For:hkim
Printed on:Fri, May 1, 1998 11:04:22

Document:elec422
Last saved on:Wed, Dec 17, 1997 17:47:12

The “Vanilla” Arithmetic and Logic Unit

1. Functional Description

1.1 Overview

The ALU consists of five main parts: control unit, 4 8–bit operand registers, 8–bit parallel adder, 17–bit accumulator/shift register, and a logic unit. ALU will support three 8–bit arithmetic operations (ADD, SUBTRACT, and MULTIPLY) in both positive and negative numbers (2’s complement, MULTIPLY will only support positive numbers). ALU supports three full 8–bit logical operations (NOT, OR, and AND).

Also, ALU will be able to perform two types of shift operations (SHIFT RIGHT with 0 replacement and SHIFT RIGHT with loop around) in 9–bit format. This means that the input is the 8–bit and the actual SHIFTER is in 9–bit. So, in all of the cases, both SHIFT will perform exactly the same kind of SHIFT b/c you cannot do a multiple SHIFTs within one cycle.

1.2 Op–Code Assignment

Due to number of commands it supports, the ALU will have 4–bit op–code, and following is the assignment. Reserved Op–codes will be treated as a NOOP (no operation).

Op–Code	Operation
0000	ADD
0001	SUB
0010	MULT
0011	Reserved (NOOP)
0100	NOT
0101	OR
0110	AND
0111	Reserved (NOOP)
1000	SHFT with 0
1001	SHFT with loop around
1010	Reserved (NOOP)
1011	Reserved (NOOP)
1100	Reserved (NOOP)
1101	Reserved (NOOP)
1110	Reserved (NOOP)
1111	Reserved (NOOP)

1.3 I/O Pin Description

Pin Name	Pin Count	I/O
Op–Code	4	Input
Data–In	8	Input
Data–Out	8	Output
Overflow	1	Output
Data Ready	1	Output
Reset (Restart)	1	Input
DFT (State Bits, DONECNT)	6	Output
Clocks	2	Input

1.4 Control Logic

The control logic will be implemented in FSM. Due to the complexities of the logic, the control section is divided into two sections (a main controller and a counter). Basically, the inputs of the main controller are the CLK1, CLK2, OP–CODE, RESET, M (Multiplier bit for multiply operation) and DONECNT bit (from the counter) The following are control output signals: LDREG1, LDREG2, LDREG3, LDREG4, LDACCUM, LDREGOUT, CLRACCUM, SUBT, INADDER, INPAD, DATARDY, SHIFT, SEL0, SELL, INV, OR, AND, STARTCNT, OUTACCUM, OUTREG2, OUTADDER, OUTLOGIC. DATARDY signal is required b/c all of the operation have different timings (for example, AND operation will take 3 CLK cycles, as opposed to MULT, which can take anywhere from 12 to 20 CLK cycles)

1.5 Adder/Subtractor/Multiplier/Shifter

The Adder is a 8–bit parallel adder with an accumulator, so it could be used as a regular adder and a part of a multiplier. It will implement no complex carry look–ahead algorithms to save on logic complexities. Subtractor basically is the same adder with 2's complement on subtracter. 2's complement is implemented by complementing the subtracter, and feeding 1 in the first bit of the adder carry–in bit. For regular ADD and SUB operations, only bit 15:7 (total of 8 bits) of the 17–bit accumulator/shift register will be used.

The multiplier basically is a shift–adder that will repeat the add and shift / shift only operation for 8 times to implement 8–bit by 8–bit multiply. It will only support positive number multiply for circuit complexity and timing issues.

Shift operation is done by the 17 bit accumulator/shift register, but will only use 15:7 bits.

The logic operations will be done in separate one or two 8–bit registers, depending on how many operands each logic function require.

