1986

The Gate Array Implementation of an Area Calculation Pipeline

Edward J. Janssen
University of Central Florida

THE GATE ARRAY IMPLEMENTATION OF AN AREA CALCULATION PIPELINE

BY

EDWARD JOSEPH JANSSEN
B.S.E.E., University of Florida, 1980

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the Master of Science degree in Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1986

ABSTRACT

A gate array is a semi-custom designed integrated circuit. The integrated circuit is designed by a customer and then turned over to a vendor to be manufactured. A single gate array is capable of replacing a full board or more of SSI and MSI components.

An area calculation path of a special purpose computer was designed into a gate array. LSI Logic Corporation was used as the vendor. The gate array was designed and then simulated with the Tegas Description Language. The simulation revealed a worst case timing problem which was corrected by adding an additional stage in the pipeline. The additional stage increased the time a first result is available at the output of the pipeline, but did not effect the rate at which successive results are available. The simulation and actual gate array prototype were proven with a calculated set of test vectors.

The benefit of using gate arrays comes from reduced costs and increased reliability.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Gate arrays have been around since the mid 60's. But because of a lack of standardization in design, simulation, and test in those years, gate array usage did not become widespread until recent years (Hartmann 1980).

A gate array is a semi-custom integrated circuit. Wafers are produced in large quantities as a cellular arrangement of logic elements. At this point, the gate array is 70% complete. It is the final metalizing state of the gate array which creates the semi-custom design. Gate arrays are available with densities from several hundred to 10,000 gates; a gate being most commonly defined as the equivalent of a two-input NAND or NOR gate (Hartmann 1980).

The high density capabilities on a single chip enable a gate array to replace a full board of standard family components. As a result, gate array usage can result in cost savings by lowering both inventory and spare requirements. Another positive aspect of gate arrays is the increase in reliability by reducing component quantity (Pitts 1981).

But on the negative side, the semi-custom design aspect of gate arrays result in large initial non-recurring engineering (NRE) costs. In order for the use of a gate

array to be practical from a cost standpoint, the customer must plan on purchasing a large enough quantity of gate arrays to offset these NRE costs. Depending upon the application, gate arrays may begin showing savings after 1,000 pieces (Pitts 1981).

A boom in the gate array industry took place in 1978 when IBM used gate array technology in their 4300 series computers. This use along with others proved gate arrays to be practical from a reliability standpoint.

With proven reliability and the potential of cost savings, gate array use has increased. Estimates show that gate array use was 1% of the bipolar digital market in 1980 and 4% in 1984 (Hartmann 1980).

In an effort to stay on top of advancing technology and reduce production costs, gate arrays have become an important part of present day logic systems. One such system, a special purpose computer (SPC), which had previously been designed with TTL technology, was investigated for possible gate array applications. The number of these special purpose computers to be produced was estimated at 300. Therefore any gate array application which was to show a savings would have to be one of multiple use.

A calculation pipeline which is used eight times per system was found to be a good candidate for the gate array application. The purpose of this report is to describe the design, simulation, testing, and benefits of a pipeline algorithm gate array using the SPC calculation pipeline as an example.

CHAPTER I

THE DESIGN

Choosing a Vendor

The first step in the design process was to choose a vendor. This step was important before beginning the design, because most vendors now have their own logic libraries of predesigned functional elements, usually called macrocells or macrofunctions, which are used as building blocks to create a design. Switching between two vendors' libraries may not be an easy task.

Three vendors were considered; General Electric, Motorola, and LSI Logic Corporation (LSI). The first important point considered was that of education in gate array design. All three vendors offered gate array design courses to educate the first time gate array designer. When the gate array pipeline design began, General Electric did not source a gate array containing enough gates to meet the expected density of the design, and was ruled out. Motorola and LSI both had gate arrays with enough density; so their macrocell libraries were considered next.

LSI Logic Corporation has an extensive library of functional elements which closely resemble the TTL 7400 series family and LSI has an impressive success rate of

first time working designs. Motorola's library did not resemble the 7400 series as closely as LSI. Thus, LSI Logic Corporation was chosen as the vendor.

The LSI Logic Corporation gate array used, was the LL5240. The LL5240 is a three micron CMOS chip with a maximum of 4,200 usable gates. The LL5240 has TTL compatible input and output buffers which this design required (LSI 1983).

Definition of a Pipeline

A pipeline can be described as an array or string of registers that contain arguments for which computations are in various stages of completion (Hill 1978). A pipeline is formed by using pipeline registers to split a system into smaller functional sections. This can result in increased system speed. For example consider the fictional system shown in Figure 1.

The total time (T_{total}) through this fictional system is 100 nanoseconds (ns) from "data in" to "data out." Because of the different time required by each function, no new data may be input until the present set of data has been completely processed. (i.e., the input operation requires 10ns; but if a new set of inputs were introduced at the end of the 10ns, the first set of data being processed by function 1, which takes 30ns, would be destroyed before

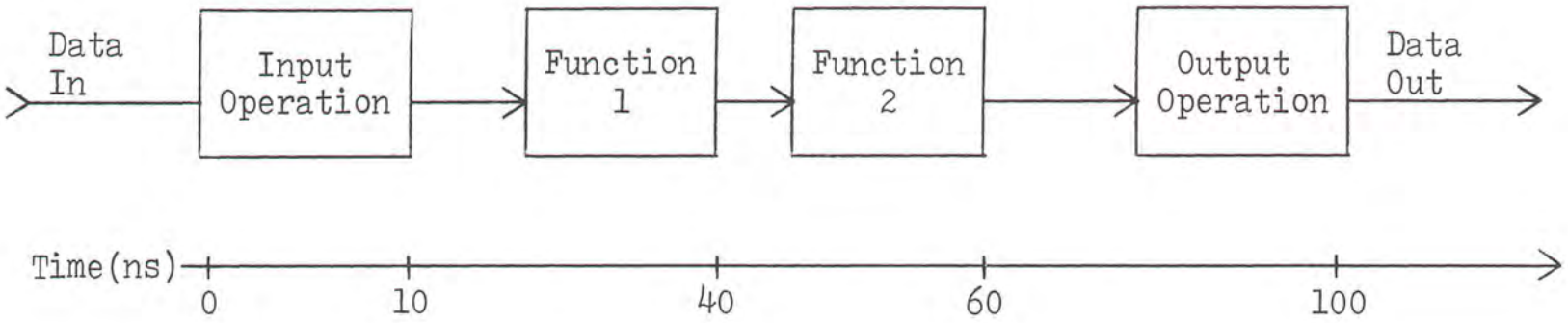

Figure 1. Fictional System

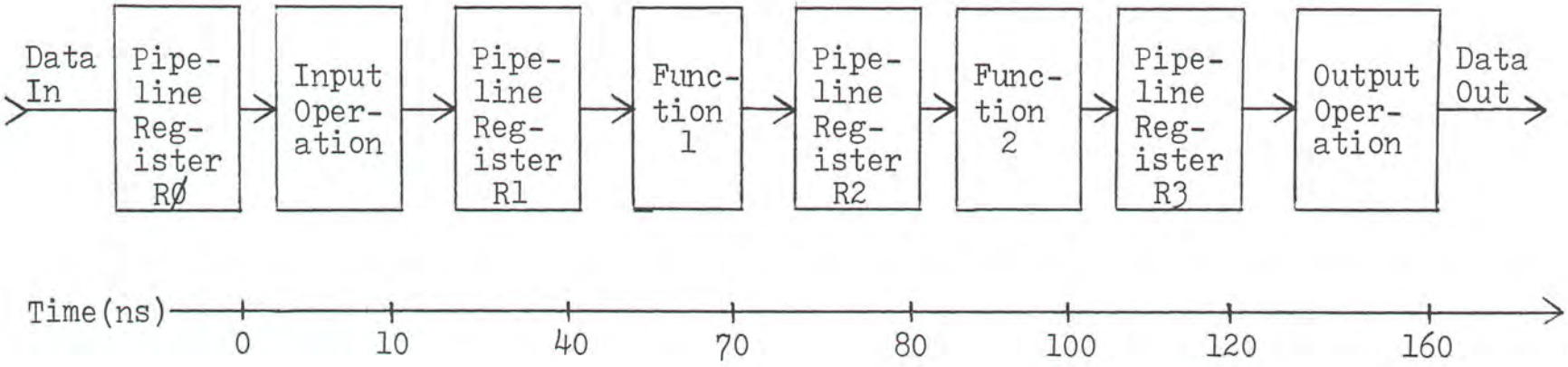

Figure 2. Fictional System With Pipeline Registers

its processing had finished). Therefore, "data out" is only valid every 100ns. Now consider the fictional system of Figure 2, which is the same system as shown in Figure 1, but to which pipeline registers have been added. As can be seen in Figure 2, the pipeline registers act as buffers between the functional areas; therefore, if data is input every clock time (40ns), at the end of each clock time valid data is present at the output of each functional area, or stage. Since the slowest function (T_{slow}) of this system is the output operation, the shortest system clock period allowed is 40ns. If data is input every 40ns, the first set of data through the system takes 160ns, but every subsequent set of data will be valid every 40ns. The effective speedup, T_{total}/T_{slow}, is four. This is shown graphically in Figure 3 (Siewiorek 1982).

