Microarchitecture Specification for the Back-propagation State Machine (BSM)

John DeLacy, Subbarao Vanka, Dan Bedell, Lea Williams, and Kamal Sarkez

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 88-016

Abstract

This technical report contains the Microarchitecture Specification for the Analog Processor Node or APN, which was designed in the 1987 OGC Advanced VLSI class, and fabricated by MOSIS. In addition, the results of chip testing are included. The BSM is designed to operate in conjunction with an APN by updating the weights in the APN according to the back-propagation of error algorithm. Groups of BSM can be connected into feedforward, layered networks.

Microarchitecture Specification:

BSM Chip

CSE 529 - 1987

John DeLacy, Subbarao Vanka

Dan Bedell, Lea Williams and Kamal Sarkez

August 13, 1987

1. Introduction

1.1. General Description

The Back Propagation State Machine (BSM) chip performs the adjustment of "weights" associated with inter-node connections in a neural-like connection network. The weights are used in the processing node (PN) associated with each BSM. The BSM calculates new weights for each iteration of the network.

The BSM implements the back propagation algorithm for a network which has input, output and hidden processing nodes. It calculates new weights for the four inputs to the PN associated with it and calculates error terms for the next lower level of nodes.

The following simplifying assumptions have been made to implement the BSM in silicon:

- each PN in the network connects to at most 4 other units,
- the parameter η is an externally programmable 4-bit value,
- The output function O_{pj}, for all PNs, is given by

$$O_{pj} = f(\sum_{i=1}^{4} O_i w_{ij}),$$

where
O_i represents one of the four inputs to the PN,
w_{ij} represents the weight for that input,
$f(x)$ is the function $\frac{1}{1 + e^{-x}}$, and

The BSM requires five sets of input data for its calculations: the value η, programmed from outside the network; the PN's inputs and output; an error term from the next higher level of nodes; and the previous weights used by the PN.

The BSM outputs two sets of data. First, it calculates the new weights for the PN and writes them into the PN. It then calculates an error quantity for each of the four new weights and sends them to the next lower level of nodes.

1.2. BSM Equations

The equation for the change in weight (Δw) used in the BSM is

$$\Delta w_i = O_i \eta O_j (1 - O_j) X \ .$$

Since $\Delta w = w_{new} - w_{old}$, the new weight calculated by the BSM is

$$w_{new_i} = O_i \eta O_j (1 - O_j) X + w_{old_i} \ .$$

The quantity X could be two different values, depending on whether the BSM is in an output node or a hidden node.

$$X_1 = T_j - O_j$$

or

$$X_2 = \sum_{k=1}^{4} \delta_k w_{jk}$$

(X_1 is for BSM in output node,
X_2 is for BSM in hidden node)

The variables are defined as follows:

η is a constant loaded beforehand,
O_i is one of four inputs to the PN,
O_j is the output of the PN,
T_j is the "teaching vector" or the expected output, and
$\delta_k w_{jk}$ is the error quantity from the next higher level node.

The BSM must calculate four error quantities for transmission to the next lower level of nodes. It calculates

$$\delta_j w_{ij} = O_j (1 - O_j) X w_{ij} \ .$$

1.3. BSM Algorithm

The algorithm executed by the BSM chip is:

```
Reset

    Input η value

    Input initial weights

    Do (Forever)

        {
          Input PN input values

          Input PN output value

          Wait for "GO" signal
```

Disassert "DONE" signal

Input X input (error term from level above)

$\delta = O_j(1 - O_j)X$

For (i = 1 to 4) do
{
$weight_i = \delta \eta O_i + weight_i$

write $weight_i$ to PN's $weight_i$

write (δ*$weight_i$) to lower level $node_i$
}
Assert "DONE" signal
}

2. External Interface

2.1. System Architecture

The Back Propagation State Machine (BSM) and the Processing Node (PN) form the basic processing unit in the neural-like connection architecture. The architecture allows upto 3 levels of such processing units. Each level consists of upto 4 processing units. The units are referred to as Input, Hidden or Output units depending on which of the 3 levels they belong to. Figure 2.1 shows the system architecture in terms of processing units.

The processing units are constructed using the BSM and the PN as building blocks. At the Output level each processing unit is made up of one BSM and one PN. For the Output units Data propagates only in the Forward direction (from Input to Hidden to Output levels). But for the Input and Hidden units data propagates in the Forward and Backward directions. For these processing units an extra PN is used for the backward path. Figure 2.1 shows the 2 types of processing units.

Each Input or Hidden processing unit is capable of communicating with upto 4 other processing units at the next higher level for Forward data propagation. Similarly each Output or Hidden unit is capable of communicating with upto 4 other processing units at the next lower level for Backward data propagation. Since this involves a large number of interconnections between processing units, a virtual connection is preferred to a physical connection. This is done via an addressing scheme which allows any processing unit to address 4 other units, and to access upto 8 different registers within that unit.

2.2. BSM External Signals

The signals that interface the BSM to the other chips in the system are shown in Figure 2.3. There are three distinct sections to the external interface.

2.2.1. System Bus Interface

The System Bus will be made up of three four-bit busses on the "a" and "b" interfaces. On the "a" interface the busses are bidirectional while on the "b" interface they are input busses. The busses are the Data bus, the Unit Address bus and the Register Address bus. All busses on the "a" interface and "b" interface are connected to the shared System bus. The System Bus uses a Request/Grant (Daisy chain) handshake protocol. (Refer to Figure 2.2.) It uses the following signals:

araddr_B1: Four-bit Bidirectional Address bus used to specify the Register address. The bus is driven by the BSM during accesses to an FPN or BPN. It is received by the BSM during accesses by an FPN or BPN to the BSM and used to select one of the internal registers of the BSM.

acaddr_B1: Four-bit output-only address bus used to specify the Unit address. The bus is driven by the BSM for accesses to an FPN or BPN.

adata_B1: Four-bit Bidirectional Data bus used to transfer data to or from the BSM. The bus is driven by the BSM during write operations by the BSM to an FPN or BPN. It is received by the BSM during write operations by an FPN or BPN to the BSM. It is also used by the host processor for initialization.

awtwr_1: Weight Write Strobe. This output is driven by the BSM to write new weight values into the FPN.

adelwr_1: Delta Write Strobe. This output is driven by the BSM to write new error values (δw_i)into the BPN.

axwren_1: X Write Enable. This input enables writing the error value from the BPN into the BSM.

avalwren_1: Value Write Enable. This input enables writing the input value from the FPN into the BSM.

braddr_B1: Four-bit Input Address bus used to specify the Register address. The bus is driven by an FPN during write operations to the BSM.

bcaddr_B1: Four-bit Input Address bus used to specify the Unit address. The bus is driven by an FPN or BPN during write operations to the BSM.

bdata_B1: Four-bit Input Data Bus used to transfer data to the BSM. The data lines are driven by the FPN to write the computed output into the BSM.

bwren_1: Write Enable. It is driven by the bwrstrb_1 signal from the FPN to strobe data into the BSM.

2.2.2. Host Processor Interface

reset_1: System Reset. Used to initialize the entire network to a known state.

rd_1: Host Processor Read Input. Used to read from internal registers within the BSM while cs_1 (Chip Select) is active.

wr_1: Host Processor Write Input. Used to write into internal registers within the BSM while cs_1 (Chip Select) is active.

cs_1: Chip Select Input. Used to put the BSM in an initialization or diagnostics mode. Normally inactive while the network is performing computations.

go_1: System Synchronization Input. Used to synchronize the computations of all the nodes in the network externally.

done_1: BSM Done Output. Used to indicate that the BSM has completed the computations and updated the FPN and BPN registers.

clk_1: Clock Phase 1 Input. Phase one of a 2-phase non-overlapping clock.

clk_2: Clock Phase 2 Input. Phase two of a 2-phase non-overlapping clock.

2.2.3. Bus Access Control Signals

reqin_1: Bus Request Input. This signal, when low, enables an internal request onto the output request pin, breqout_1. In addition to breqout_1 being low, the busbusy_1 signal must also be low for the BSM to become the bus owner.

reqout_1: Bus Request Output. This signal is driven by the BSM when no higher priority agent has requested the bus and no other agent (lower or higher priority) is currently using the bus. Along with breqin_1, this signal is used to form a Daisy-chained priority resolution circuit for bus access control.

busy_1I: Bus Busy Bidirectional. This signal is activated by the node which is currently using the bus. All nodes receive the signal as an input and hold off their accesses to the bus if this signal is active.

3. Internal Architecture

The BSM consists of seven modules which together perform the functions of interfacing with the System busses, storing data values internally, and performing computations within the BSM. Each module is described in detail in the following sections. See Figure 3.1 for details.

3.1. Bus Interface Module (BiMOD)

The Bus Interface Module handles the protocol necessary for the chips to share the system bus (the bidirectional data bus and bidirectional address busses). It is assumed that all signals will be used during phase 1. Thus, all outputs will be created with a phase 2 strobe. In fact, all outputs are created by a standard dynamic PLA.

3.1.1. Signals to/from off chip

clocks: Phase 1 and phase 2; self explanatory.

wstrb_1: Write strobe. This output of the BiMOD indicates that valid data is on the busses and should be strobed into the addressed unit. This output will probably have to be an open drain signal. Note that in the BSM, this output does not go off chip; it is multiplexed inside ResMOD.

reset_1: Reset. This input to the chip will reset all the state machines to their intitial states.

reqin_1: Request in. This input comes from a chip with higher priority for use of the bus. If it is asserted, the BiMOD will not attempt to use the bus. Also, in the first clock period after access has been granted, the BiMOD will check reqin_1 and abort an output attempt if it is asserted. This handles simultaneous requests.

reqout_1: Request out. This output of the BiMOD will signal to the lower priority chips that the BiMOD demands the bus. It will be connected to one input of an OR gate, the other input being connected to reqin_1 and the output of the OR gate from the next higher priority source. When reqout_1 of a given chip is asserted, the external gates will cause reqin_1 of all lower priority chips to be simultaneously asserted.

busy_1: Busy. This is a bidirectional signal that is an open drain output of the chip and an input for the BiMOD. The gate for the open drain driver will be driven by reqout_1. Busy_1 will indicate when the bus is in use for all chips, regardless of priority. This signal is to prevent a higher priority chip from intervening in the middle of a data transaction.

3.1.2. Signals to/from on chip

outreq_1: Output request. This signal comes from another module. When it becomes asserted, the BiMOD state machine will check the bus. If the bus is available, then an attempt is made to send data out; otherwise, the BiMOD will wait until the bus is available to attempt to send data. Outreq_1 must stay asserted until the BiMOD successfully gets the bus (i.e., until outenab_1 goes high).

outenab_1: Output enable. This output signal from the BiMOD enables the output buffers, placing the contents of the data bus and the address busses onto the interchip bus. It should drive the output buffers of all address and data busses.

3.1.3. BiMOD Operation

The BiMOD controls access to the output bus of a chip. Since several chips can use a bus, and more than one could request access simultaneously, a mechanism must be provided to handle the potential conflict. The BiMOD contains a state machine that arbitrates bus requests among the chips that can write to the interchip bus.

The BiMOD state machine will produce an output on reqout_1 and consequently assert busy_1 when it receives a request for output (outreq_1 asserted) and busy_1 and reqin_1 are disasserted. If reqin_1 is asserted one clock cycle later by a higher priority chip, the state machine aborts the output attempt.

Once the BiMOD gets control of the bus, it continuously asserts reqout_1 for the entire write cycle. Since reqout_1 also drives the gate of the open drain buffer for busy_1I (the "I" is because the open drain buffer will invert the signal; there needs to be an inverter between busy_1I and the input busy_1 to BiMOD), assertion of reqout_1 will also assert busy_1. These signals cause all chips on the bus to wait until the current user is finished with the bus. The write cycle is three clock cycles long; the data and address busses will be valid for all three cycles (except for the initial charge-up delay, which will be somewhere around 40 to 60 ns). The ouput strobe signal (wstrb_1) will occur in the second clock cycle of the write cycle.

3.1.4. BiMOD Use

In order for a chip to use the BiMOD, the following sequence of operations is suggested. When a chip desires output, it will assert outreq_1. This should stay asserted until the chip is granted use of the bus, which is indicated by outenab_1 going high. When outenab_1 goes low again, the write operation was successfully completed. The chip is not allowed to change the inputs to the data and address output buffers until outenab_1 goes low again.

