to be constructed in the FIR Filter project. 1-bit ripple adders were used to construct an 8-bit Parallel Array Multiplier, which was analysed for power consumption. Delay Balancing was then applied to the Array Multiplier to reduce the power consumption due to switching by 35%. Four 4-bit Carry Look-ahead Adders were rippled and delay balanced to provide a low-power fast adder for use in the system to add 16-bit numbers. A Sign-magnitude number representation was also selected for the FIR coefficients to be stored in the system.

6 References

1. **"EZ431VLSI Design Project: Lecture notes" –** Bashir Al-Hashimi, University of Southampton, England
2. **"Digital System Design with VHDL" –** Mark Zwolinski, *Prentice Hall, ISBN0-201-36063-2*
3. **"Delay balanced multipliers for low-power DSP core" –** T. Sakura, Digest of Technical Papers, *IEEE symposium on Low Power Electronics, pp36-37, 1995.*

Appendix 1: Typical OrCAD test setup

This Appendix contains a schematic of a typical OrCAD transition monitoring setup overleaf. The schematic shows the setup for a 4-bit Carry Look-ahead Adder.

Appendix 2: Detailed results of OrCAD simulation

This Appendix shows the detailed results of the OrCAD power analysis simulations

1-Bit Adders

The table below shows the patters applied to the 1-bit adders.

Ripple Adder			CLA Adder		
A	**B**	**Cin**	**A**	**B**	**Cin**
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	0
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	1

These are the numbers of transitions on the outputs of gates in the adders.

Ripple Adder Total 0-1 gate transitions: 10
CLA Adder Total 0-1 gate transitions: 7

4-Bit Adders

This table shows the inputs applied to the 4-bit adders. X* is a sequence defined in the table. Where it appears in the tables indicates that the other input was held stable and the X* pattern was applied to the other input. Where no number appears in the transitions column, a persistent error prevented the simulation from running.

X*	Ripple Adder			CLA Adder		
0000	**A**	**B**	**No. of Transitions**	**A**	**B**	**No. of Transitions**
0001	0000	X*	43	0000	X*	49
0011	0001	X*	30	0001	X*	36
0010	0010	X*	-	0010	X*	-
0110	0011	X*	21	0011	X*	33
0111	0100	X*	-	0100	X*	-
0101	0101	X*	-	0101	X*	-
0100	0110	X*	-	0110	X*	-
1100	0111	X*	18	0111	X*	20
1110	1000	X*	45	1000	X*	52
1111	1001	X*	44	1001	X*	42
1011	1010	X*	-	1010	X*	-
1010	1011	X*	22	1011	X*	34
1000	1100	X*	-	1100	X*	-
1001	1101	X*	-	1101	X*	-
0001	1110	X*	-	1110	X*	-
	1111	X*	16	1111	X*	16

Ripple Adder Total 0-1 gate transitions: 239
CLA Adder Total 0-1 gate transitions: 282

8-Bit Ripple Adder

Ripple Adder			CLA Adder		
A	**B**	**No. of Transitions**	**A**	**B**	**No. of Transitions**
00000000	X*	16	00000000	X*	34
00000001	X*	39	00000001	X*	78
00000011	X*	40	00000011	X*	52
00000111	X*	43	00000111	X*	-
00001111	X*	42	00001111	X*	-
00011111	X*	42	00011111	X*	-
00111111	X*	44	00111111	X*	-
01111111	X*	46	01111111	X*	-
11111111	X*	44	11111111	X*	-
11111110	X*	40	11111110	X*	-
11111100	X*	36	11111100	X*	-
11111000	X*	32	11111000	X*	-
11110000	X*	28	11110000	X*	-
11100000	X*	24	11100000	X*	-
11000000	X*	20	11000000	X*	25
10000000	X*	16	10000000	X*	20

X*
00000000
00000001
00000011
00000111
00001111
00011111
00111111
01111111
11111111
11111110
11111100
11111000
11110000
11100000
11000000
10000000

Ripple Adder Total 0-1 gate transitions: 131
CLA Adder Total 0-1 gate transitions: 209
(NB: totals only for where the test worked for both adders)

Appendix 3: Simulation of VHDL implemented Transition Counter

Overleaf is a simulation waveform output plot showing the operation of a transition counter in the VHDL testing suite. The Counter “Count0” is incremented on the positive edge (0-1 transition) of the signal Y0.

Appendix 4: Adder Power Testing in VHDL

The following waveform plots show the results of the power analysis performed on the 16-bit ripple adder after 1ms and 5ms.

The results are as follows:
1ms -> 5844 + 4061 = 9905
5ms -> 36172 + 26705 = 62877

as shown in Table 11 of the report.

Appendix 5: Functional testing of Multiplier in VHDL

The following simulation waveform output shows the functional testing of the 8-bit multiplier, displaying correct operation.

Appendix 6: Multiplier Power Testing in VHDL

The following three waveform plots show the power analysis results for the 8-bit unbalanced 8-bit Parallel Array Multiplier.

The results are as follows:
1ms -> 104,506 transitions
5ms -> 576,104 transitions
10ms -> 1,103,731 transitions

as shown in Table 15 of the report.

Appendix 7: Parallel Array Multiplier Structure

The following page shows the structure of an 8-bit Parallel Array Multiplier as constructed in OrCAD.