For the fictional system described, the pipeline system clock rate was determined by the slowest functional area. Pipelines may also be designed to fit required clock rates. The clock rate of the area calculation pipeline will be 8.33MHz or a period of 120ns. Since this pipeline has already been designed with TTL logic, the required position of pipeline registers is known.

The Area Calculation Pipeline to be Implemented

Each clock time, the SPC calculated, in parallel, four sequential sets of data. One area calculation pipeline (ACP)

Functional Area

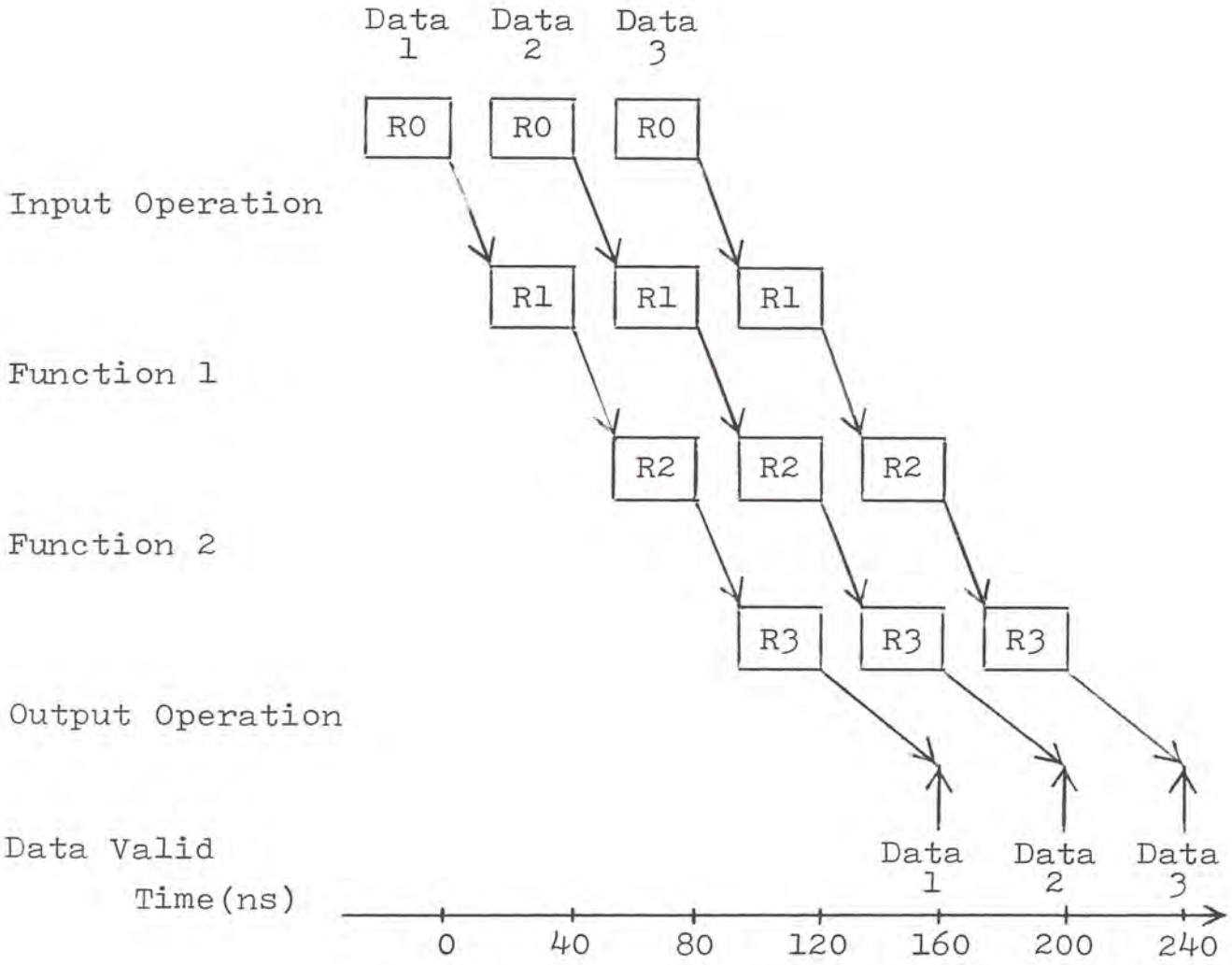

Figure 3. Time Function Diagram of the Fictional System.

is part of the overall calculation pipeline for each set of data. The sets of data are accumulated for 250 clock times so that a total of 1000 sequential sets of data are calculated. The 1000 accumulated sets then become a 1000 pixel raster line for a video display.

The purpose of the ACP is to calculate the values of three areas, A, B, and C, from preliminary information about areas A and B. The ACP is primarily a series of multiplexing and addition operations based upon control inputs. The control inputs to each ACP depend upon which of the four sets of data is being calculated by that ACP. The determination of these control inputs occur before the ACP and will not be discussed in this report. The calculation is done in two steps. The ACP is therefore broken into two sections as shown in Figure 4. The first step, or section, is the parallel calculation of the intermediate values AI and BI of areas A and B. The second section calculates the final values of A, B, and C. The symbols for the inputs to the ACP are described by Table 1, and the symbols for the outputs are described by Table 2.

Pipeline Section 1

As was previously stated, section 1 performs the parallel calculation of AI and BI. These two calculations are functionally identical; so for the present, only the calculation of AI will be discussed. The calculation of AI can