3.2. Result Output Module (ResMOD)

The Result Output Module provides the sequencing and control necessary for writing the results of the BSM's computations to the various destination units. The ResMOD performs two operations; it writes the four newly computed weights into the FPN of the local node, and it writes the product δw_i for i = 0 to 3 to the four lower level BPN's.

3.2.1. Signals to/from off-Chip

clocks: Phase 1 and phase 2; self explanatory.

reset_1: Reset. This input to the chip will reset all the state machines to their intitial states.

go_1: This signal is an input from the host system. When it is asserted, the BSM begins its calculations. It is used by ResMOD to reset the state of resdone_1 and return the state machine to its idle condition.

awtwr_1: Weight write strobe. This output of the ResMOD goes to the other FPN's on the bus indicating that valid weight data is on the busses and should be strobed into the addressed unit. This output will probably have to be an open drain signal.

adelwr_1: Error write strobe. This output of the ResMOD goes to the other BPN's on the bus indicating that valid error data (δw_i) is on the busses and should be strobed into the addressed unit. This output will probably have to be an open drain signal.

resdone_1: ResMOD done. This output of the ResMOD signals to the external processor that the BSM is finished with all back propagation activities. (It is called "done_1" in the external interface.) It will be asserted when the BSM has written all of its results into the BPN's and FPN's. It will be disasserted again when the next go_1 signal is received.

3.2.2. Signals to/from on-Chip

compdone_1: Computation done. This input to the ResMOD comes from the Computation Sequencer Module (SeqMOD). It signals the ResMOD that all computations are done and the ResMOD then begins its writing operations.

wrstrb_1: Write strobe. This input to the ResMOD comes from the Bus Interface Module (BiMOD). It will be routed either to awtwr_1 or adelwr_1, depending on which PN is being written to.

outenab_1: Output enable. This input signal comes from the BiMOD. It indicates that an output cycle on the system bus is under way. When this signal falls low again, it indicates that the write operation is over.

outreq_1: Output request. This output of the ResMOD goes to the BiMOD. It is used to request an output write cycle on the interchip bus.

rrd_1: ResMOD read. This output from ResMOD goes to DecMOD. It requests a read cycle to the register file.

regaddr_B1: Register address bus. This output bus from ResMOD goes to the register address output buffers and carries the address of the register currently being written.

unitaddr_B1: Unit address bus. This output bus from ResMOD goes to the unit address output buffers and carries the address of the unit currently being written.

rraddr_B1: ResMOD register address bus. This bus goes to DecMOD to select the register being read in the register file.

r8_B1: Register 8 ouput. This bus carries the contents of register 8 in the register file (the unit address programmed into the chip). It is a static output of the register.

3.2.3. ResMOD Operation

The ResMOD sequences through two write operations; one to the FPN to write all four new weights, and one to the BPN's to write the newly-computed error quantities. It uses BiMOD to gain access to the bus. It uses DecMOD to create the proper register select and register read signals to RegMOD.

When compdone_1 is asserted, the ResMOD state machine begins the sequence of eight register write operations. First it writes the four newly-computed weights to the FPN. It puts the BSM's unit address on the unitaddr_B1 bus and writes one new weight with 00 on the regaddr_B1 bus, the next weight with 01 on regaddr_B1, and so on for all four new weights.

Next, the ResMOD will write the four error quantities to the next lower level BPN. It does this by putting the BSM's unit address on the regaddr_B1 bus and writing one new error quantity for each of the four unit addresses, 0 to 3, on the unitaddr_B1 bus.

When the ResMOD is finished with all eight writes, it asserts done_1 and returns to its initial state.

3.2.4. ResMOD Use

The Computation Sequencer Module (SeqMOD) must assert compdone_1 when it is finished with all computations and has all the results stored away in the appropriate registers. Once ResMOD sees compdone_1, it will begin its operation and go to completion. It needs no further enabling. Once ResMOD has started, no other module or entity can be allowed to use the register file. There is no provision made to detect or avoid simultaneous accesses to the register file.

3.3. Decode Module (DecMOD)

The Decode module performs the decoding of addresses to generate the internal select, read and write signals to the Register file module. Accesses to the Register file can be from outside the BSM (from an FPN or BPN) or from inside the BSM (from the Sequencer Module or the Result Module). All accesses are controlled by the Decode Module which multiplexes the control signals onto the select, read and write signals which go to the Register file.

3.3.1. Signals to/from off Chip

araddr_B1: Register address from the System bus connected to the "a" interface of the BSM.

acaddr_B1: Chip address from the System bus connected to the "a" interface of the BSM.

braddr_B1: Register address from the System bus connected to the "b" interface of the BSM.

bcaddr_B1: Chip address from the System bus connected to the "b" interface of the BSM.

adelwr_1: Control signal generated by the Result module to indicate that a new Error quantity is being written out to the bus.

awtwr_1: Control signal generated by the Result module to indicate that a new Weight is being written out to the bus.

axwren_1: Control signal from the System bus to write the Xin data value into the BSM.

avalwren_1: Control signal from the System bus to write a new Input value into the BSM.

bwren_1: Control signal from the System bus to write a new Output (Oj) value into the BSM.

cs_1: Chip select input from the Host Processor.

wr_1: Write input from the Host Processor.

rd_1: Read input from the Host Processor.

3.3.2. Signals to/from on Chip

xinrd_1: Control signal from the Sequencer module to read the Xin value from the Registerfile.

etard_1: Control signal from the Sequencer module to read the Eta value from the Registerfile.

idrd_1: Control signal from the Result module to read the Unit ID value from the Register file.

ojrd_1: Control signal from the Sequencer module to read the Output (Oj) value from the Register value.

SOrd_1(1:4): Four control signals from the Sequencer module to read the four Input values from the Register file.

SWrd_1(1:4): Four control signals from the Sequencer module to read the four Weights from theRegister file.

RWrd_1(1:4): Four control signals from the Result module to read the four Weights from the Register file.

SWwr_1(1:4): Four control signals from the Sequencer module to write the four Weights into the Register File.

SEwr_1(1:4): Four control signals from the Sequencer module to write the four Error quantities into the Register file.

RErd_1(1:4): Four control signals from the Result module to read the four Error quantities from the Register file.

rs_1(0:15): Sixteen select signals from the Decode module to select one of sixteen regsitersfrom the Register file for a read or a write operation.

rreg_1: Read signal from the Decode module to read the selected register in the Register file.

wreg_1: Write signal from the Decode module to write the selected register in the Register file.

3.4. Register File Module (RegMOD)

This module contains sixteen four bit registers and the logic required to control the reading and writing of these registers.

3.4.1. Signals to/from off Chip

RegMOD does not directly interface to any external signals.

3.4.2. Signals to/from on Chip

in_B1: Four bit data bus used to write to a selected register.

out_B1: Four bit data bus used to read a selected register.

rreg_1: A signal which enables a register read.

wreg_1: A signal which enables a register write.

r0_B1: A four bit bus tied to the output (Q) bits of register 0.

r1_B1: A four bit bus tied to the output (Q) bits of register 1.

r2_B1: A four bit bus tied to the output (Q) bits of register 2.

r3_B1: A four bit bus tied to the output (Q) bits of register 3.

r4_B1: A four bit bus tied to the output (Q) bits of register 4.

r5_B1: A four bit bus tied to the output (Q) bits of register 5.

r6_B1: A four bit bus tied to the output (Q) bits of register 6.

r7_B1: A four bit bus tied to the output (Q) bits of register 7.

r8_B1: A four bit bus tied to the output (Q) bits of register 8.

r9_B1: A four bit bus tied to the output (Q) bits of register 9.

r10_B1: A four bit bus tied to the output (Q) bits of register 10.

r11_B1: A four bit bus tied to the output (Q) bits of register 11.
r12_B1: A four bit bus tied to the output (Q) bits of register 12.
r13_B1: A four bit bus tied to the output (Q) bits of register 13.
r14_B1: A four bit bus tied to the output (Q) bits of register 14.
r15_B1: A four bit bus tied to the output (Q) bits of register 15.

rs0_1: Select line for register 0.
rs1_1: Select line for register 1.
rs2_1: Select line for register 2.
rs3_1: Select line for register 3.
rs4_1: Select line for register 4.
rs5_1: Select line for register 5.
rs6_1: Select line for register 6.
rs7_1: Select line for register 7.
rs8_1: Select line for register 8.
rs9_1: Select line for register 9.
rs10_1: Select line for register 10.
rs11_1: Select line for register 11.
rs12_1: Select line for register 12.
rs13_1: Select line for register 13.
rs14_1: Select line for register 14.
rs15_1: Select line for register 15.

3.4.3. Internal Structure of RegMOD.

RegMOD has 16 4-bit registers which can be selected by individual Select lines. Along with a rreg_1 and wreg_1 the registers can be read or written to. During read operations the data from the register is put out on the individual output data bus (r0_B1 through r15_B1). The same data is also put out on a common tristateable output data bus called out_B1. During write operations the data from the common input data bus called in_B1 is written into the selected register.

3.4.3.1. Individual register

Each register consists of four independent RS type latches, configured as "data latches", with Resets tied to Sets through inverters. An array of four latches, with Enable inputs tied to a single register select line, comprises an individual four bit register.

3.4.3.2. Register file

Corresponding "Set" inputs of the data latches of each of the sixteen registers are wired together with the corresponding line of the four bit input bus, in_B1. Assertion of a register's select line, the write signal, wreg_1, and PH1, allows the register's latches to load in_B1 on the trailing edge of PH1. The contents of the register are therefore valid in PH2.

Corresponding bits of the sixteen register outputs are gated onto the output bus, out_B1, so that when a register select signal, the read signal, rreg_1, and PH1 are asserted, the selected register's value is placed on out_B1.

Each register also directly drives its own output bus (one of: r0_B1, r1_B1, ... , r15_B1). These bus signals are valid during PH1 and can be accessed in parallel by the other modules on the chip.

3.5. Sequence Control Module (SeqMOD)

This module generates the control signals required to sequence the ALU through the microinstruction sequence needed to perform the BSM's internal computations. The computation sequence is triggered by the "GO" signal received by the BSM from external logic. During the computation the BSM computes the new weights and error quantities. At the end of the computation the Computation Sequencer generates a "Computation Done" signal to the Bus Access Control logic. The new weights and error quantities are then written out over the System Bus to the FPN and the BPN's by the Result module (ResMOD). The Computation Sequencer consists of 2 state machines and 3 combinational logic blocks which generate the control signals to transfer data between registers and to perform arithmetic operations using the ALU. The state machines and combinational logic are described below.

3.5.1. Signals to/from off Chip

go_1: The go_1 signal triggers SeqMOD to start the computation sequence. This signal is received from an external system synchronization module.

reset_1: The reset_1 signal is used to reset all internal state variables within SeqMOD.

clk_1: Phase 1 of the 2-phase clock used to synchronize all the state machines.

clk_2: Phase 2 of the 2-phase clock used to synchronize all the state machines.

3.5.2. Signals to/from on Chip

in_B1: Internal 4-bit data bus used to transfer data to RegMOD. SeqMOD drives the data onto this bus at the same time as it generates a write signal for one of the registers.

out_B1: Internal 4-bit data bus used to transfer data from RegMOD to SeqMOD. SeqMOD generates a read signal and expects data to be put on this bus from one of the registers.

xbus_B1: Internal 6-bit data bus used to transfer one of the operands from the SeqMOD to AmMOD. All intermediate results such as the contents of the temporary registers within SeqMOD are transfered over this bus.

ybus_B1: Internal 4-bit data bus used to transfer the second operand from SeqMOD to AmMOD. Values such as "Eta", the weights and the input values from the register file are transfered over this bus.

done_1: Indicates that AmMOD has completed the current operation, in response to an Add or Multiply command from SeqMOD.

ojrd_1: Enables the Oj register contents onto the input bus of the Lookup Table within SeqMOD.

xrd_1: Enables the most significant bit of the Xin register onto the input of a latch within SeqMOD.

etard_1: Enables the "Eta" value onto the ybus_B1 during the computation of new weights and error quantities.

sord_1(1:4): These are 4 control signals used to read the Input values O1 through O4 out of RegMOD. They are outputs from SeqMOD to DecMOD.

swrd_1(1:4): These are 4 control signals used to read the Weights W1 through W4 out of RegMOD. They are outputs from SeqMOD to DecMOD.

swwr_1(1:4): These are 4 control signals used to write the Weights W1 through W4 into RegMOD. They are outputs from SeqMOD to DecMOD.

swwr_1(1:4): These are 4 control signals used to write the Errors Err1 through Err4 into RegMOD. They are outputs from SeqMOD to DecMOD.

mult_1: This signal initiates a multiplication operation using AmMOD. The signal is an output from SeqMOD to AmMOD.

add_1: This signal initiates an addition operation using AmMOD. The signal is an output from SeqMOD to AmMOD.

done_1: This signal indicates the end of an Addition or Multiplication operation. The signal is an output from AmMOD to SeqMOD.

compdone_1: This signal is an indication to ResMOD that the computation is complete and that the new Weights and Error values are in RegMOD.