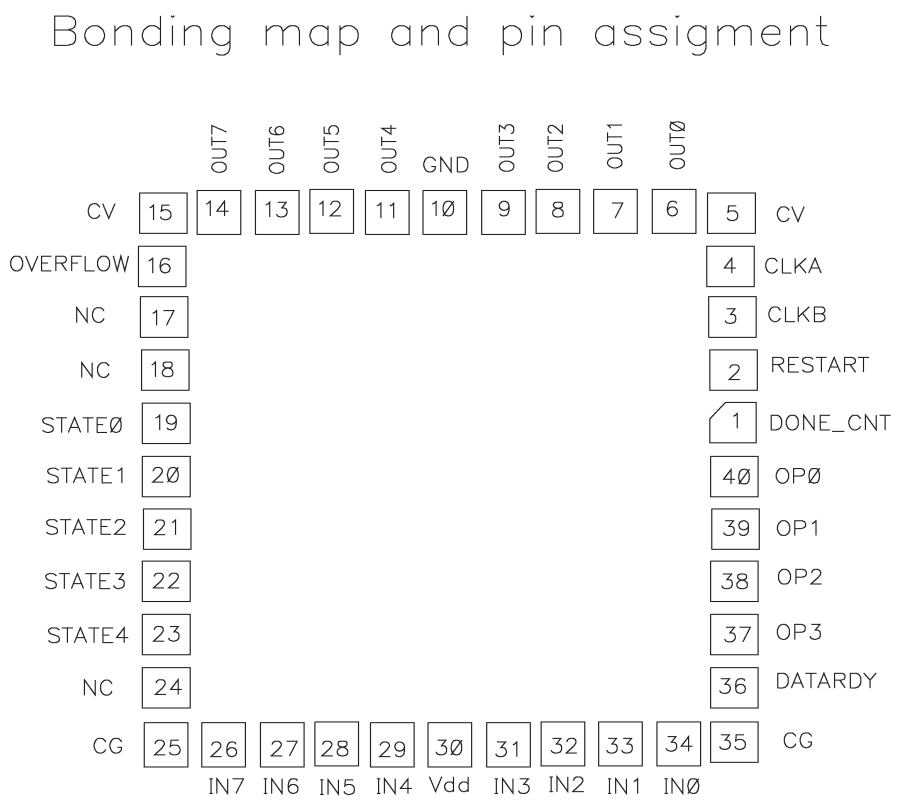
In order to utilize same 16–bit accumulator/shift register, 8–bit register, and the 8–bit adder for many different operations, the data flow will be controlled by many MUXs and passgates controlled by various control signals.

1.6 Input/Output Control

Inputs and outputs (mainly data) is handled by two main 8–bit bus systems. The input bus is hooked up to 5 different registers and output is hooked is shared by four different circuits. The input bus is common to all the inputs of registers and output is tri–stated when not in use.

For the inputs, the first clock cycle will accept the first operand, and second clock will accept the second operand (if there is one). For the outputs, data will output in one clock cycle, except the multiply where the results are in 16–bit form. The multiply will take two clock cycles to output its entire data.