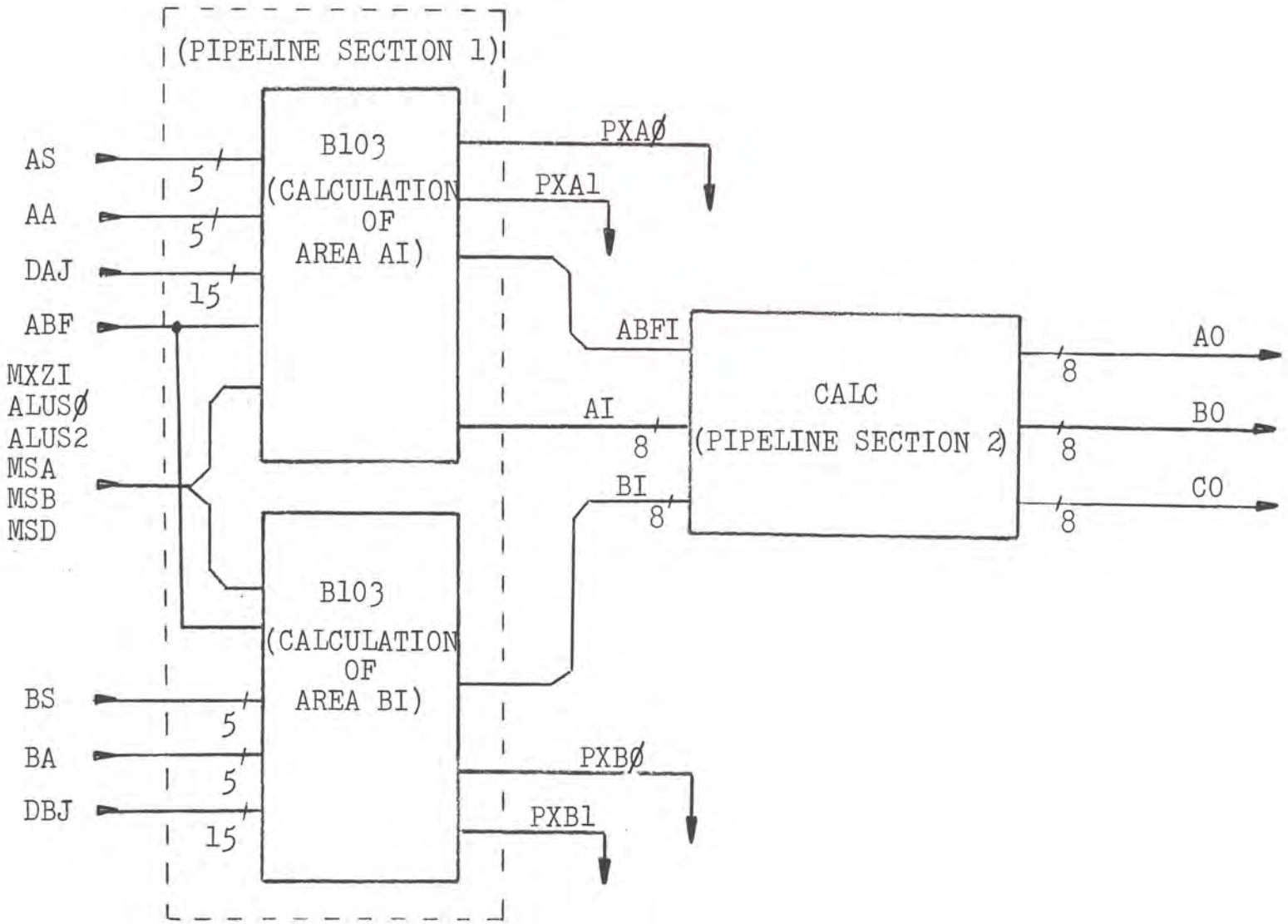

Figure 4. Block Diagram of the Area Calculation Pipeline (ACP).

TABLE 1

CALCULATION PIPELINE INPUTS

INPUT	DESCRIPTION
AS	First possible initial value of area A
AA	Second possible initial value of area A
DAJ	Incremental change in area A
BS	First possible initial value of area B
BA	Second possible initial value of area B
DBJ	Incremental change in area B
ABF	Area priority flag
ALUSØ	Control used to calculate multiples of DAJ and DBJ
ALUS2	Control used to calculate multiples of DAJ and DBJ
MSA	Mux control used between AS, a new AA, an old AA, or the last value of area A
MSB	Mux control used between AS, a new AA, an old AA, or the last value of area A
MSD	Mux control used to select old or new value of DAJ and DBJ

TABLE 2

CALCULATION PIPELINE OUTPUTS

OUTPUT	DESCRIPTION
A0	The resulting area of A
B0	The resulting area of B
C0	The resulting area of C
PXA0	Set to one if A0 is zero, otherwise set to zero
PXA1	Set to one if A0 is equal to $.FF_{16}$, otherwise set to zero
PXB0	Set to one if B0 is zero, otherwise set to zero
PXB1	Set to one if B0 is equal to $.FF_{16}$, otherwise set to zero

be broken into four functional areas as shown in Figure 5. AHPL will be used to describe these functions (Hill 1978).

The first function, AREAMUX, is basically a four-to-one mux. One of the four values, AS, AA, registered AA (AAR), or SA, is selected based on the values of control inputs MSAR and MSBR, as shown in Table 3. An AHPL description of AREAMUX is given in Figure 6.

The second functional area, DELTA, selects a value, DS, from a two-to-one mux choosing between the values of DAJ and registered DAJ (DAJR). This selection is based on control input MSDR. DELTA also registers some control signals used throughout the ACP. An AHPL description of DELTA is shown in Figure 7.

The third functional area is NDELTA. NDELTA first calculates multiples of 1, 2, 3, and 4 times the value selected by DELTA and then performs a four-to-one mux operation of these multiples based on the values of control inputs ALUSØR and ALUS2R. The selection of the multiples depends upon which one of the four sequential data sets the particular ACP is calculating. An AHPL description of NDELTA is given in Figure 8.

The final functional area of the first pipeline section is NAREA. NAREA performs a summation of the outputs of

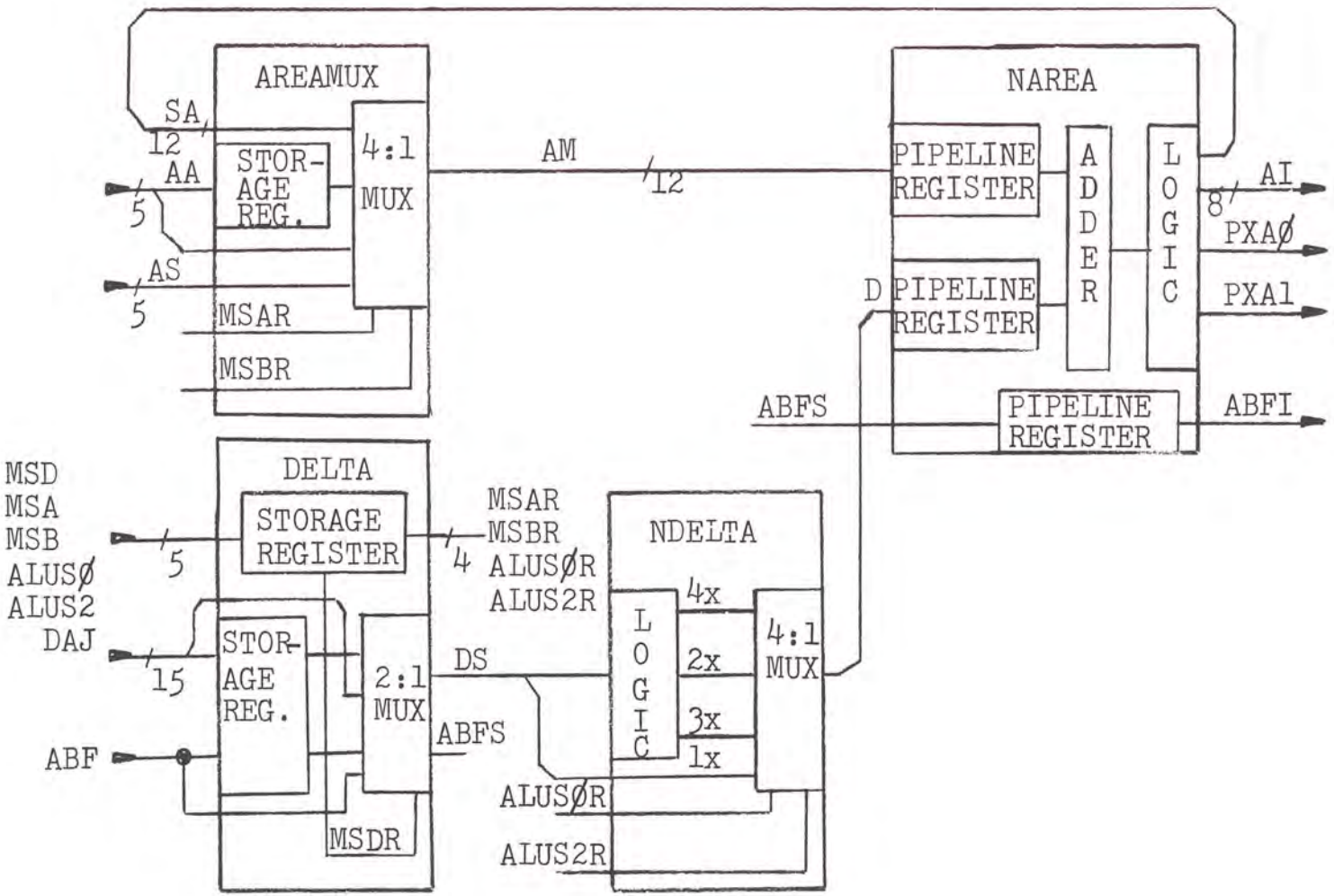

Figure 5. Block Diagram of Area AI Calculation.