3.5.3. Internal structure of SeqMOD

3.5.3.1. Sequence State Machine

The Sequence state machine implements the computation algorithm for the BSM. It has 7 states and it works along with a 4-state counter to generate the control signals needed to read values out of the RegMOD or the temporary registers within SeqMOD, to control the AmMOD and to write the results of intermediate or final computations into appropriate registers.

3.5.3.2. 4-State Counter for Weights and Errors

The 4-State Counter is used in conjunction with the Sequence state machine to determine which of the weights or errors is being computed at any given time. The state of the counter is used to generate the read or write signals to one of the four weight or error registers.

3.5.3.3. Lookup Table

The Lookup Table logic generates the Oj(1-Oj) value using the Oj value from RegMOD. This block consists of combinational logic whose output is ready to be used whenever the ojrd_1 signal goes active.

3.5.3.4. Control Logic

The Control logic block consists of combinational logic which decodes the state of the Sequence state machine and the Counter and some external signals to generate the read, write and computation signals.

3.5.3.5. Encode Logic

The Encode logic block consists of combinational logic which encodes signals generated by the Control logic block and provides the encoded address and read and write signals to DecMOD, to access registers from RegMOD.

3.5.3.6. Temporary Registers

Three 6-bit temporary registers, Reg1, Reg2 and Reg3, are used to store intermediate values which are common to the rest of the computation.

3.6. Adder/Multiplier Module (AmMOD) (version 3)

The Adder/Multiplier module adds or multiplies any two integers. It can add any two 8-bit unsigned data values or two 7-bit signed values. The signed data is in two's complement form. It can multiply two 6-bit unsigned data values and returns a 12-bit answer.

3.6.1. External Signals (off chip)

reset_1: The reset signal is used to reset all the pla inputs and outputs and to set the a register (areg) to zero.

ph1: Phase 1 of the 2-phase clock is used to load the input registers and set up the inputs and latch the outputs for the plas. It also is used to signal the load for the output register when the arithmetic operation is finished.

ph2: Phase 2 of the 2-phase clock is used to latch the inputs of the plas,to reset areg, and set done_1 when the operation is finished.

3.6.2. External Signals (on chip)

xbus_B1: Internal 8-bit data bus (6 of which are used by the bsm, on that chip the 2 msb are grounded) used to transfer one of the arithematic operands to AmMOD.

ybus_B1: Internal 8-bit data bus (4 of which are used by the bsm, on that chip the 4 msb are grounded) used to transfer the second operand to AmMOD. Values used in any arithmetic operation are transfered over these two buses.

obus_B1: Internal 12-bit data bus (6 of which are used by the bsm) used to transfer the results of an addition or multiplication operation out of AmMOD to another module or off chip. Obus_b1 is valid after done_1 is high and stays valid until the next out overwrites it.

done_1: Indicates that AmMOD has completed the current operation, in response to an add or multiply command. The done_1 signal goes high in phase 2 following a load of the oreg_B1 in phase 1. This allows the output on the obus_B1 to be used in the next phase 1.

add_1: This signal causes AmMod to add the two numbers given on the input busses.

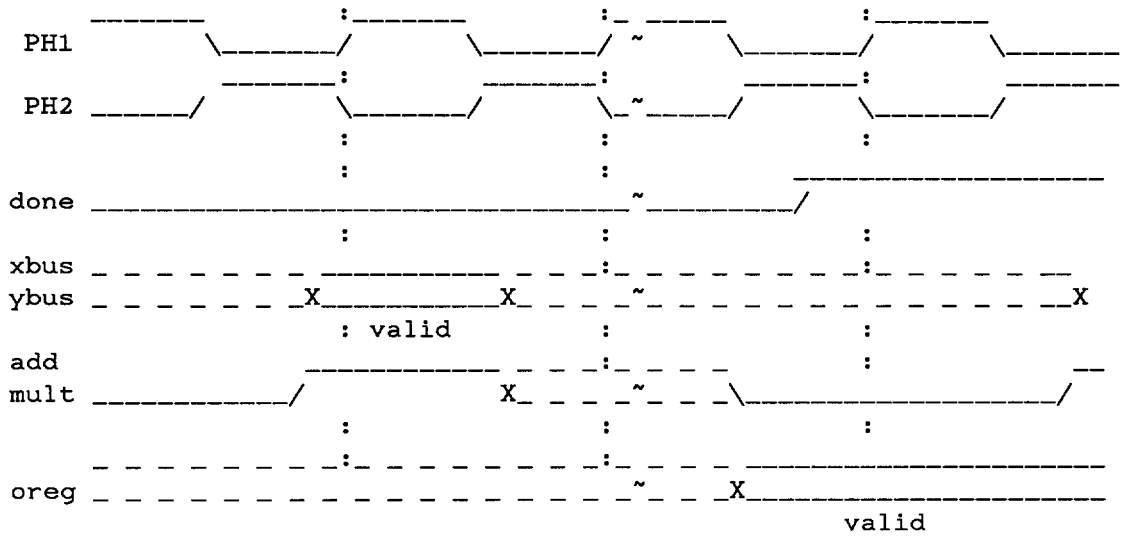
mult_1: This signal causes a multiplication of the two inputs to occur.

3.6.3. Internal Functionality

This module contains 3 submodules or units an adder/accumulator unit, a control unit, and a counter unit(see attached block diagram). The timing requirements are shown in figure 3.6.1.

3.6.3.1. Adder/accumulator unit

The adder/accumulator contains four registers, 8 full adders, and some logic gates. Three of the registers are loaded/unloaded from the previously mentioned buses. Two 8-bit registers xreg_B1 and yreg_B1 are loaded from xbus and ybus respectively and oreg_B1 a 12-bit register is unloaded by obus. The other register is a 13-bit accumulator (areg_B) that stores the intermediate sums during multiplication. The xreg and the yreg are loaded when an add or mult,ph1, and i or done are high. The signal i_2 comes from the control unit and is high if AmMOD is idle. The other signals have been discussed previously. The xreg is loaded directly from the bus. The ybus is loaded into bits 0-8 of the yreg if add is



1) All input signals must be valid during PH1.
2) Mult and add should be low before done goes high and stay low until after the output is read.
3) The output is vaild the first PH1 after done is high.
4) Oreg is valid until the next output writes over it.

FIGURE 3.6.1. AmMOD Timing diagram

high and bits 2-8 if mult is high. The areg is gate delayed to be loaded after xreg during phase one. If the operation is addition xreg is loaded into bits 4-11 of the areg. If the operation is multiplication xreg is loaded into bits 0-5. Multiplcation is done by a sequence of shifts and adds, so the areg is shifted right one bit when ps_2 is recieved from the control unit. The adder/accumulator unit also contains a 8-bit adder. The 8-bit adder is 8 full adders which preforms the addition operation. When pa_2 is recieved from the control unit a parallel add is performed on yreg and bits 4-11 of the areg with the results stored in areg.

3.6.3.2. Control unit

The control unit controls the function of what the module is doing. It is a pla with a small amount of external logic. The input signals for the pla are n_2, a_2, k_2, m_2, and r_2. n_2 is mult exclusive ored with add, a_2 is add and not mult,k_2 is the count signal, m_2 is the lsb of areg, and r_2 is the same as reset_2. The output signals for the pla are ps_2, pa_2, d_2, and i_2. ps_2 tells the areg to shift right 1, pa_2 tells the adders to do a parallel add, d_2 is done_2 before it is phase 2 trapped, and i_2 is one when AmMOD is idle. Ever phase 1 if the done signal or idle is high it checks to see if the add or mult signal is high if not it remains idle (see attached flow chart).

If the add signal is high it begins a sequence of events to cause an add to occur. After the x register is loaded in phase one, during that same phase the x register is loaded into the accumulator and d_2 and/or i_2 are set to 0. An addpulse is given to all the adders and a parallel add is done. In the next phase one the answer is latched into the accumulator register. The data in the accumulator is then shifted right four places and loaded into the output register. The number of shifts is keep track of by the counter unit. The done signal is then set to a one.

If the mult signal is given the controller begans a sequence of events to cause a multiply to occur. After the x register is loaded into the accumulator register done and/or idle are set to 0. The count, which is a loop arround counter initially set to zero, is checked to see if it is equal to 6. If it is k is set

to one, if not k remains equal to zero. The least significant bit of areg (m_2) is checked, if it is a one add (pa_2) and shift (ps_2) areg right one place. If it is a zero skip add and shift areg right 1 place. Then check k if k is zero go back to check count step above and repeat sequence. If k is a one load the areg into the output register. Set the done signal to one.

3.6.3.3. Counter unit

The counter unit is a state machine which counts to three for addition and five for multiplication. The counter is used during addition to place the sum in the lsb of the areg, so it can be loaded into the oreg. During multiplication it is used to keep track of how many bits have been shifted. When all 8-bits have been shifted the multiplication is complete and the counter tells the control unit to stop.

3.7. Input/Output Module (PinMOD)

3.7.1. Signals to/from off Chip

xgo_1: External signal which starts the BSM computation of new weights and error quantities.

xrd_1: External signal which performs a read operation from one of the internal registers of the BSM, as specified by araddr_B1 or braddr_B1.

xwr_1: External signal which performs a write operation to one of the internal registers of the BSM, asa specified by araddr_B1 or braddr_B1.

xcs_1: External signal which selects the BSM for a read or write operations which originate from the Host processor.

xavalwren_1: External signal which performs a write operation to one of the "input value" registers of the BSM from the "a" interface of the BSM.

xaxwren_1: External signal which performs a write operation to the "Xin" register of the BSM from the "a" interface of the BSM.

xbwren_1: External signal which performs a write operation to one of the "input value" registers of the BSM from the "b" interface of the BSM.

xreqin_1: External signal which indicates to the BSM that a higher priority BSM in the network has requested access to the System bus.

xbusy_1: External signal which indicates to the BSM that the System bus is being used by some other BSM or PN.

xreset_1: External System reset signal which resets all internal state machines within the BSM.

xbcaddr_B1: External "chip address" bus on the "b" interface of the BSM. This is an input bus.

xbraddr_B1: External "register address" bus on the "b" interface of the BSM. This is an input bus.

xacaddr_B1: External "chip address" bus on the "a" interface of the BSM. This is a bidirectional tri-state bus.

xaraddr_B1: External "register address" bus on the "a" interface of the BSM. This is a bidirectional tristate bus.

3.7.2. Signals to/from on Chip

go_1: Internal signal which starts the BSM computation of new weights and error quantities.

rd_1: Internal signal which performs a read operation from one of the internal registers of the BSM, as specified by araddr_B1 or braddr_B1.

wr_1: Internal signal which performs a write operation to one of the internal registers of the BSM, asa specified by araddr_B1 or braddr_B1.

cs_1: Internal signal which selects the BSM for a read or write operations which originate from the Host processor.

avalwren_1: Internal signal which performs a write operation to one of the "input value" registers of the BSM from the "a" interface of the BSM.

axwren_1: Internal signal which performs a write operation to the "Xin" register of the BSM from the "a" interface of the BSM.

bwren_1: Internal signal which performs a write operation to one of the "input value" registers of the BSM from the "b" interface of the BSM.

reqin_1: Internal signal which indicates to the BSM that a higher priority BSM in the network has requested access to the System bus.

busy_1: Internal signal which indicates to the BSM that the System bus is being used by some other BSM or PN.

reset_1: Internal System reset signal which resets all internal state machines within the BSM.

bcaddr_B1: Internal "chip address" bus on the "b" interface of the BSM. This is an input bus.

braddr_B1: Internal "register address" bus on the "b" interface of the BSM. This is an input bus.

acaddr_B1: Internal "chip address" bus on the "a" interface of the BSM. This is a bidirectional tri-state bus.

araddr_B1: Internal "register address" bus on the "a" interface of the BSM. This is a bidirectional tristate bus.

3.7.3. Internal structure of PinMOD

PinMOD consists of input and output buffers required to interface to the System Bus on the "a" and "b" interfaces of the BSM. For input signals PinMOD implements the input buffers. For output and bidirectional signals PinMOD also implements the tristate output buffers which are controlled by internal signals.