1.7 Bonding map and Pin Assignment



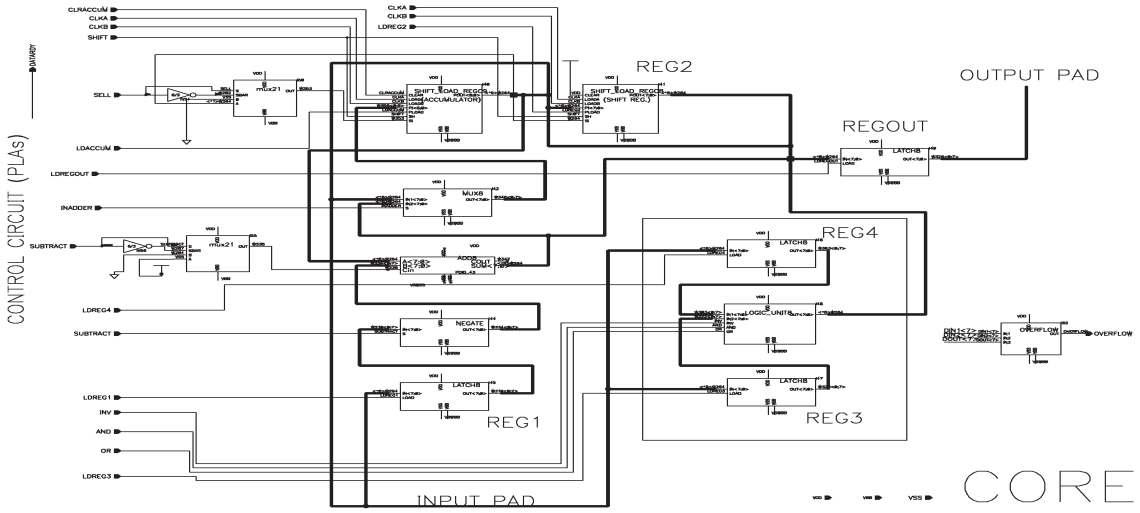
2. Circuit Design

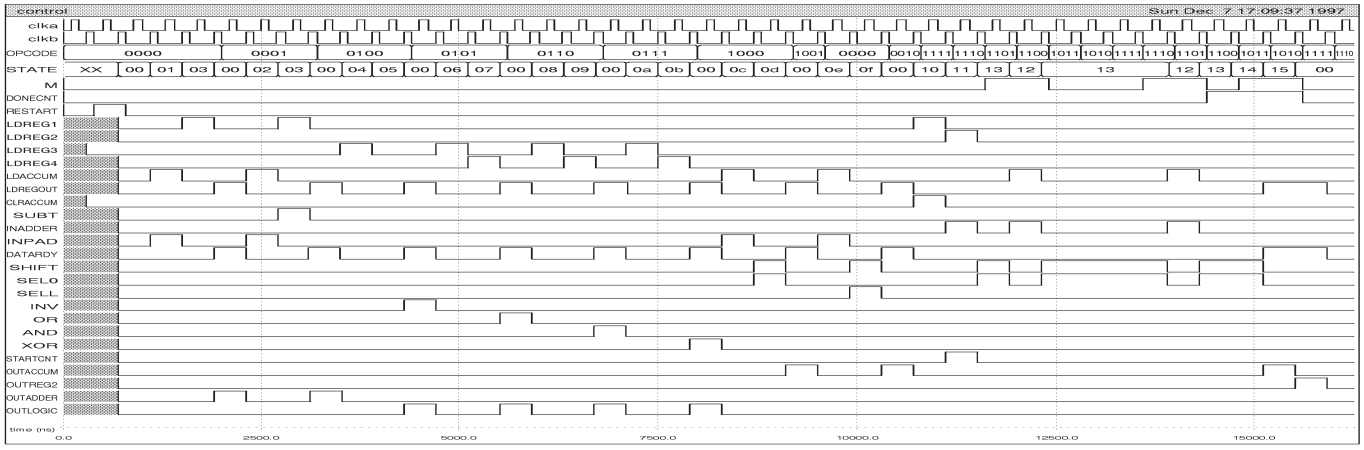
This design relies heavily on two types of latches: regular and shiftable and loadable latch. Regular latches are used to store the input and output data. Shiftable registers are used in multiply and shift functions. Adder circuit is a basic ripple–carry adder with no look–ahead mechanism, but has been carefully sized for optimal speed performance. MUXs and T–gates have been used to control and limit the data flow.

2.1 Logic Diagrams

Schematics of sub–cells and upper level cells (Continue on next page)

2.2 System Timing Diagrams





2.3 Irsim Results of Subcells

Continue on next page

2.4 Irsim Results of Entire chip

Top level cell is called “full_chip.” Continue after IRSIM results of the subcells