TABLE 3

SELECTION OF AREA VALUES IN THE AREAMUX SUB-MODULE

INPUTS		AREA SELECTED
MSAR	MSBR	
0	0	SA
0	1	AA Registered
1	0	AA
1	1	AS

```
MODULE:   AREAMUX
          MEMORY:   AR [5]
          INPUTS:   AS [5] ; AA [5] ; SA [12] ;
                    MSAR: MSBR
          OUTPUTS:   AM [8]

1.   AR←—AA;
     AM = (SA!(AR,7TØ)!(AA,7TØ)!(AS,7TØ))*DCD(MSAR,MSBR)

END SEQUENCE
END
```

Figure 6. AHPL Description of AREAMUX.

```
MODULE:  DELTA
         MEMORY:  MSAR; MSBR; ALUSØR; ALUS2R; MSDR;
                  DAJR[15] ; ABFR
         INPUTS:  MSD; MSA; MSB; ALUSØ; ALUS2; DAJ[15];
                  ABF
         OUTPUTS:  MSAR; MSBR; ALUSØR; ALUS2R; ABFS; DS[15]

1.  MSDR ← MSD; MSAR ← MSA; MSBR ← MSB; ALUSØR ← ALUSØ;
    ALUS2R ← ALUS2; ABFR ← ABF; DAJR ← DAJ;
    ABFS = (ABFR!ABF)*(~MSDR, MSDR); DS = (DAJR!DAJ) *
    (~MSDR, MSDR)

END SEQUENCE
END
```

Figure 7. AHPL Description of DELTA.

```
MODULE:  NDELTA
         INPUTS:  ALUSØR; ALUS2R; DS [15]
         OUTPUTS:  D [15]

1.  D = (DS!(DS1:14,0)!(ADD(DS;DS1:14,0))!(DS2:14,2T0))*
    DCD(ALUSØR,ALUS2)

END SEQUENCE
END
```

Comment: The values of DS times two and four are created by shifting DS one and two times respectively. The value of three times DS is created by adding DS to two times DS.

Figure 8. AHPL Description of NDELTA.

AREAMUX and NDELTA. If the resulting sum is negative, the sum is clamped to zero. If the sum is greater than one, it will be clamped to an eight bit binary fractional representation of one ($.11111111_2$); if between zero and a fractional one value, the sum will be unchanged. Two other outputs of NAREA are PXAØ and PXA1. PXAØ is set to one if the sum is zero otherwise it is clamped to zero. PXA1 is set to one if the sum is a fractional one. An AHPL description of NAREA is shown in Figure 9.

A complete AHPL description of the calculation of AI is given in Figure 10. One pipeline register is required by pipeline section 1.

Pipeline Section 2

AI, BI, and ABFI are the inputs to pipeline section 2, CALC. CALC determines the values of areas A, B, and C. The sum of A, B, and C is required to equal an eight-bit binary fractional one ($.11111111_2$ or $.FF_{16}$). This is because the three areas will be used to determine the color of the pixels of the 1000 raster line. Each pixel has a unity value. The values of A, B, and C are calculated based on AI, BI, and the state of the control flag, ABFI, as shown in Table 4. The block diagram of CALC in Figure 11 shows the use of one pipeline register stage. The AHPL code which describes CALC is shown in Figure 12. A time function diagram of the ACP is shown in Figure 13. Notice in Figure 13, that there

```
MODULE:  NAREA
         MEMORY:  AMR[12] ; DR[15] ; ABFSR
         INPUTS:  AM[12] ; D[15] ; ABFS
         OUTPUTS:  AI[8]; PXAØ; PXA1; ABFI;
         BUS:  SA[12] ;

1.  AMR ←AM; DR ←D; ABFSR ←ABFS;
    SA = ADD(AMR;DR0:11); AI = SA0:7; PXA1 = ∧/AI;
    PXAØ = ~∨/AI; ABFI = ABFSR

END SEQUENCE
END
```

Figure 9. AHPL Description of NAREA.

```
MODULE:  AI CALCULATION
         MEMORY:  AR 5 ; MSAR; MSBR; ALUSØR; ALUS2R; MSDR;
                  DAJR[15] ; ABFR; AMR[12] ; DR[15] ; ABFSR
         INPUTS:  AA[5]; AS[5]; MSD; MSA; MSB; ALUSØ, ALUS2;
                  DAJ[15]; ABF
         OUTPUTS:  AI[8] ; PXAØ; PXA1; ABFI
         BUSES:  AM[12] ; ABFS; DS[15] ; D[15] ; SA[12]

1.  AR←AA; MSDR←MSD; MSAR←MSA; MSBR←MSB;
    ALUSØR←ALUSØ; ALUS2R←ALUS2; ABFR←ABF; DAJR←DAJ;
    AM = (SA!(AR,7TØ)!(AA,7TØ)!(AS,7TØ))*DCD(MSAR,MSBR);
    ABFS = (ABFR!ABF)*(~MSDR,MSDR);
    DS = (DAJR!DAJ)*(~MSDR,MSDR);
    D = (DS!DS1:14,0)!(ADD(DS;DS1:14,0))!(DS2:14,2TØ))*
        DCD(ALUSØR,ALUS2R);
    AMR←AM; DR←D; ABFSR←ABFS;
    SA = ADD(AMR;DR0:11); AI = SA0:7; PXA1 = ∧/AI;
    PXAØ = ~∨/AI; ABFI = ABFSR

END SEQUENCE
END
```

Figure 10. AHPL Description of AI Calculation.

TABLE 4

CONDITIONS GOVERNING VALUES OF AREAS A, B, AND C

CONDITIONS	VALUE OF AREAS
$AI + BI < .FF_{16}$	$A = AI$ $B = BI$ $C = .FF_{16} - (AI + BI)$
$AI + BI \geq .FF_{16}$, $ABFI = 0$	$A = AI$ $B = .FF_{16} - AI$ $C = 0$
$AI + BI \geq .FF_{16}$, $ABFI = 1$	$A = .FF_{16} - BI$ $B = BI$ $C = 0$