4. Timing Specifications

4.1. BSM Write Operations

The BSM writes the new weights to the FPN or the error quantities to the previous level BPN's using araddr_B1, acaddr_B1, adata_B1, awtwr_1, and adelwr_1. The timing diagram is shown in Figure 4.1.

4.2. FPN Write Operations

The FPN writes the Input values into the next level FPN's and BSM's using araddr_B1, acaddr_B1, adata_B1, avalwren_1. It writes the computed output, O_j, into the current level BSM with bdata_B1, bcaddr_B1, bwren_1, with braddr_B1 hardwired to don't cares. The timing diagram is shown in Figure 4.1.

4.3. BPN Write Operations

The BPN writes the computed output value into the BSM over the system bus using the adata_B1, using xwren_1 for the write strobe. The BSM ignores both araddr_B1 and acaddr_B1 for this operation. The timing diagram is shown in Figure 4.1.

4.4. Host Read/Write Operations

The Host processor can read or write registers within the BSM, FPN or BPN by asserting the reset_1 signal and the appropriate cs_1 signal. The timing diagram is shown in Figure 4.1.

4.5. Bus Access Control

Bus access is controlled by the Daisy-chained reqin_1 and reqout_1, along with the busy_1 signal. An example of this protocol is shown in Figure 4.1.

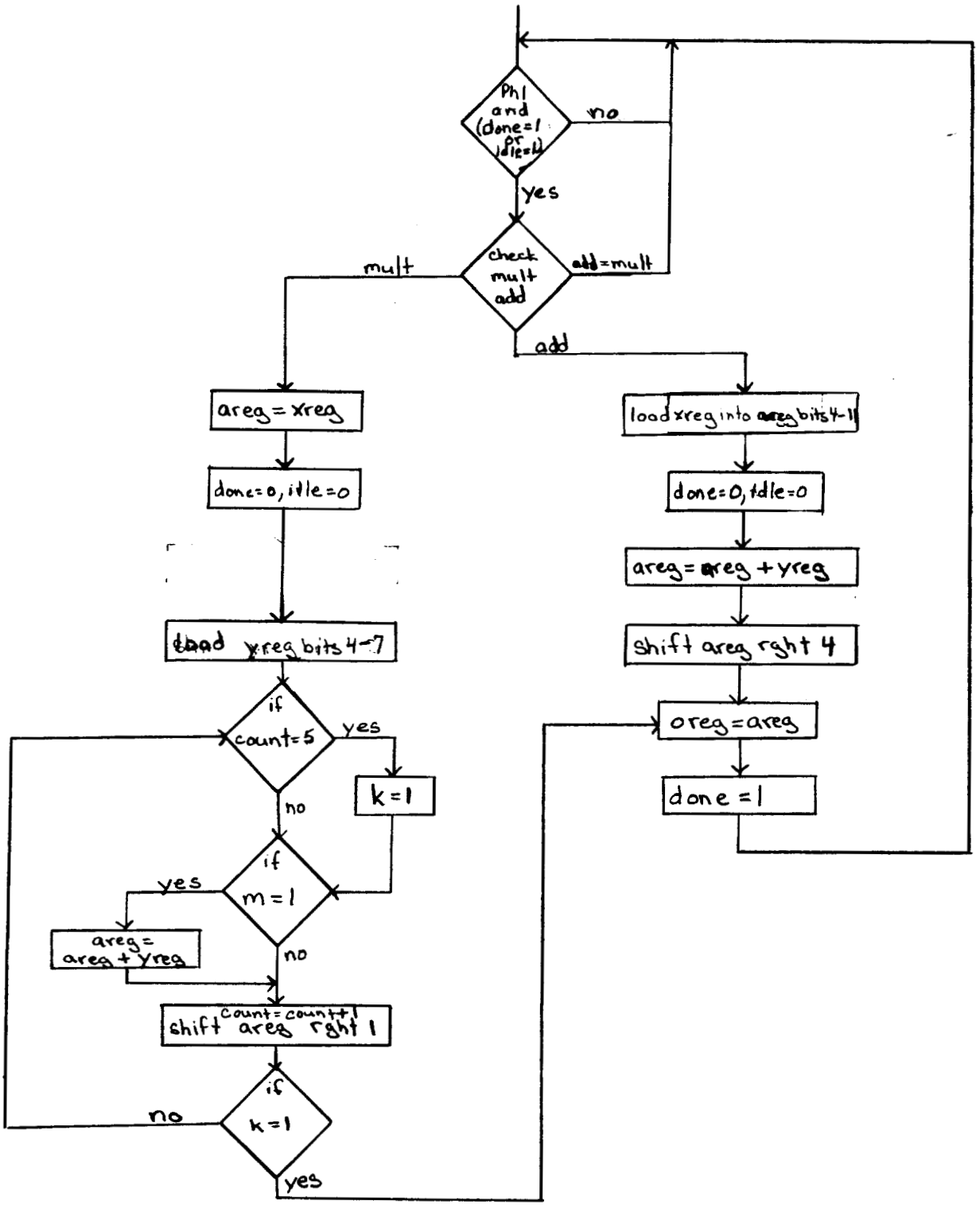


Figure Adder/Multiplier Execution Sequence

Appendix A

BSM IMPLEMENTATION ESTIMATES August 13, 1987

1. Transistor count, layout, and implementation time estimates.

The following table (BSM Table) contains a rough estimate of layout area, transistor count, number of cell placements, number of required connections and a time estimate for completions of each section of the BSM chip. The layout area is in micron's squared times 1000 and is approximated from standard cell sizes currently available in scmos for magic. The pla sizes are estimated from these formulas: height is 120+8p and width is 70+16i+8o. The variable p is the minterms, i is the inputs, and o is the outputs. The time was wstimated by the following equation: layout time is (connections+placements)10min..

Table BSM. BSM IMPLEMENTATION ESTIMATES

Section	Trans#	Area K microns sq	Placements	Connections hours	Time
ResMOD	158	373.4	23	73	45
AmMOD	1174	3225.6	178	369.5	145
BiMOD	26	88.8	11	8	8
DecMOD	398	1575	73	200	45.5
SeqMOD	1322	TBD	102	636	123
RegMOD	818	2235	63	155	36.3
PinMOD	70+	80.3+	67	120	67
TOTAL					

Appendix A.1

AM IMPLEMENTATION ESTIMATES August 13, 1987

1. Transistor count, layout, and implementation time estimates.

The following table (AM Table) contains a rough estimate of layout area, transistor count, number of cell placements, number of required contections and a time estimate for completions of each section of the AM module. The layout area is in lambda's squared times 1000 and is approximated from standard cell sizes currently available in scmos for magic. The pla sizes are estimated from this formula: height is 120+8p and width is 70+16i+8o. The variable p is the minterms, i is the inputs, and o is the outputs. The time was estimated by the following equation: Layout time is (connections+placements)10min.

Table AM. AMMOD IMPLEMENTATION ESTIMATES

Section	Area K lam**2	Trans#	Placements	Connections	Layout time 6hr/wk	netlist/mossim 6hr/wk
AMMOD**	42.8	1 PLA	1	5		
	73.7	65 pgates	65	130		
	7.9	7 inv	7	7		
	1.3	1 nor2	1	1.5		
	1.4	1 nand2	1	1.5		
	9.5	5 nand3	5	10		
Total	93.7	182	80	155	3.75	3.75
REGISTER	2.3	2 pgates	2	4		
UNIT	4.5	4 inv	4	4		
2-8bit reg	-	-	-	24		
1-10bit reg	*26	*26	13	15		
Total	290.3	312	19	47	1.83	1.83
CONTROL	69.2	1 PLA	1	8		
UNIT	5.7	5 pgate	5	10		
	3.4	3 inv	3	3		
	1.9	1 nand3	1	2		
	1.3	1 nor2	1	1.5		
Total	81.4	26	11	24.5	.99	.99
ACAD	22.7	20 pgate	20	40		
UNIT	5.7	5 inv	5	5		
	2.6	2 nor2	2	3		
	1.9	1 nor3	1	2		
	5.7	3 nand3	3	6		
	41.4	13 dlatchr	13	26		
Total	80	238	44	82	3.5	3.5
ADDER	9.1	8 inv	8	8		
SUBUNIT	3.9	3 nor2	3	4.5		
	1.9	1 nor3	1	2		
	4.1	3 nand2	3	4.5		
	1.9	1 nand3	1	2		
8 adders	*8	*8	8	20		
Total	166.8	416	24	51	2.08	2.08
TOTAL	716.8	1174	178	369.5	12.15	12.15

** These counts are for only the devices used to connect the units in the AM module.

2. TIME ESTIMATE FOR AmMOD

The total time to implement is 24.3 weeks X 6hr/week = 145 hours.

3. LAYOUT ESTIMATE FOR AmMOD

Converting from lambda to microns and adding 100% for interconnect: 716.8K X 1.5 X 1.5 X 2 = 3,225,600 square microns.

Appendix A.1

SEQMOD IMPLEMENTATION ESTIMATES

Table SEQMOD. SEQMOD IMPLEMENTATION ESTIMATES

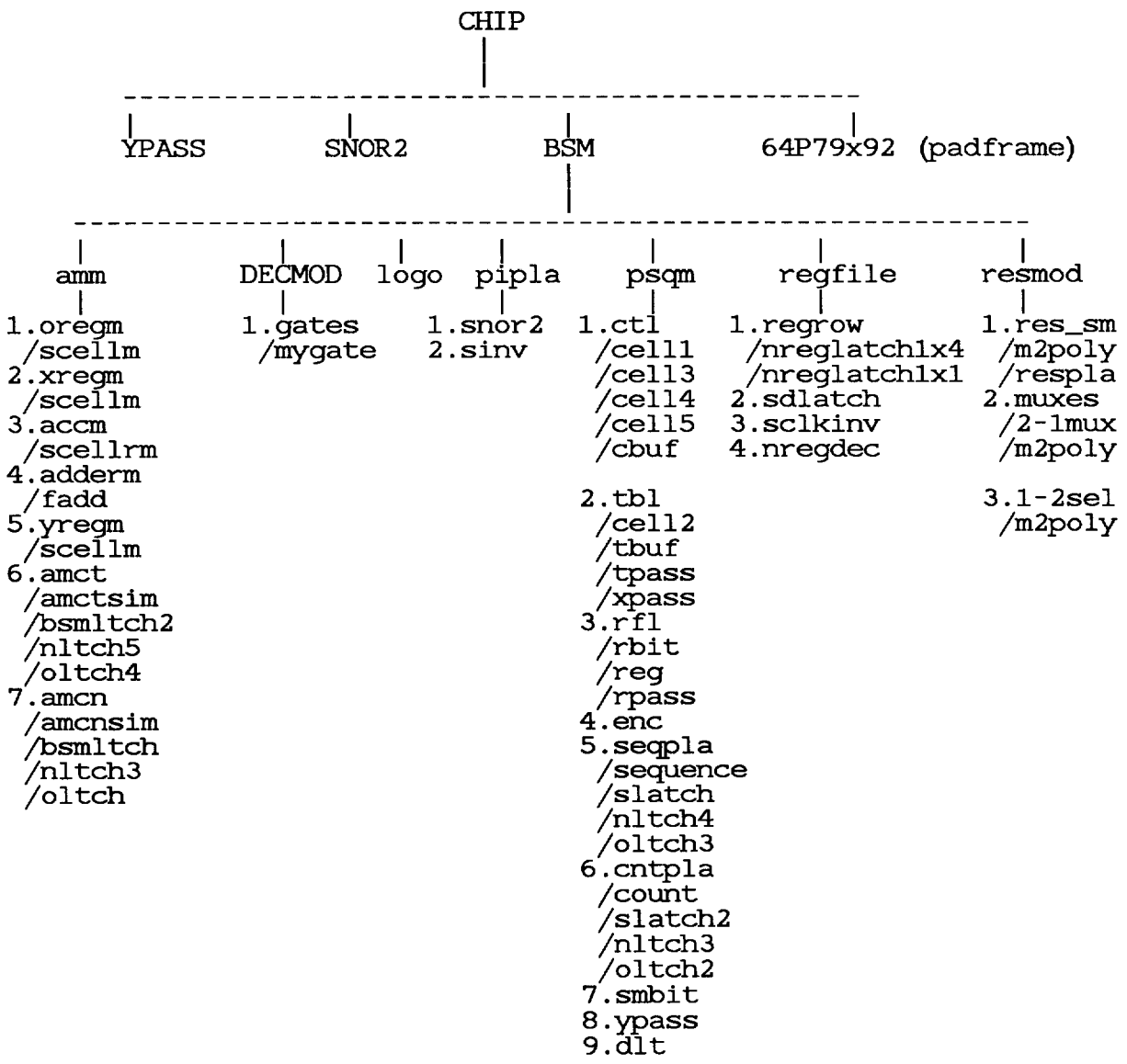
Section	Area K lam**2	Trans#	Placements	Connections	Layout time 6hr/week	netlist/mossim 6hr/week
CONTROL	TBD	9 nand2	9	TBD		
	TBD	18 nand3	18	TBD		
	TBD	6 nor2	6	TBD		
	TBD	1 nand4	8	TBD		
	TBD	8 inv	8	TBD		
	TBD	192	42	115		
ENCODE	TBD	8 nand2	8	TBD		
	TBD	1 nand3	1	TBD		
	TBD	2 nand4	2	TBD		
	TBD	1 nor2	1	TBD		
	TBD	1 nor3	1	TBD		
	TBD	6 inv	6	TBD		
	TBD	76	19	45		
TABLE	TBD	30 nand4	30	TBD		
	TBD	15 nand3	15	TBD		
	TBD	5 nor2	5	TBD		
	TBD	5 inv	5	TBD		
	TBD	360	5	188		
SEQUENCE	TBD	1 PLA	1	10		
	TBD	162	1	10		
COUNT	TBD	1 PLA	1	6		
	TBD	66	1	6		
REGISTERS	TBD	18 Cells	18	108		
	TBD	180	18	108		
LATCHES	TBD	7 Latches	7	42		
	TBD	70	7	42		
TOTAL	TBD	1322	102	636	123	40