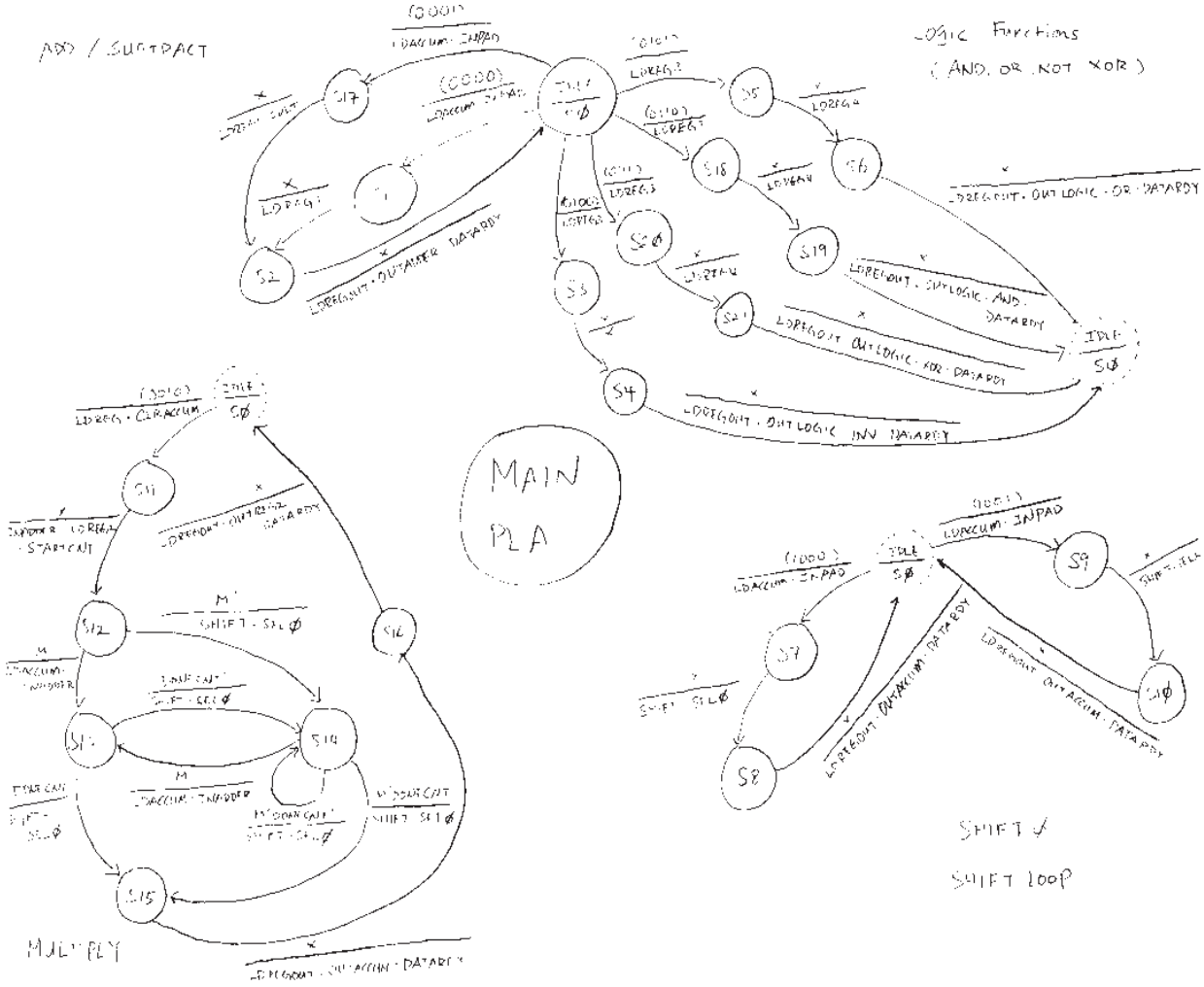
3. PLA Description

This circuit has two PLAs for control: the main controller and a counter. Counter is only activated during the MULTIPLY cycles and puts out DONECNT signal when it is done counting 8 shifts, and stops the main controller. The main controller takes care of other operations.

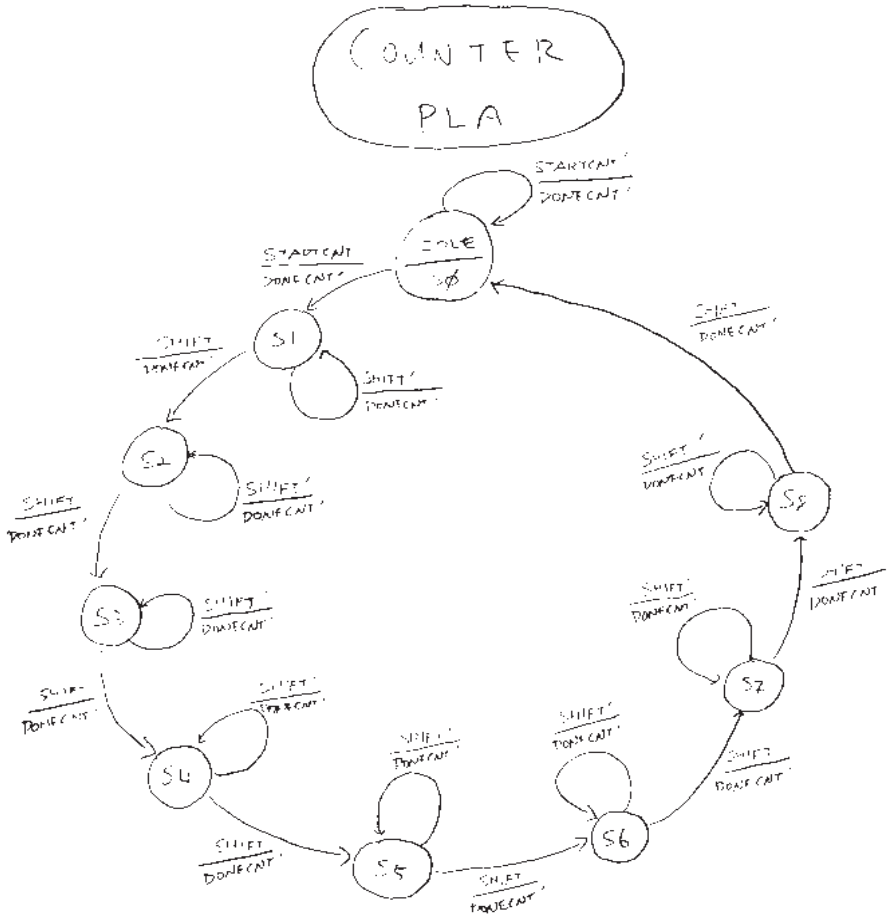
Every operation basically takes three steps of computation: load REGA, load REGB, and load REGout. It loads two operands into two separate registers, then the operators (logic or arithmetic) cells generate proper outputs between the clock, then loading REGout will put those results into the output bus.