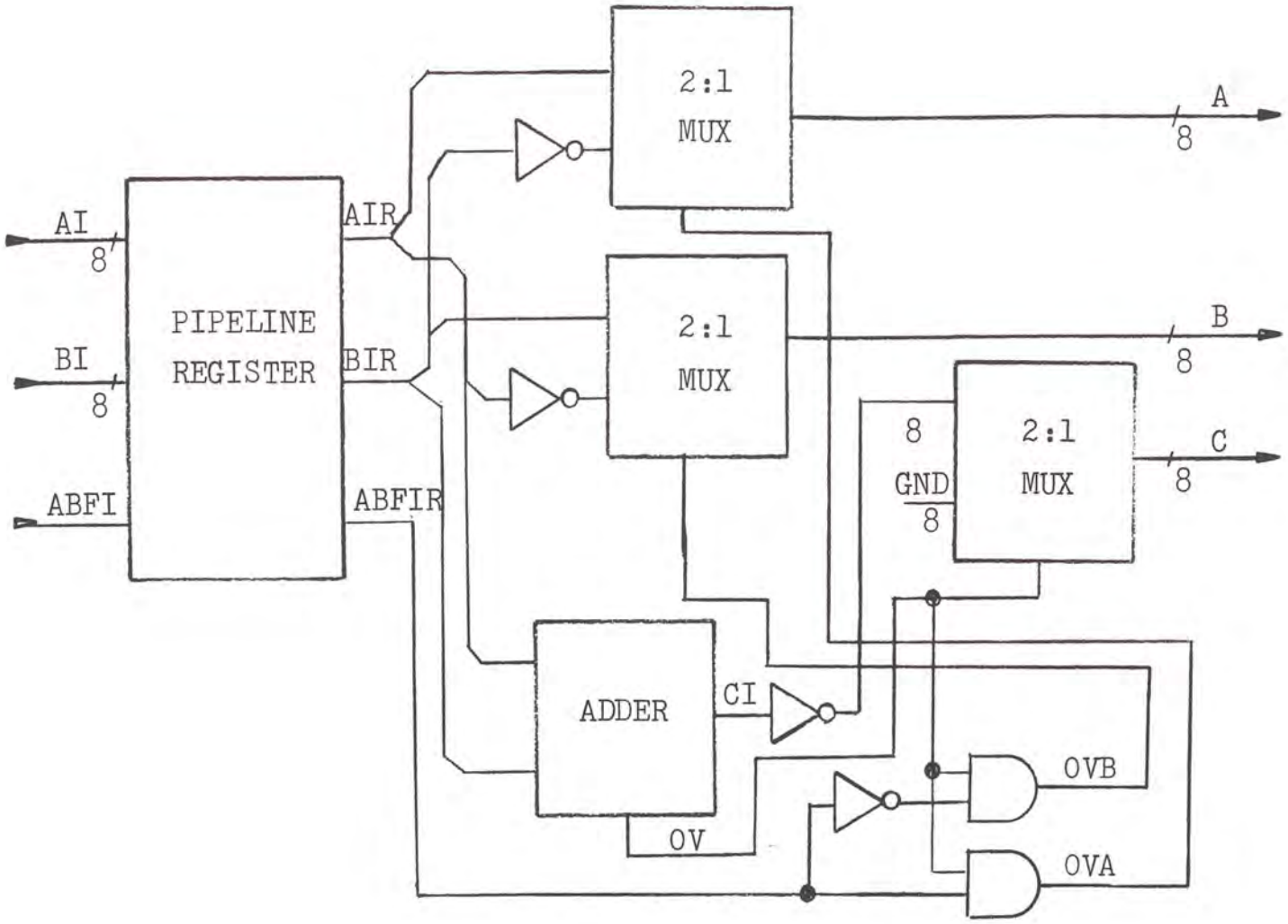

Figure 11. Block Diagram of CALC.

```
MODULE:  CALC
         MEMORY:  AIR[8]; BIR[8]; ABFIR
         INPUTS:  AI[8]; BI[8]; ABFI
         OUTPUTS:  A[8]; B[8]; C[8]
         BUSES:  CI[9]

1.  AIR ←AI; BIR ←BI; ABFIR ← ABFI;
    CI = ADD(Ø,AIR; Ø,BIR); A = (AIR!BIR) * ((ABFIR∧CI0),
         (ABFIR∧CI0));
    B = (BIR!AIR) * ((ABFIR∧CI0), (ABFIR∨CI0));
    C = (CI1:8!8TØ) * (CI0,CI0)

END SEQUENCE
END
```

Figure 12. AHPL Description of CALC.

Functional Area

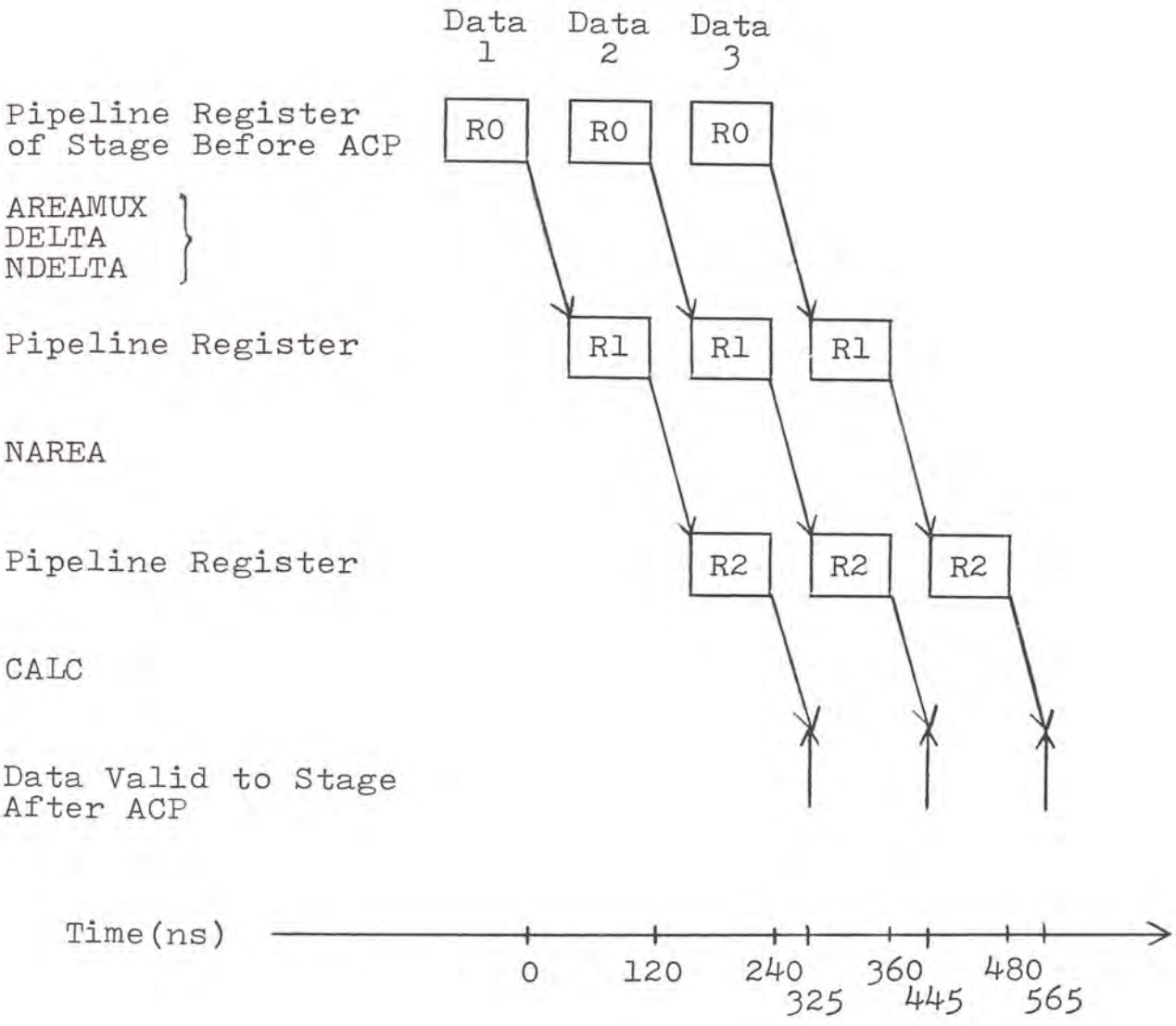

Figure 13. Time Function Diagram of the ACP.

is an estimated 85ns between the output of the pipeline register of CALC to the time the data is valid for the stage following ACP. This time was estimated using TTL technology. This is the estimated time to complete CALC function. This time is also part of the 120ns set-up time for the following stage.

Layout of the Gate Array Design

The first step in laying out a gate array is to break the design into small functional areas or modules. This step becomes important when testing the gate array. It is much easier to debug a gate array test simulation if the overall simulation is composed of smaller functional simulations which can be tested independently. The ACP was first broken into the two major pipeline sections previously discussed. Section 1 was broken into two modules, called B103, to calculate AI and BI. The modules which calculate AI and BI are functionally identical. The only difference is that ABFI is not used in the BI calculation module. B103 was further broken down into the functional sub-modules; AREAMUX, DELTA, NDELTA, and NAREA. Section 2 was considered basic enough and not broken into sub-modules. The design breakdown is shown in Figure 14.