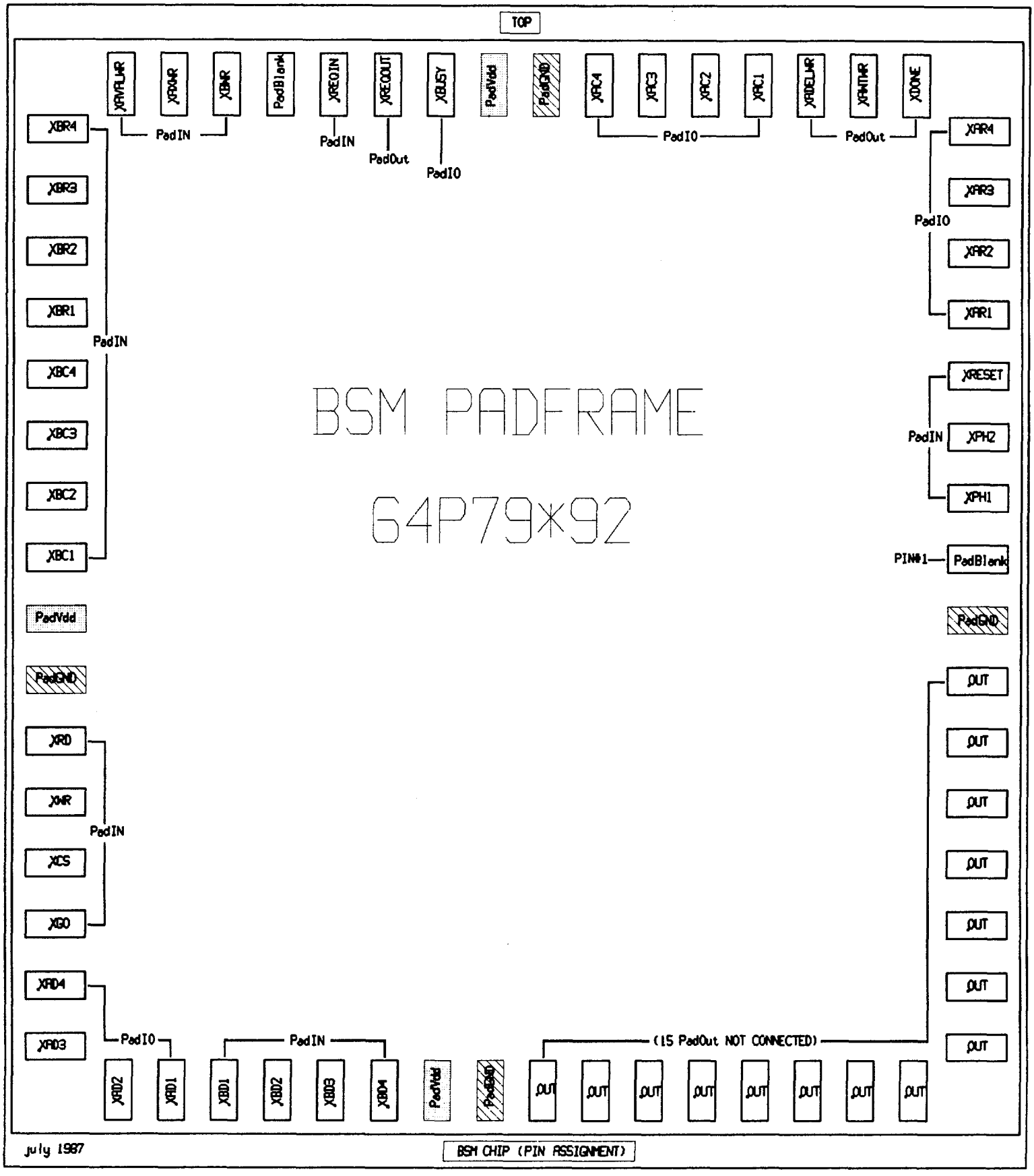
BSM : Summary of Layout Estimates

Module	Placements	Interconnects	Transistors	PLA's
DecMOD	73	200	398	0
RegMOD	27	393	1130	0
ResMOD	31	65	144*	1 (8,14,20)
BiMOD	7	13	18*	1 (6,6,6)
SeqMOD	102	636	1322	2
AmMOD	TBD	TBD	TBD	TBD
PinMOD	53	70	314	0
Total	293	1377	3326	4

* = Transistor count for PLA's not included

BSM CHIP LAYOUT CELLS'HIERARCHY

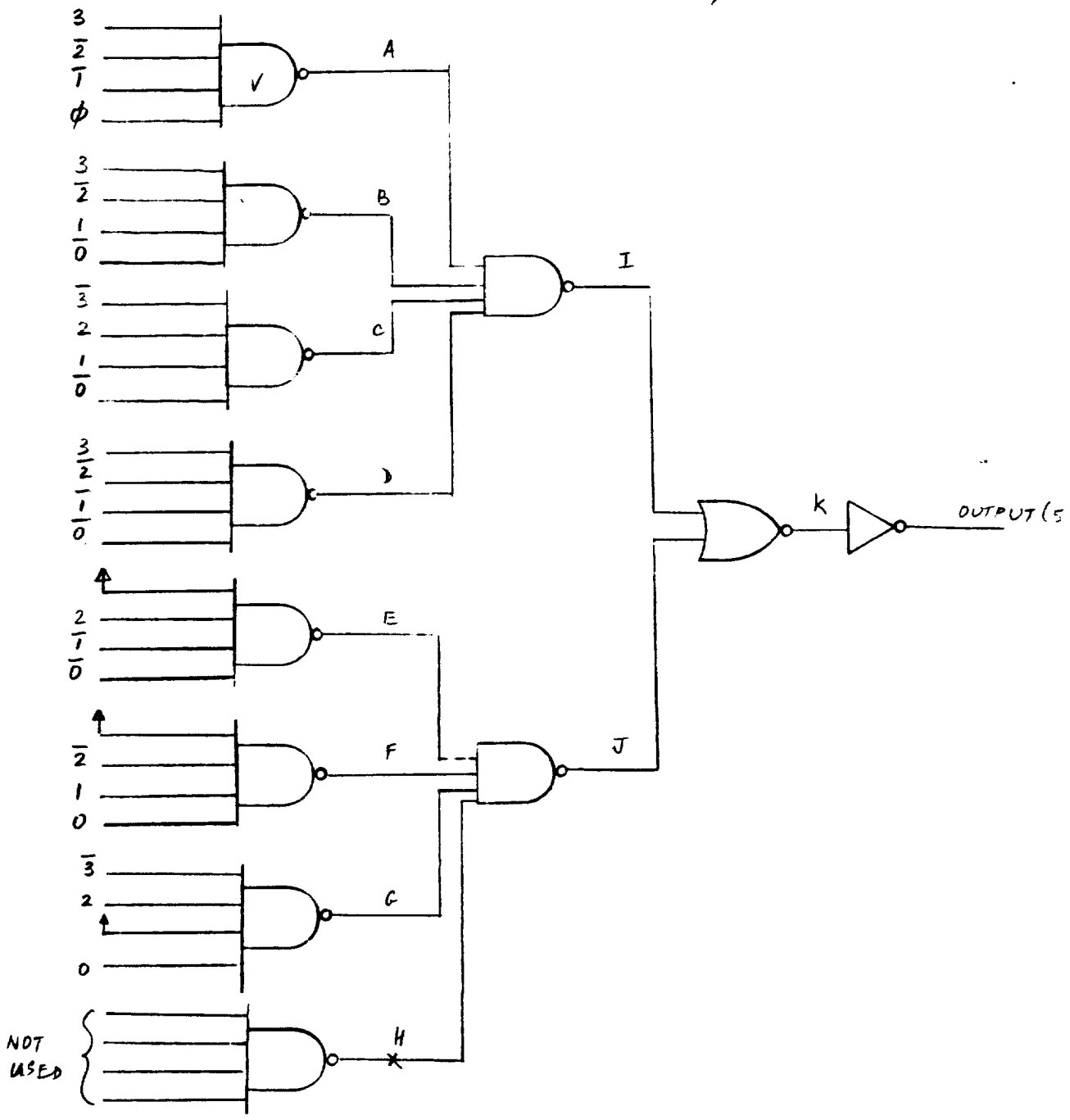



TOP
XAVAILWR
XARAWR
XBWR
PadBlank
XREQIN
XREQOUT
XBUSY
PadVdd
PadGND
XRC4
XRC3
XRC2
XRC1
XADELWR
XFAWTWR
XDONE
XBR4
XBR3
XBR2
XBR1
XBC4
XBC3
XBC2
XBC1
PadVdd
PadGND
XRD
XWR
XCS
XGO
XRD4
XRD3
XAR4
XAR3
XAR2
XAR1
XRESET
XPH2
XPH1
PadBlank
PadGND
OUT
PadIN
PadIO
PadOut
BSM PADFRAME
64P79*92
PIN#1
(15 PadOut NOT CONNECTED)
XAD2
XAD1
XBD1
XBD2
XBD3
XBD4
july 1987
BSM CHIP (PIN ASSIGNMENT)