3.1 State Diagram

Main PLA



Counter PLA



3.2 Input to MEG

MAIN CONTROLLER

```
—— control.meg
—— By: Han Kim
—— Main PLA controller for the ALU project
—— RESTART probably required on initial run

INPUTS:    OP0 OP1 OP2 OP3 M DONECNT RESTART;
OUTPUTS:   LDREG1 LDREG2 LDREG3 LDREG4 LDACCUM LDREGOUT CLRACCUM
           SUBT INADDER INPAD DATARDY
           SHIFT SEL0 SELL INV OR AND XOR
           STARTCNT OUTACCUM OUTREG2 OUTADDER OUTLOGIC;

RESET ON RESTART TO state0;


state0:  case (OP3 OP2 OP1 OP0)
—— ADD
           0 0 0 0   =>  state1      (LDACCUM=1 INPAD=1);
—— SUB
           0 0 0 1   =>  state17     (LDACCUM=1 INPAD=1);
—— MULT
           0 0 1 0   =>  state11     (LDREG1=1 CLRACCUM=1);
—— NOT
           0 1 0 0   =>  state3      (LDREG3=1 INV=0 OR=0 AND=0 XOR=0);
—— OR
           0 1 0 1   =>  state5      (LDREG3=1 INV=0 OR=0 AND=0 XOR=0);
—— AND
           0 1 1 0   =>  state18     (LDREG3=1 INV=0 OR=0 AND=0 XOR=0);
—— XOR
           0 1 1 1   =>  state20     (LDREG3=1 INV=0 OR=0 AND=0 XOR=0);
—— SHFT0
           1 0 0 0   =>  state7      (LDACCUM=1 INPAD=1);
—— SHFTL
           1 0 0 1   =>  state9      (LDACCUM=1 INPAD=1);
         endcase  =>  LOOP;

—— ADD / SUB
state1:       GOTO  state2   (LDREG1=1);
state17:       GOTO  state2   (LDREG1=1 SUBT=1);
state2:       GOTO  state0   (LDREGOUT=1 OUTADDER=1 DATARDY=1);

—— LOGIC FUNCTIONS
—— NOT
state3:       GOTO  state4;
state4:       GOTO  state0   (LDREGOUT=1 OUTLOGIC=1 INV=1 DATARDY=1);

—— OR
state5:       GOTO  state6   (LDREG4=1);
state6:       GOTO  state0   (LDREGOUT=1 OUTLOGIC=1 OR=1 DATARDY=1);
```

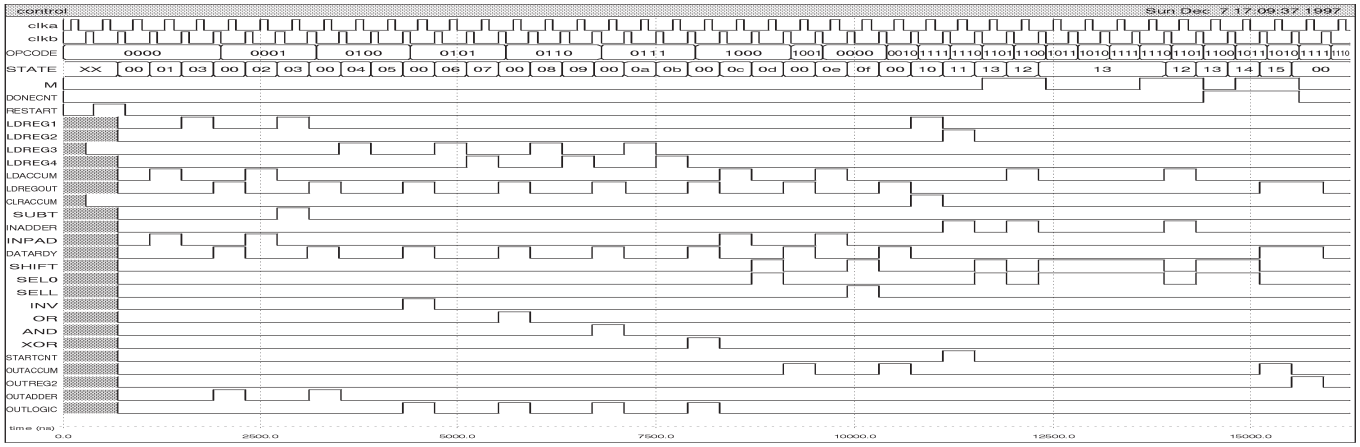
```
—AND
state18:      GOTO  state19  (LDREG4=1);
state19:      GOTO  state0   (LDREGOUT=1 OUTLOGIC=1 AND=1 DATARDY=1);

— XOR
state20:      GOTO  state21  (LDREG4=1);
state21:      GOTO  state0   (LDREGOUT=1 OUTLOGIC=1 XOR=1 DATARDY=1);

— SHIFT FUNCTIONS
— SHFT0
state7:       GOTO  state8   (SHIFT=1 SEL0=1);
state8:       GOTO  state0   (LDREGOUT=1 OUTACCUM=1 DATARDY=1);

— SHFTL
state9:       GOTO  state10  (SHIFT=1 SELL=1);
state10:      GOTO  state0   (LDREGOUT=1 OUTACCUM=1 DATARDY=1);

— MULTIPLY
state11:      GOTO  state12  (INADDER=1 LDREG2=1 STARTCNT=1);
state12:      IF M THEN state13 (LDACCUM=1 INADDER=1) ELSE state14 (SHIFT=1 SEL0=1);
state13:      IF DONECNT THEN state15 (SHIFT=1 SEL0=1) ELSE state14(SHIFT=1 SEL0=1);
state14:  case (M  DONECNT)
            0   0    =>  state14  (SHIFT=1 SEL0=1);
             0   1    =>  state15  (SHIFT=1 SEL0=1);
             1   ?    =>  state13  (LDACCUM=1 INADDER=1);
        endcase  =>   ANY;
state15:      GOTO  state16  (LDREGOUT=1 OUTACCUM=1 DATARDY=1);
state16:      GOTO  state0   (LDREGOUT=1 OUTREG2=1  DATARDY=1);
```