The next gate array design step is to design each sub-module with macrocells. As stated before, LSI was used as

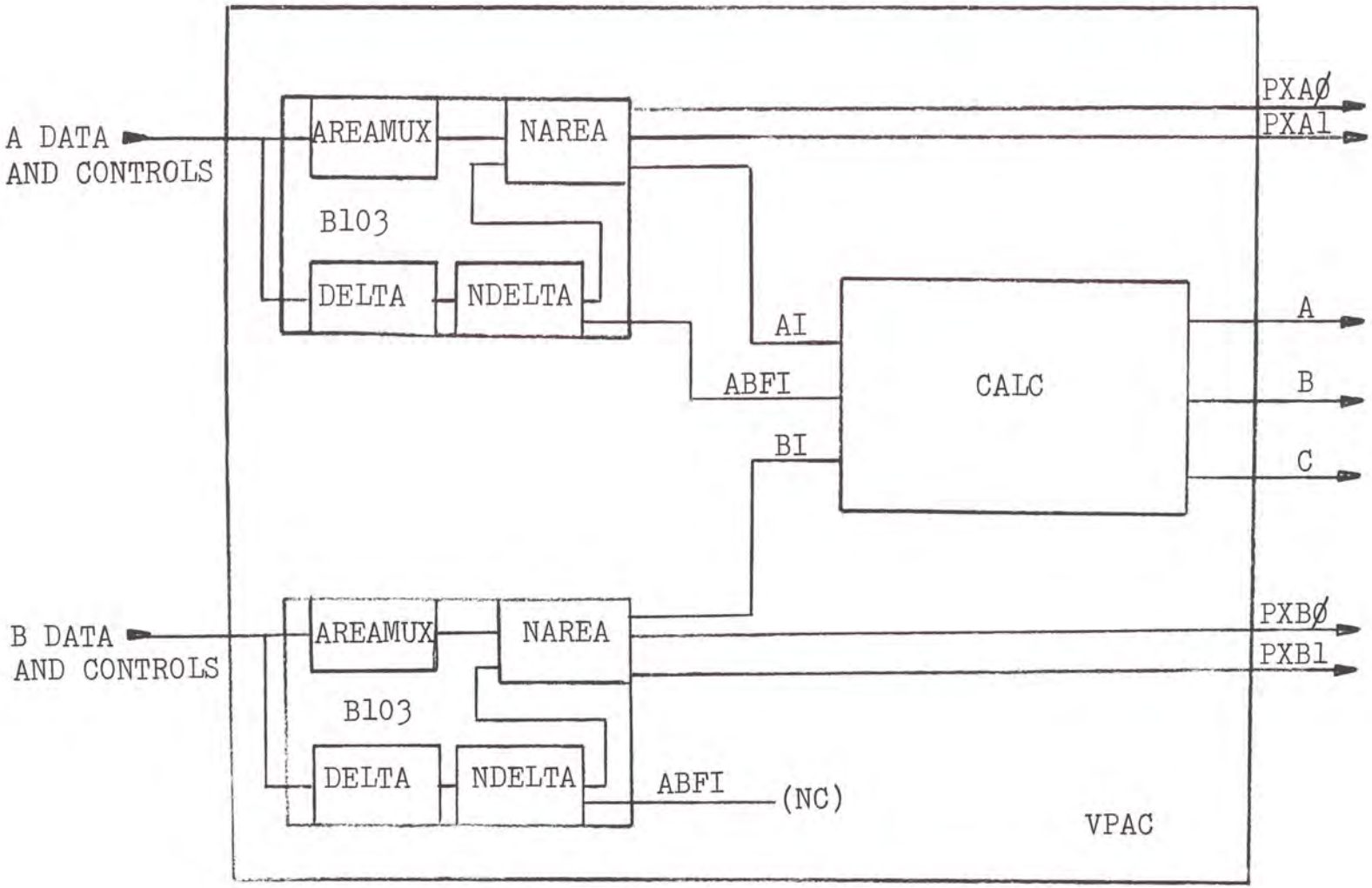

Figure 14. Design Breakdown of ACP Into Sub-modules.

the vendor. LSI's macrocell library, which closely resembles the TTL 7400 series logic family, is described by their CMOS Macrocell Manual (LSI 1983).

CHAPTER II

DESIGN SIMULATION

Under the LDS system of LSI Logic Corporation, the simulation of a gate array begins by describing the array using the Tegas Description Language (TDL). The design is then verified by a Design Verification Program before testing begins.

Simulation Using TDL

Each module of the design must be described with a Network Description File (NET File). A skeleton example of the NET File is shown in Figure 15. The NET File describes the module in terms of inputs, outputs, bidirectional signals, wireand signals, a brief module description, and network interconnection description. The input, output, bidirect and wireand sections are just listings of the signals of each type. The network interconnection description is accomplished by using a TDL statement to describe each macrocell which makes up the module. The basic macrocell description statement is

```
Macrocell Name = Macrocell Type
                 (list of inputs)$  (LSI 1983).
```

The macrocell name is determined by the designer.

```
COMPILE                          $
DIRECTORY      MASTER            $
OPTIONS        REPLACE, XREF     $

MODULE         "1: -- ENTER MODULE NAME WITH USER ID -------"
$

INPUTS         "2: -- ENTER INPUTS, SEPARATED BY COMMAS ----"
$

OUTPUTS        "3: -- ENTER OUTPUTS, SEPARATED BY COMMAS ---"
$

BIDIRECT       "4: -- ENTER BIDIRECTS, SEPARATED BY COMMAS -"
$

DESCRIPTION    "5: -- ENTER MODULE DESCRIPTION -------------"
$

LEVEL          FUNCTION

USE            "6: -- ENTER MACROCELLS, MACROFUNCTIONS USED -"
,WIREDAND       = WAND (2,1 + TRISTATE, SETZZ)
$

WIRED          USE WIREDAND      $

DEFINE         "7: -- ENTER NETWORK INTERCONNECT -----------"

END            MODULE            $
END            COMPILE           $
END            TDL               $
```

Figure 15. Network Skeleton File.

NET Files are first written for each sub-module, and then using these sub-module NET Files much like sub-routines, NET Files are written for the major modules. The major modules are then pulled together by one final NET File which describes the entire gate array. In the case of the ACP, the final NET File was called VPAC.

The next step is to compile the NET Files. The compiling of the NET Files will check for any syntax or naming errors which might occur in the NET File. Obviously the NET Files for the sub-modules AREAMUX, DELTA, NDELTA, and NAREA must compile correctly before B103 will compile. Likewise, the NET Files for B103 and CALC must compile correctly before the VPAC NET File will compile.

Design Verification

Once the complete NET File design compiles, it will be verified. The verification is accomplished by use of the LDS Design Verification Program. The verification program uses estimated wire lengths to check propagation delay times between macrocells, number of gates used, number of input/output pins used, percentage of wiring which can be accomplished by automatic layout and provides a connection cross-reference list for the gate array. The results of the VPAC verification showed a use of 2,683 gates for design and 588 gates for routing, for a total of 3,271 or 78% of the maximum 4,200 usable gates. LSI Logic Corporation recommends

using no more than 75% to 85% to avoid routing difficulties. VPAC required 60 input, 28 output, 4 grounds, and 4 power (-5Vdc) pins for a total of 96. This meant that the standard 101 pin grid package could be used. The verification output for VPAC is shown in Figure 16.

The use of the estimated propagation delay times pointed out a timing problem. Because CMOS is slower than TTL, the outputs of VPAC were not fast enough. The outputs of the ACP feed several other levels of combinational logic in the special purpose computer before being registered. The slower output of the ACP gate array combined with the propagation delay of the logic following the ACP would not meet the 120ns clock period requirement. This problem was solved by adding another stage in the ACP. A pipeline register was added to the output of CALC. The block diagram changed from that shown in Figure 12 to that of Figure 17. The time diagram of the new ACP is shown in Figure 18 and the new AHPL description is given in Figure 19. The significance of adding this additional pipeline register is that T_{total} is now 360ns instead of 325ns, as shown in Figure 13, and CALC ends with a complete stage instead of in the middle of a stage. This means there is a full clock period set-up time between the output of CALC and the pipeline register of the next stage. The effective speedup of the final configuration is three. T_{slow} is still 120ns.