SEQMOD - TABLE PAGE 1

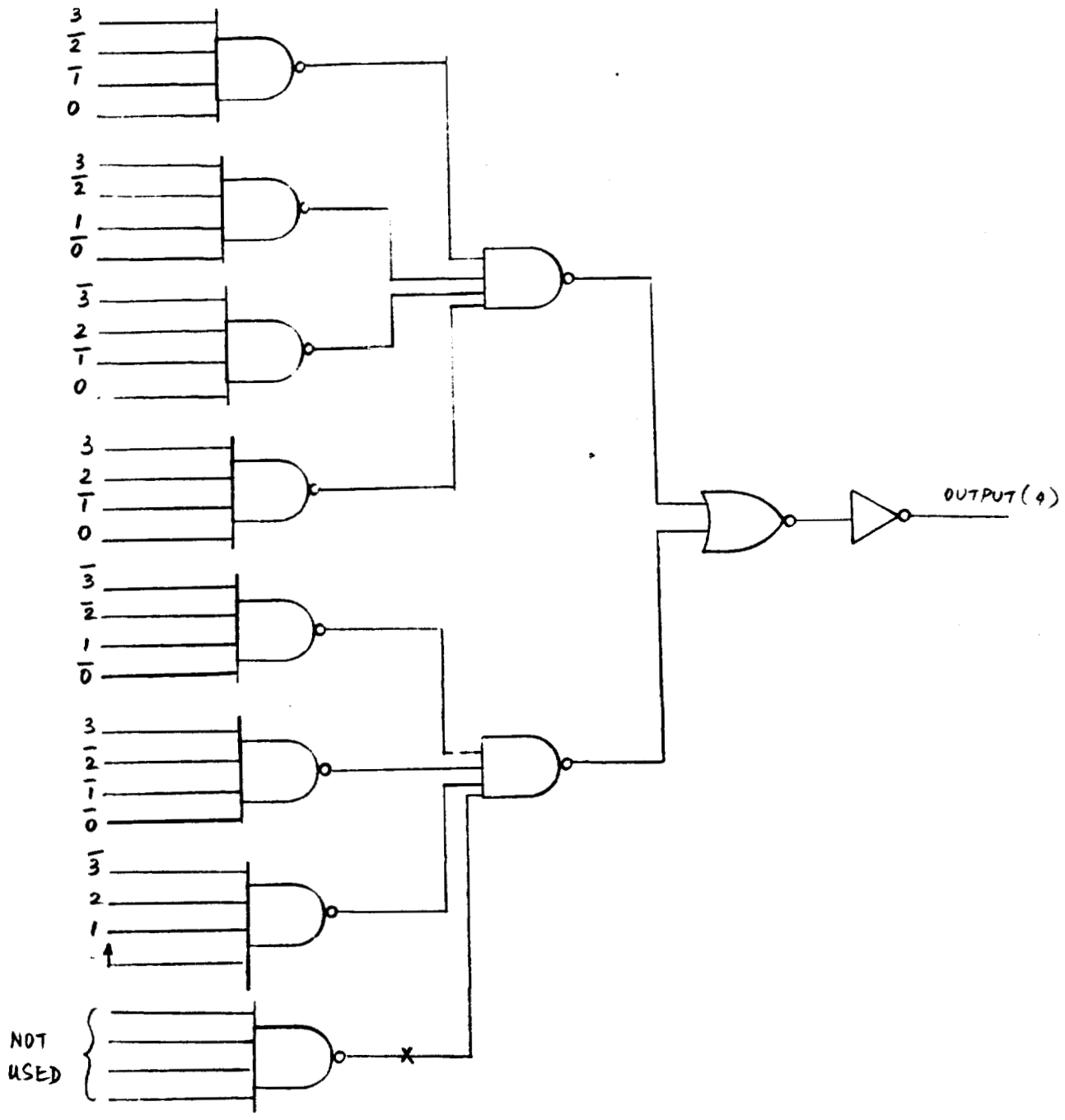
272 X1273

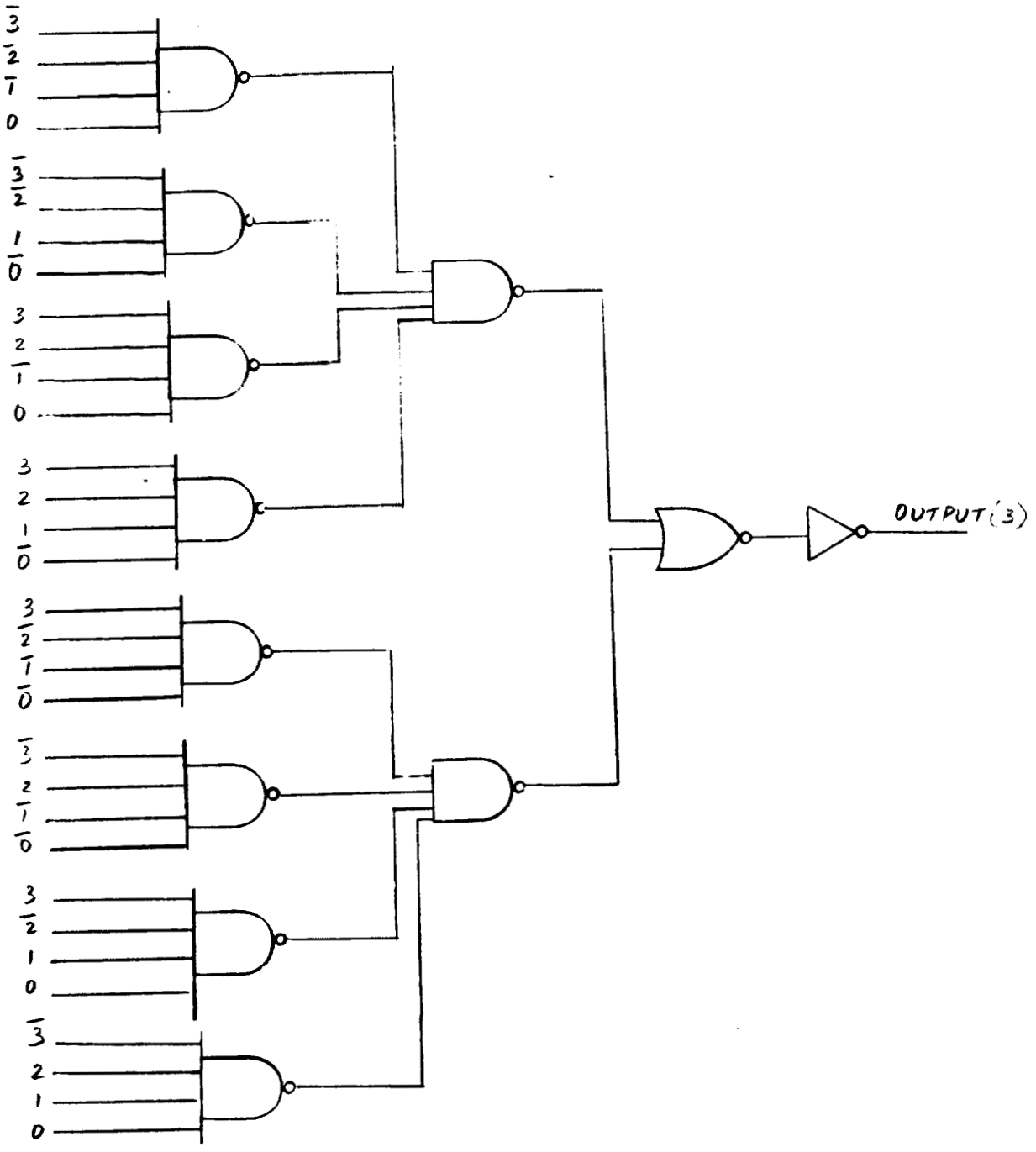
3 = mod 10 (3)
2 = mod 10 (2)
1 = mod 10 (1)
0 = mod 10 (0)