COUNTER

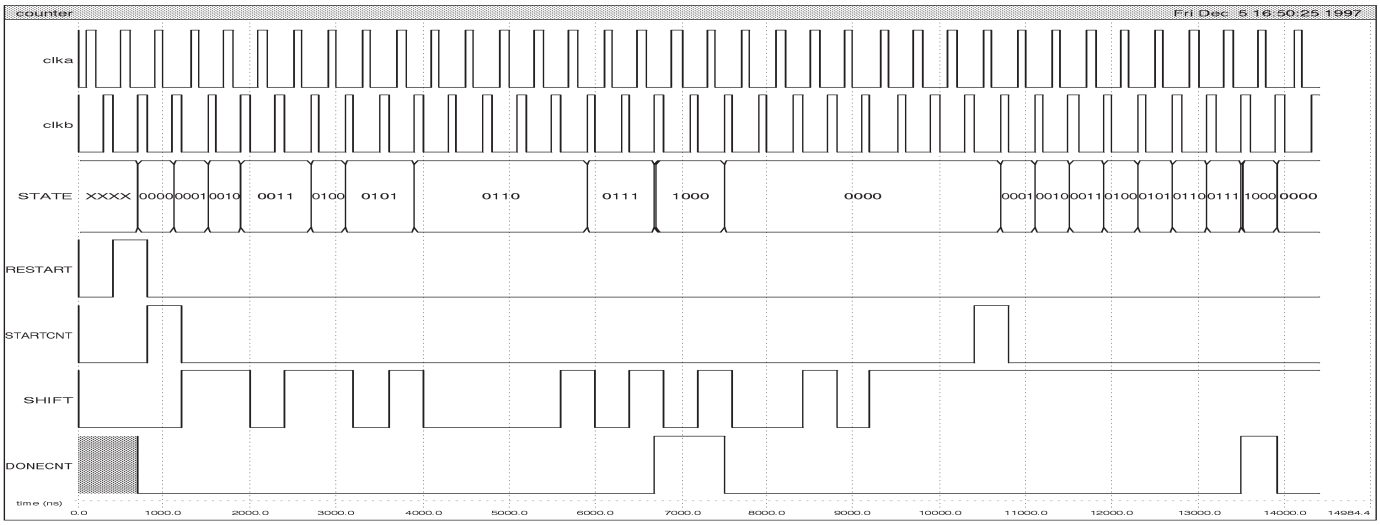
```
—    counter.meg
—    (8–1) SHIFT counter for the 16–bit multiplier
—    RESTART probably required on initial run

INPUTS:    RESTART STARTCNT SHIFT;
OUTPUTS:   DONECNT;

RESET ON RESTART TO state0;

state0:      IF STARTCNT THEN state1 (DONECNT=0) ELSE state0 (DONECNT=0);
state1:      IF SHIFT    THEN state2 (DONECNT=0) ELSE state1 (DONECNT=0);
state2:      IF SHIFT    THEN state3 (DONECNT=0) ELSE state2 (DONECNT=0);
state3:      IF SHIFT    THEN state4 (DONECNT=0) ELSE state3 (DONECNT=0);
state4:      IF SHIFT    THEN state5 (DONECNT=0) ELSE state4 (DONECNT=0);
state5:      IF SHIFT    THEN state6 (DONECNT=0) ELSE state5 (DONECNT=0);
state6:      IF SHIFT    THEN state7 (DONECNT=0) ELSE state6 (DONECNT=0);
state7:      IF SHIFT    THEN state8 (DONECNT=1) ELSE state7 (DONECNT=0);
state8:      IF SHIFT    THEN state0 (DONECNT=0) ELSE state8 (DONECNT=1);
```