LDS-II DESIGN VERIFIER NETWORK SUMMARY

Project ID:	LIA0605	LDS Account Name:	L1A0605S
Array Name:	VPAC	Directory Name:	L1A0605S
Array Type:	LSI5420	Array Family:	CMOS5K
Current Date:	03/14/84	Current Time:	09:55:07
Date 'CNET VPAC':	02/22/84	Time 'CNET VPAC':	17:58:00
CMOS5K Library Date:	02/20/84	CMOS5K Library Revision:	2.12

NETWORK STATISTICS BEFORE CELL DELETIONS

Number of Input Pads:	60	Number of Cell Types:	20
Number of Output Pads:	28	Number of Cells Used:	1747
Number of Bidirect. Pads:	0	Number of Gates Used:	3247
Min. No. Power/Ground Pads:	4	Array Usage (Per Cent):	95.98
Total Array Pads Used:	92	Minimum No. VDD Pins:	2
Available Chip Pads:	148	Minimum No. VSS Pins:	2
Available Package Pins:	152	Package Pins Used:	92
Cell Inputs To VDD:	58	Cell Inputs to VSS:	294
Number of Signal Nets:	1679	No. of Unc. Cell Outputs:	258
Average Pins/Net:	3.038	Maximum Pins/Net:	49

NETWORK STATISTICS AFTER CELL DELETIONS

Number of Input Pads:	60	Number of Cell Types:	20
Number of Output Pads:	28	Number of Cells Used:	1413
Number of Bidirect. Pads:	0	Number of Gates Used:	2683
Min. No. Power/Ground Pads:	4	Array Usage (Per Cent):	77.87
Total Array Pads Used:	92	Minimum No. VDD Pins:	2
Available Chip Pads:	148	Minimum No. VSS Pins:	2
Available Package Pins:	152	Package Pins Used:	92
Cell Inputs to VDD:	50	Cell Inputs To VSS:	93
Number of Signal Nets:	1388	No. of Unc. Cell Outputs:	207
Average Pins/Net:	3.087	Maximum Pins/Net:	49

DELAY VALUES ESTIMATED FOR COM'L, WORST CASE (70 DEG.0, VDD-4.75V)

WIREABILITY MEASURE FOR AUTOMATIC LAYOUT: 86.34

Figure 16. Verification Output.

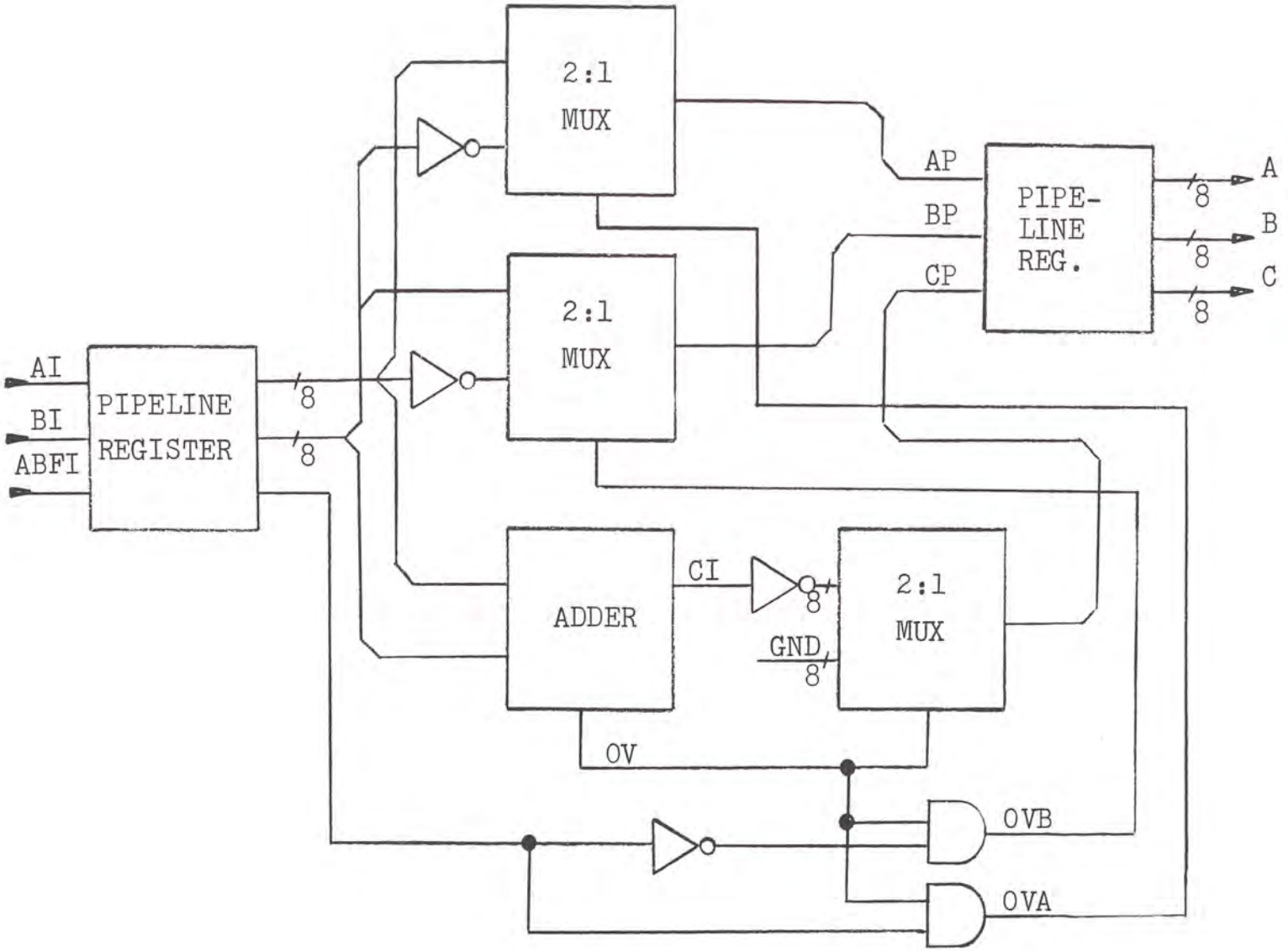

Figure 17. Block Diagram of CALC With Output Pipeline Registers.