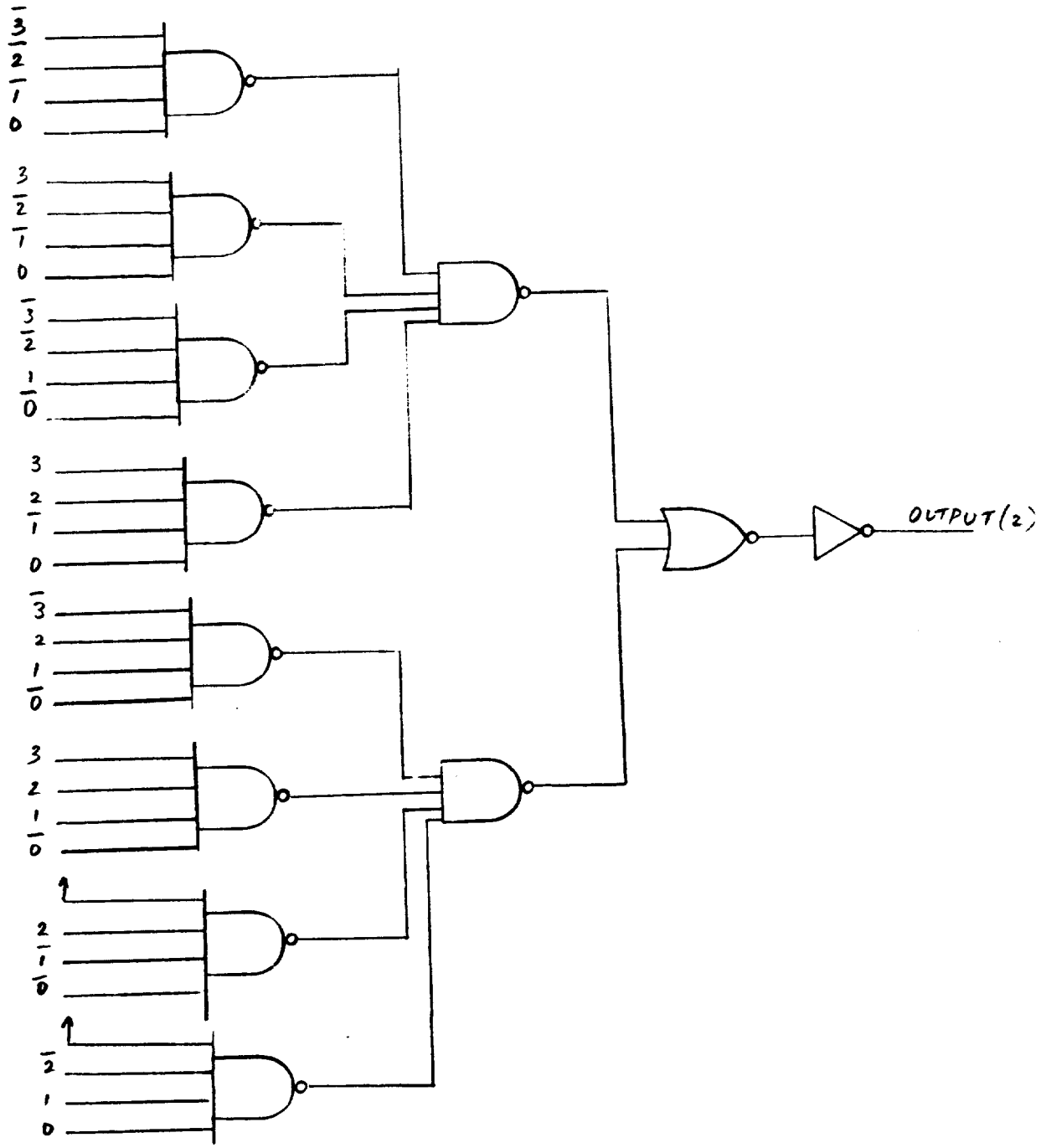


CELL2

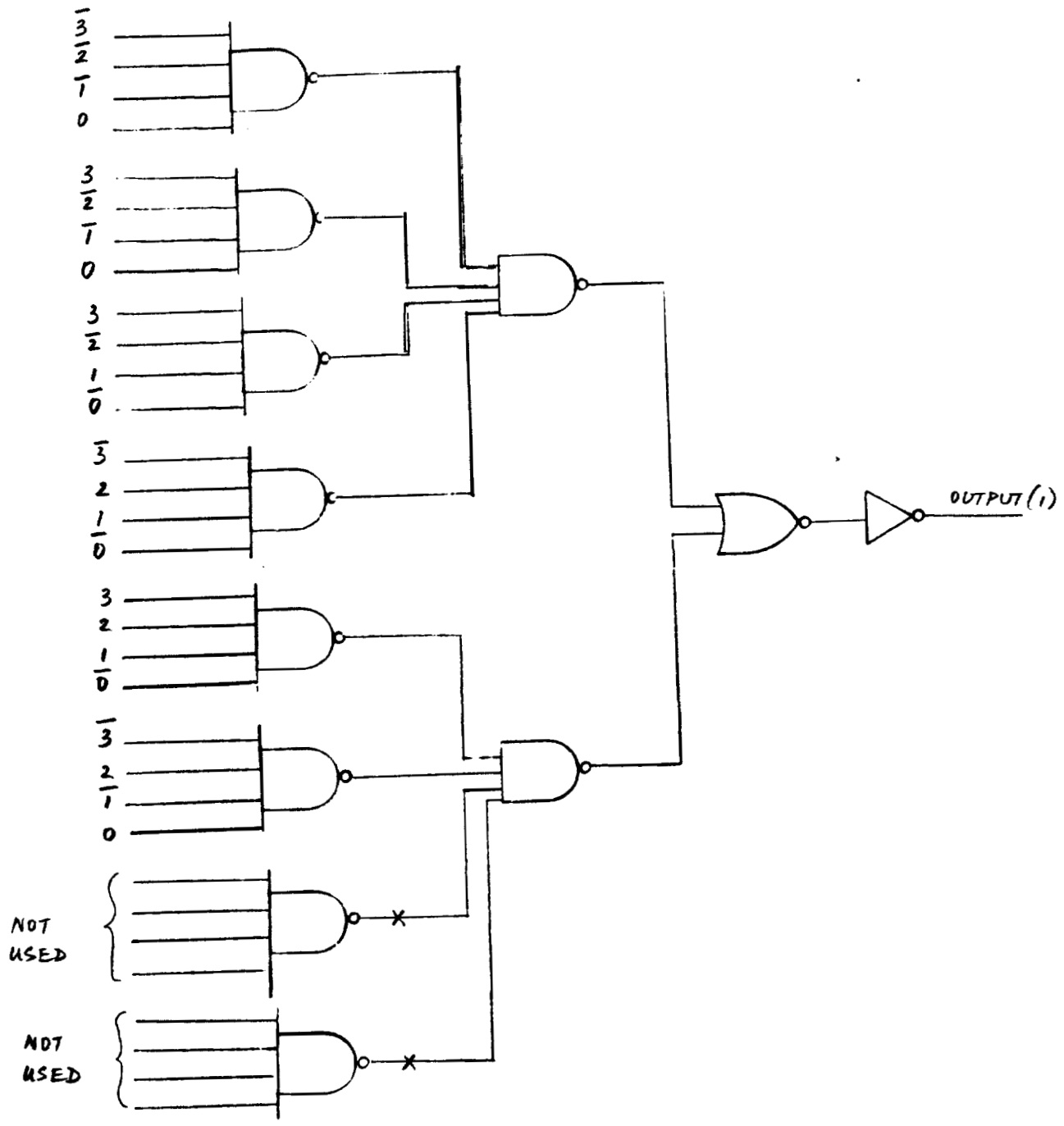
SEQMOD - TABLE PAGE2



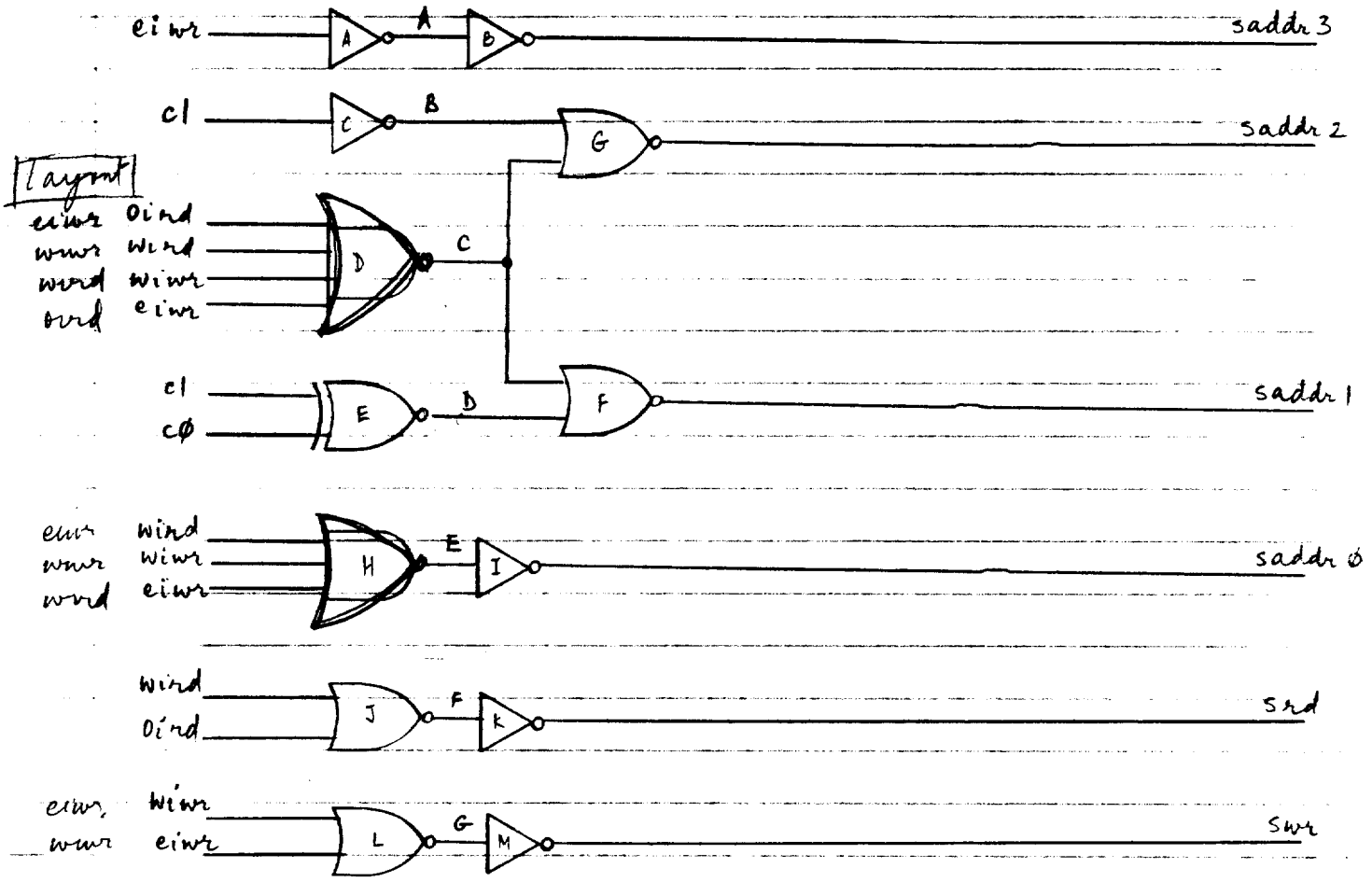

3̄
2̄
1̄
0
3̄
2̄
1
0̄
3
2
1̄
0
3
2
1
0̄
3
2̄
1̄
0̄
3̄
2
1̄
0̄
3
2̄
1
0
3̄
2
1
0
OUTPUT (3)


3
2
1
0
OUTPUT(2)