3.3 Irsim simulation

Main_PLA



Counter_PLA



4. Circuit Layout

4.1 Plots of Subcells

Continue on next page

4.2 Cell hierarchy

Continue after subcells

4.3 Floor Plan

Continue after cell hierarchy

4.4 Full Plot of Chip

Continue after floor plan

5. Performance Analysis

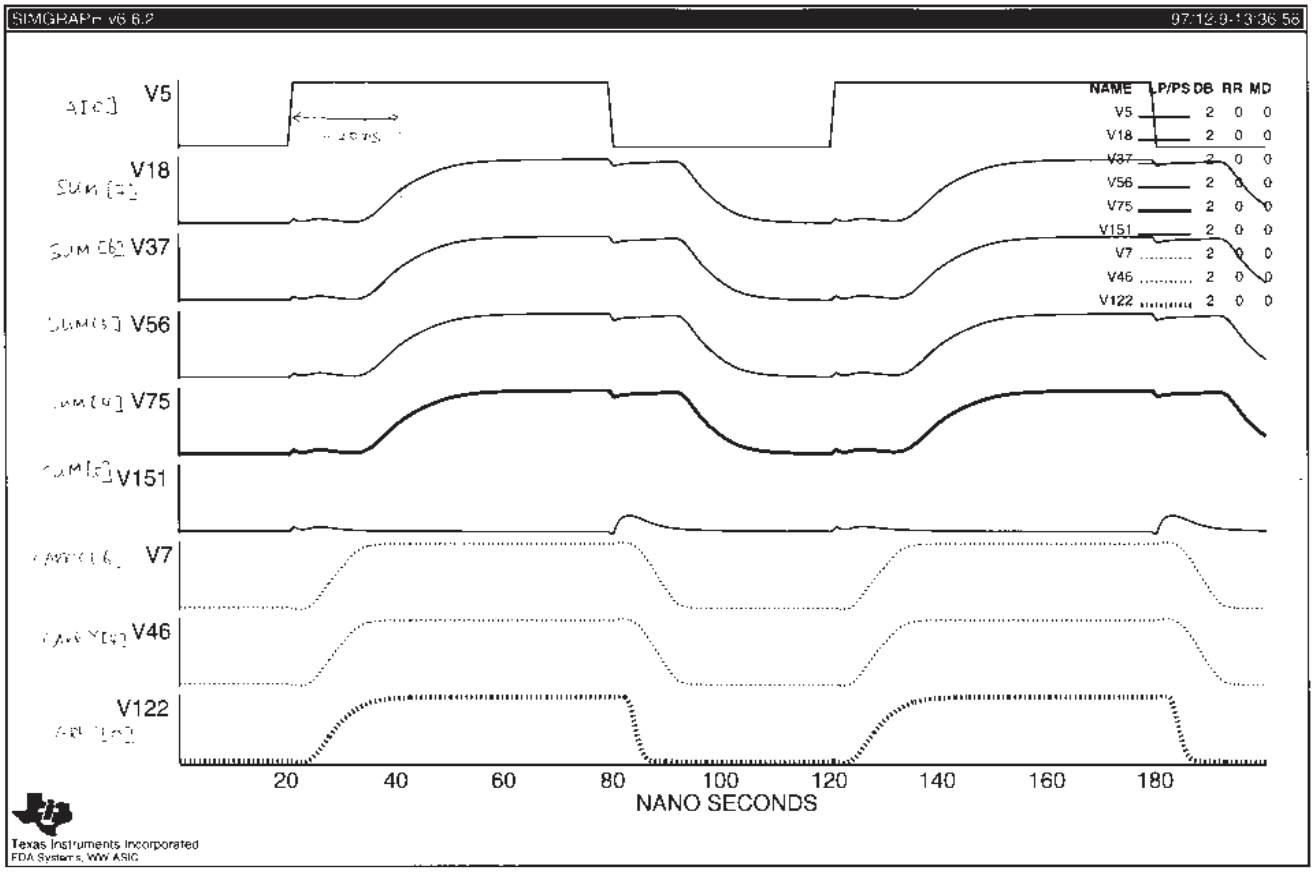
5.1 Crystal Analysis

Since the worst circuit for delay and clocking speed purpose is the adder8 cell, a SPICE simulation was performed before it was checked with Crystal. We believe SPICE gives a more accurate result, so the Crystal analysis was not necessary.