Functional Area

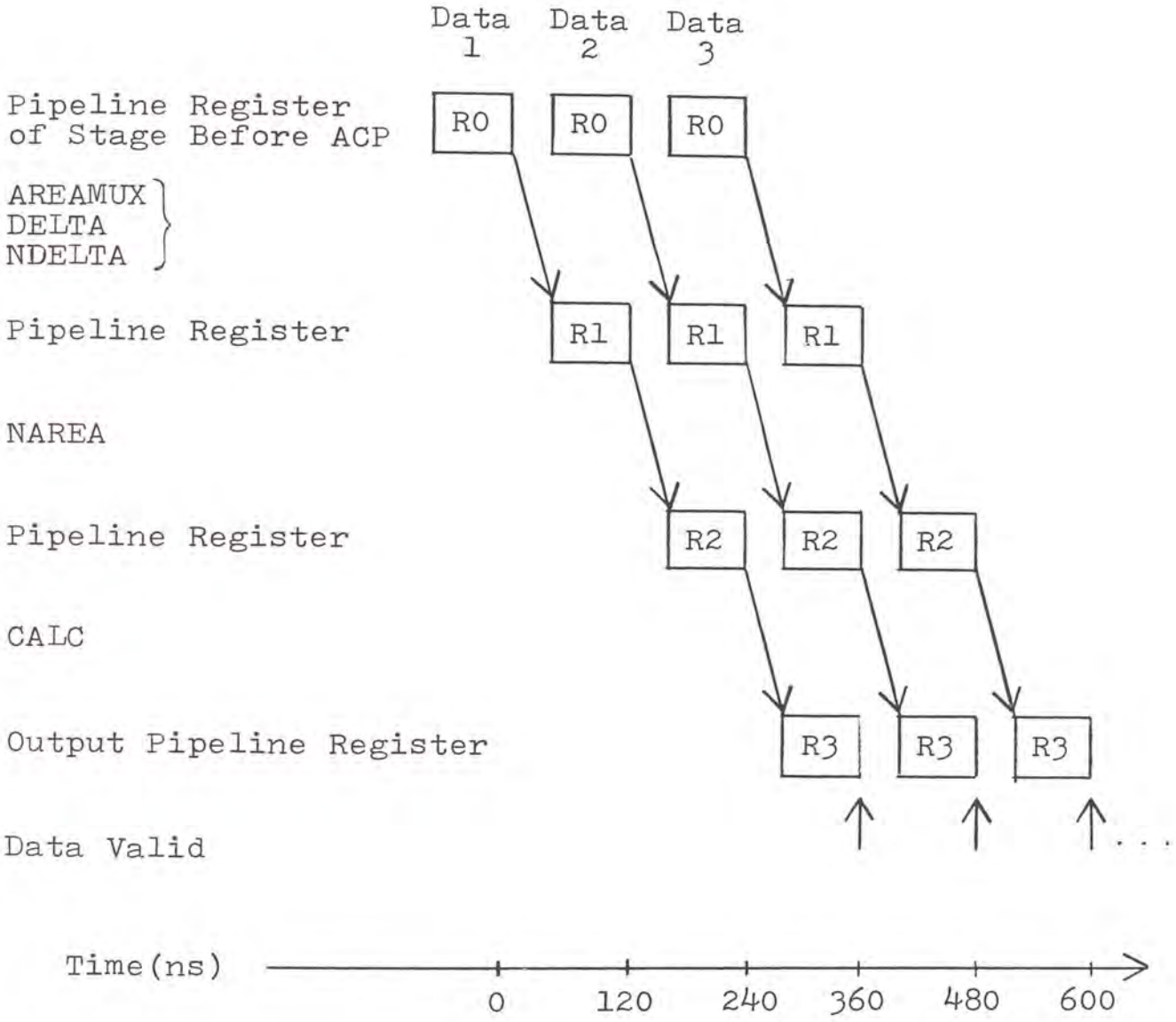

Figure 18. Time Function Diagram of ACP With Modified CALC.

```
MODULE:  CALC With Second Pipeline Register
         MEMORY:  AIR [8] ; BIR [8] ; ABFIR; APR [8] ; BPR [8] ;
                  CPR [8]
         INPUTS:  AI [8] ; BI [8] ; ABFI
         OUTPUTS:  A [8] ; B [8] ; C [8]
         BUSES:  CI [9] ; AP [8] ; BP [8] ; CP [8]

1.   AIR ←AI; BIR ← BI; ABFIR← ABFI;
     CI = ADD(∅,AIR; ∅,BIR);
     AP = (AIR:~BIR) * ((ABFIR∧CI0),(~(ABFIR∧CI0)));
     BP = (BIR:~AIR) * ((~ABFIR∧CI0),(ABFIR∨CI0));
     CP = (~CI1:8:8T∅) * (~CI0, CI0)

     APR ←AP; BPR ← BP; CPR ← CP;
     A = APR; B = BPR; C = CPR

END SEQUENCE
END
```

Figure 19. AHPL Description of CALC With Additional Pipeline Register.

CHAPTER III

TESTING THE GATE ARRAY

No less important than the gate array design is the task of testing the gate array. With the possible gate count of arrays reaching as high as 10,000, the need for built-in diagnostics becomes very important. The question also arises if a set of test vectors should be complete or adequate.

The test of the VPAC gate array as with any gate array was broken into two sections. These two sections are the tests conducted before prototype fabrication and the tests conducted after prototype fabrication.

Testing Before Prototype Fabrication

After correct compilation of the NET File for each sub-module, input and output test vectors were hand-calculated for that module. The input test vectors were then input to a Sentry tester to produce output vectors. The output vectors were compared against those calculated. When it was determined that these were correct, B103, CALC, and finally VPAC were tested in the same manner.

The decision was made to test VPAC with adequate test vectors instead of complete. Adequate vectors test the

gate array functionally but do not test every possible pattern as do complete test vectors. This choice was made because the VPAC design had already been proven with TTL technology.

The input test vectors were the same used to previously test the same design in TTL technology. The input signal SLRT was used to clear the registers of VPAC in order to start in a known state. The input vectors could then be used by the Sentry tester to generate output vectors. The Sentry outputs were then compared against those taken from the TTL implementation.

When outputs from the two implementations matched, AC and DC tests were made. AC tests were made to check propagation delays and timing skews. The DC test yields a list of parameters such as VIL, VIH, IIL, IIH, VOL, VOH, IOZ, and IDD for the inputs and outputs of the gate array (LSI, 1983).

Testing the Gate Array Prototypes

At this point, the design phase is complete. The proper acceptance forms were signed by both customer and vendor. Modifications were made to an existing special purpose computer to prepare for the gate array.

Testing of the prototype gate arrays was done in several ways. The first test was a visual one. Since the out-

put of this gate array is eventually displayed on a video monitor, the video was inspected for any signs that the gate array was not functioning as expected. Several video test patterns, such as gray scales of vertical and horizontal bars, were viewed. No flaws were observed.

In the special purpose computer, there is an input memory used just before the gate arrays. This memory was loaded with the test vectors. The data was then allowed to flow through the gate arrays. Since there are no data snapshot points at the output of the gate arrays, a logic analyzer was used to check the outputs.

The second method of testing was to apply the test vectors used to check the gate array simulation. The outputs of the prototype gate arrays matched completely the outputs which were given by the simulation and special purpose computer before modification.

Next, the inputs and the outputs of the gate array were checked for any noise problems which might be caused by crosstalk and/or reflections. Since the gate array was placed on a printed wire board and its I/O must be transmitted over a wire wrapped backplane; the I/O were subject to signal distortion as would be any other transmission line. It was found that undershoot and noise spikes on the gate array I/O were no greater, and in most cases less than

those observed on the previous TTL design. It was then concluded that no additional noise prevention measures, such as shielding or twisted pair, would be required.

Finally a combination voltage level and propagation delay test was made. The purpose of this test was to measure worst case propagation delay times for a range of voltage levels. As should be expected, the propagation delay times were slower for lower voltages and faster for higher voltages. Typical timing values measured from the rising edge of the input clock to stable outputs of the output registers were in the range of 35 to 40 nanoseconds. These values varied by minus or plus 3 nanoseconds for voltages ranging from 4.3 to 6.0 Vdc. The expected worst case timing from the simulation was 55 nanoseconds at 5.0 Vdc. The lower range of 4.3 Vdc was used, because this is the point at which the video began to show flaws. The upper voltage was limited at 6.0 Vdc to avoid damage to the gate array. The gate array specification states the operating range to be 4.75 to 5.25 Vdc.

After completing all of these tests, the prototype was accepted as a working gate array.

CHAPTER IV

BENEFITS OF USING A GATE ARRAY INSTEAD OF SSI AND MSI DEVICES

As was stated in the Introduction, the use of gate arrays involve high nonrecurring engineering costs which take production orders of around 1,000 pieces to recover from. If it were not for these NRE costs, gate arrays would show an almost immediate cost savings.

One reason for this savings is the reduction of inventory and spare costs. For example, one VPAC integrated circuit (IC) replaces a combination of 114 MSI and SSI devices. The reduction in required storage space alone is substantial. There is also a major savings in assembly cost and time when placing these ICs on printed circuit boards.

The reduction of required printed circuit boards is also a source of cost savings. In the proposed application of VPAC, on a per system basis, 24 boards were replaced by four boards. This meant a savings of 20 printed circuit boards per system. This is significant because the cost of printed circuit boards is a major expense in the total cost of a system (Pitts 1981).

The power consumption of the CMOS gate array is lower than its TTL design counterpart. This results in reduced power supply and cooling requirements. This is also a source of cost savings.

A benefit, which may be hard to think of in terms of cost, is the increase in reliability. Using a gate array instead of MSI and SSI logic reduces the number of interconnections and solder joints, and as a result reduces the chances of incurring rework costs.

Another benefit is the reduction in physical size. Size reduction reduces transportation and storage costs. Size reduction also has the added intangible benefit that smaller is better. Reduced size is also important when there are design space limitations.

The estimated cost savings of the VPAC gate array follows the basic guideline of savings beginning to occur when 1,000 production pieces are ordered.

CHAPTER V

SUMMARY

The first step before design was to choose a vendor. LSI Logic Corporation was chosen because the company had a proven working product. The design of the gate array broke down into two major sections. The first major task was to create the design using macrocells from the LSI Logic Library. The second task was to create a NET simulation file for the gate array. This was done by sectioning the design into modules and first creating NET files for each module using Tegas. Then the modules were used as building blocks to create a NET file for the complete design.

The design was first testing with a set of test vectors using the NET file simulation. The simulation pointed out the necessity for adding an additional pipeline stage. The prototype arrays were then manufactured, tested, and verified.

The benefits of a gate array are many, but the most important benefits are reduction in total production costs and increased reliability.