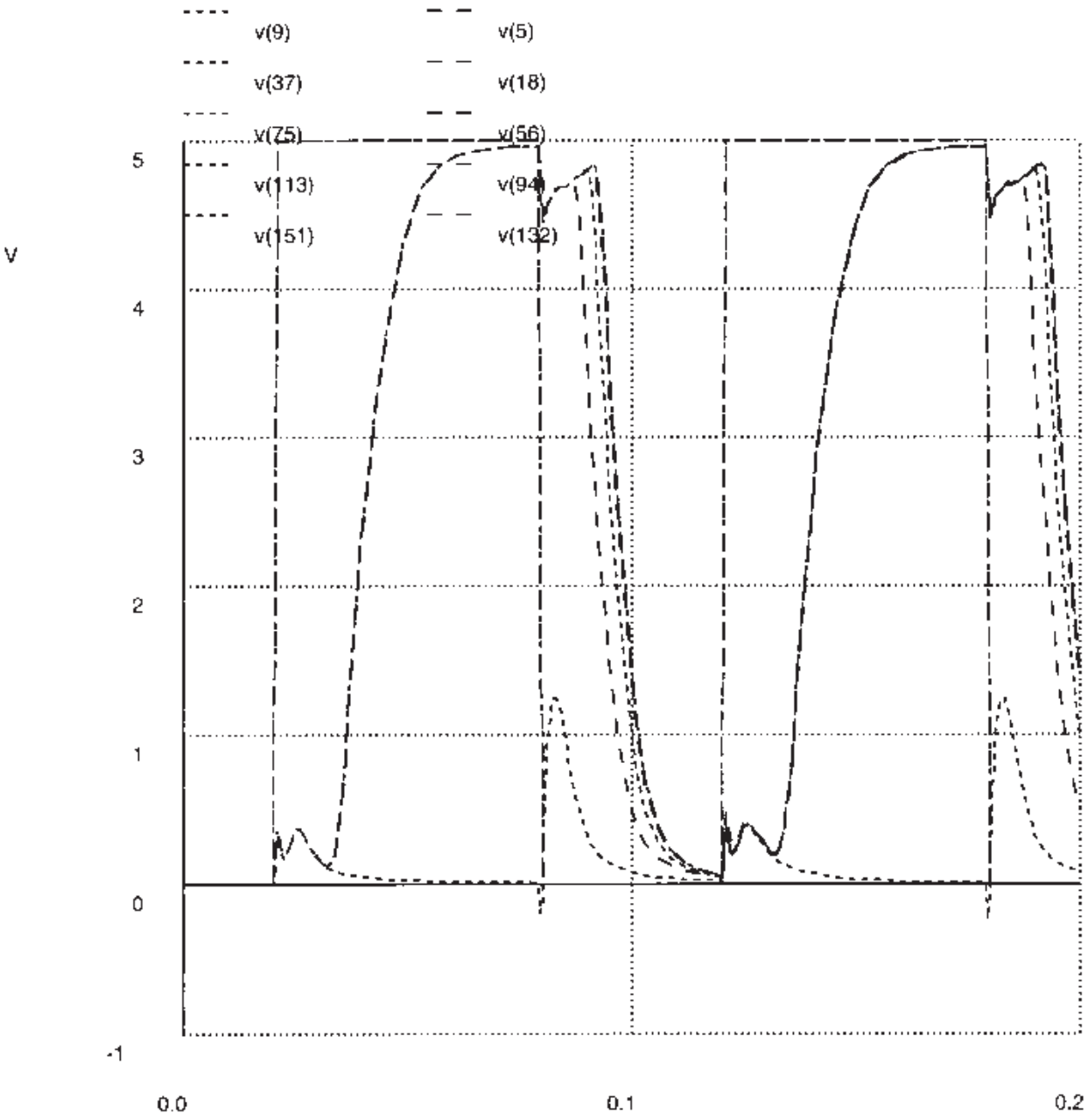
5.2 Spice Analysis of Critical Circuit (Adder8)

SPICE outputs

Adder 8



Adder 8



5.3 Maximum Clock Frequency

According to the SPICE simulation of the Adder8 cell, we can see that there is about 20ns delay between the input of the adder and last SUM bit switching (SUM[7]). This suggests that theoretically, we can clock the circuit up to 50 MHz (limited by the slowest section of the circuit, which is the Adder). With other factors and delays accounted in, we predict about 25 MHz of safe operation of this circuit.

6. Summary

The “Vanilla” Arithmetic and Logic Unit accomplished our two goals: essential and simple functions and fast operation. We accomplished a fairly simple design with some basic circuit elements that performs most major ALU functions (ADD, SUBT, etc.) We knew even before actual layout was done that the Adder will be our speed bottleneck, so a full custom and a careful layout was done, so that the delay in that circuit will be minimum.

6.1 References

1. Principles of CMOS VLSI Design; A Systems Perspective
   Weste, Neil H. E., Eshraghian, Kamran
   1993, AT&T

6.2 Comments on CAD tools

Generally the CAD tools were decent, considering that they are freeware and widely distributed. Learning curve is fairly easy on all of the tools that we used. We especially liked the PLA tool for synthesizing logic quickly and easily. Surprisingly, MAGIC had some features that some of the expensive professional layout tools does not have, such as an interactive design rule checker.