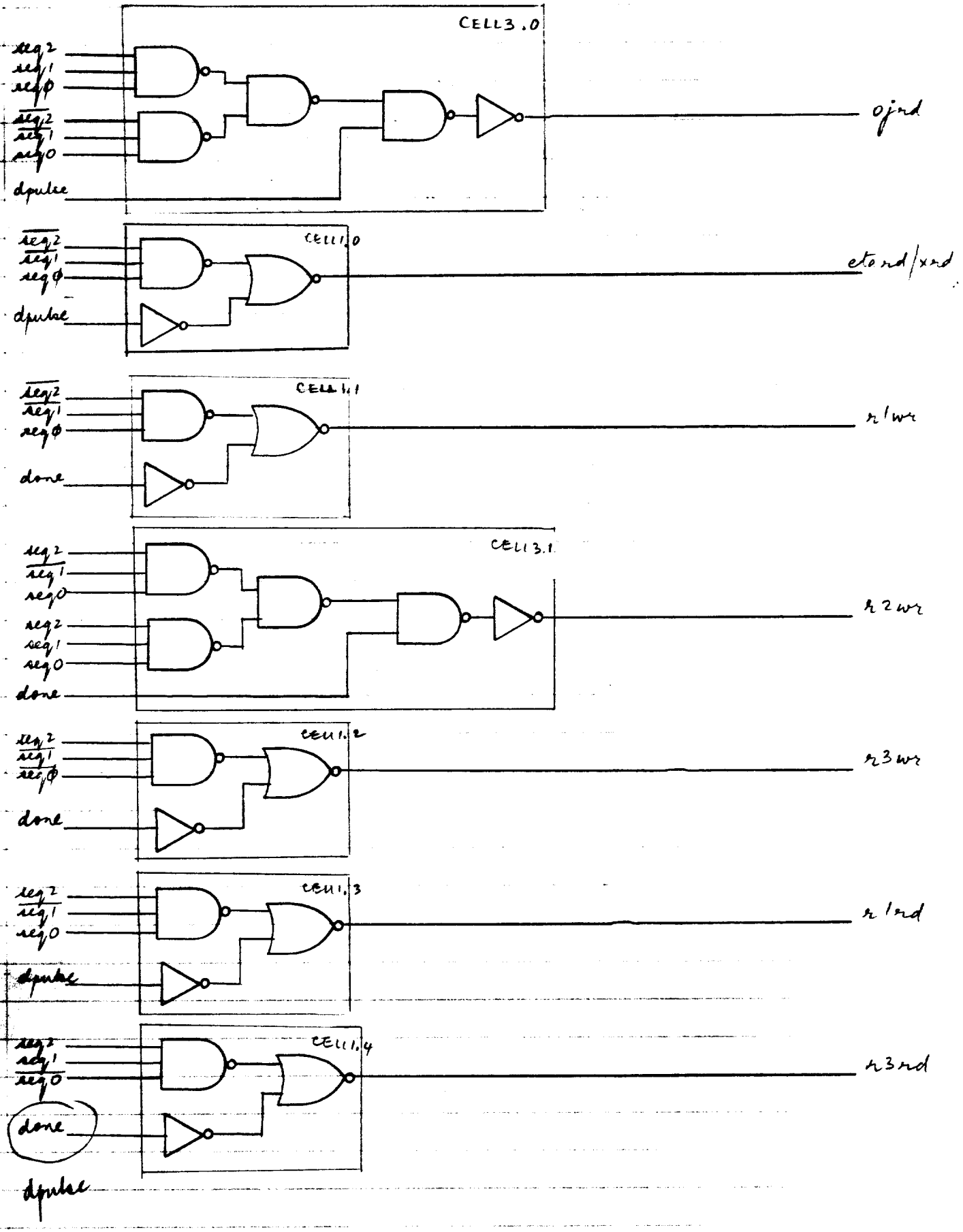
SEG MOD – TABLE – PAGE 5

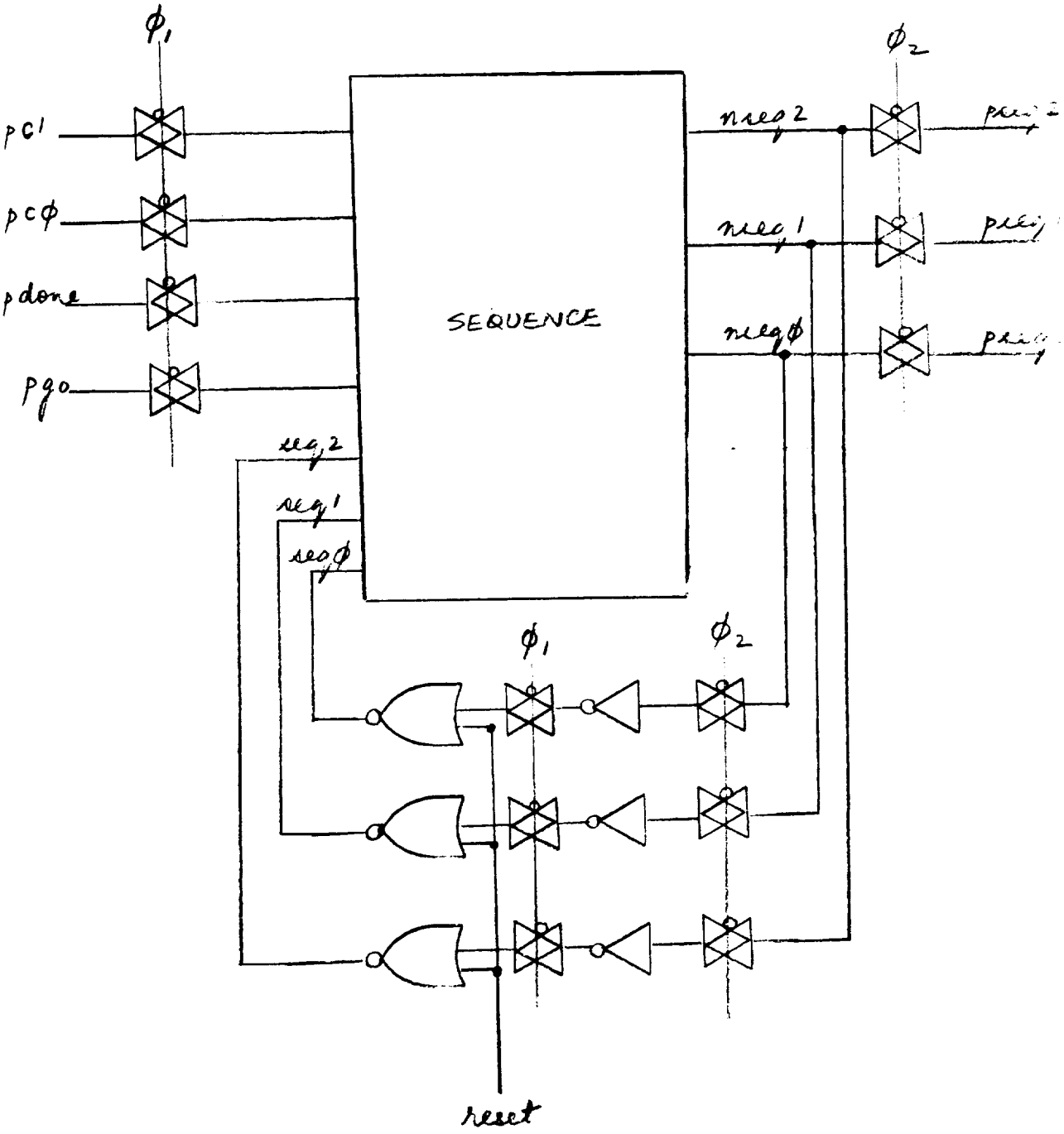


SEQMOD ENCODE PAGE 6

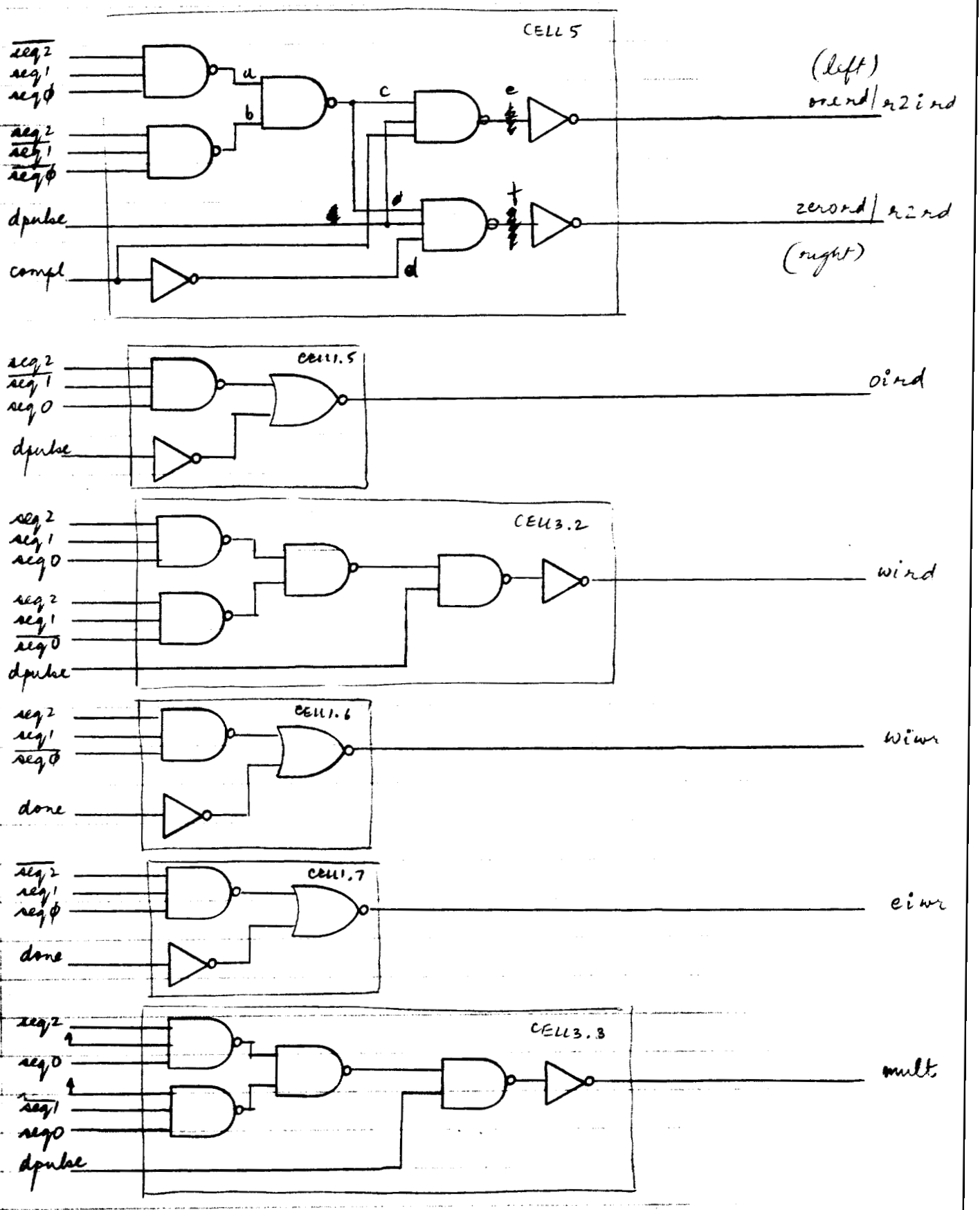


SEQMOD CONTROL PAGE 7

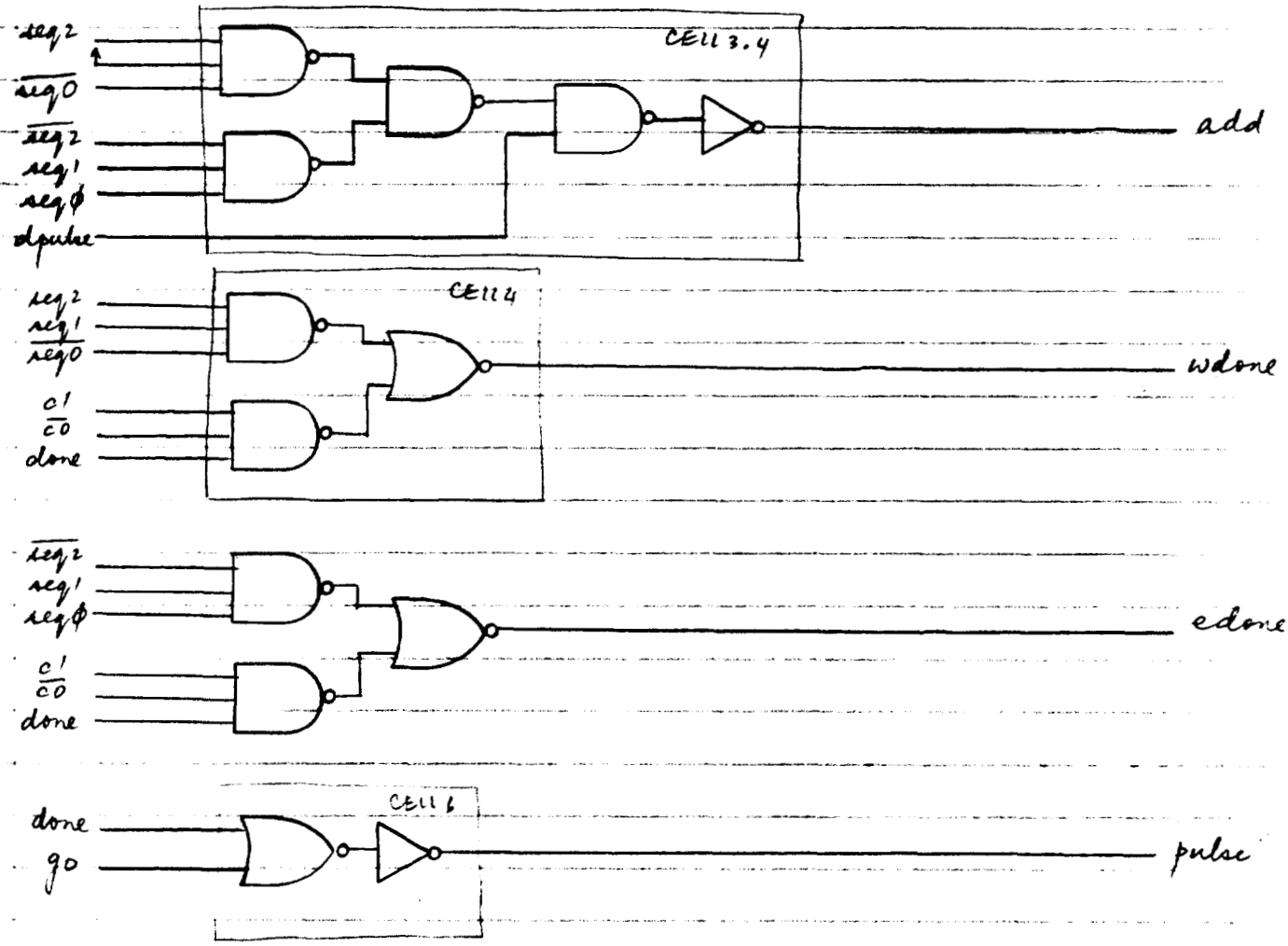


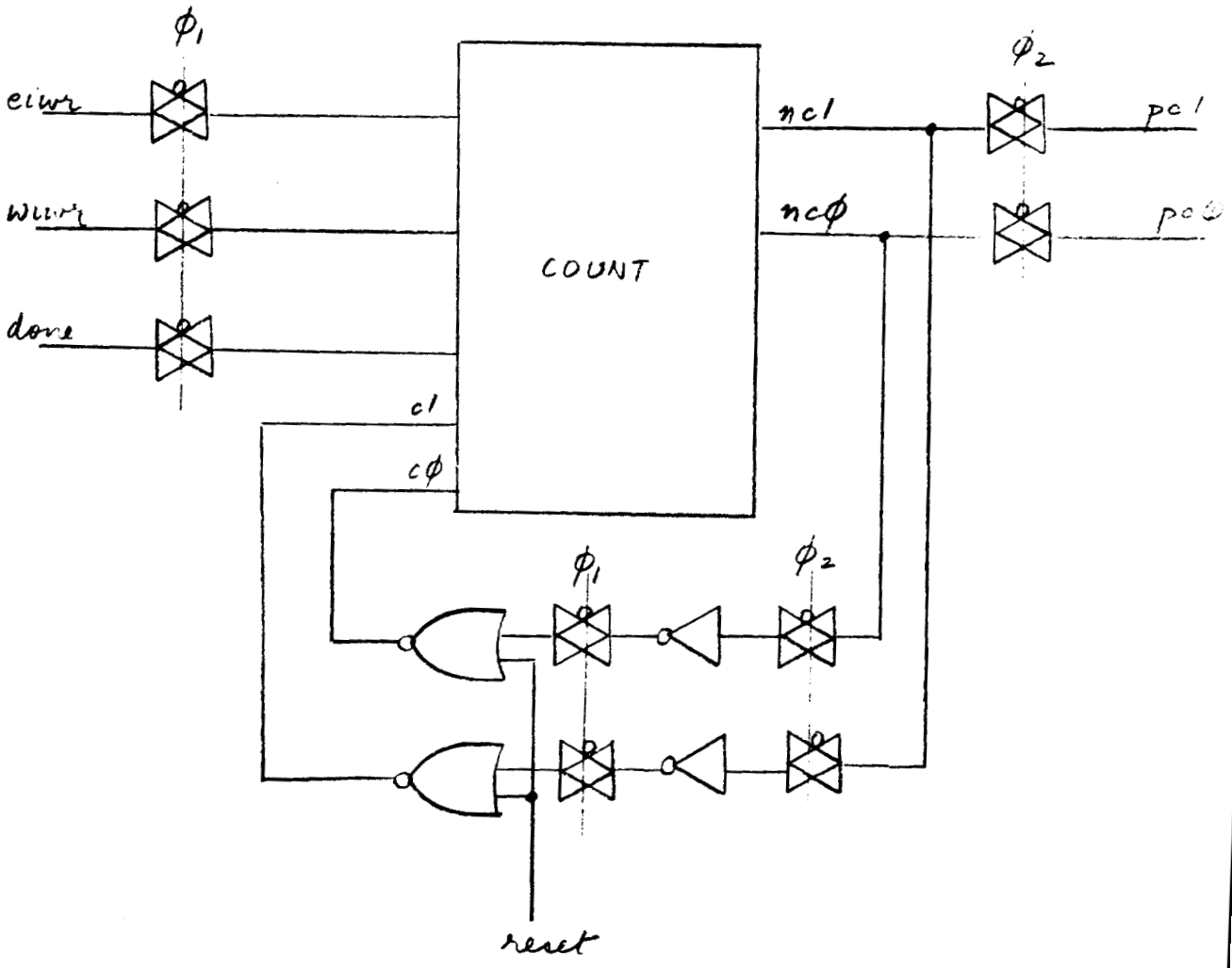

ϕ_1
ϕ_2
pc1
pcφ
pdone
pgo
SEQUENCE
nreq2
nreq1
reqφ
req2
req1
seqφ
reset

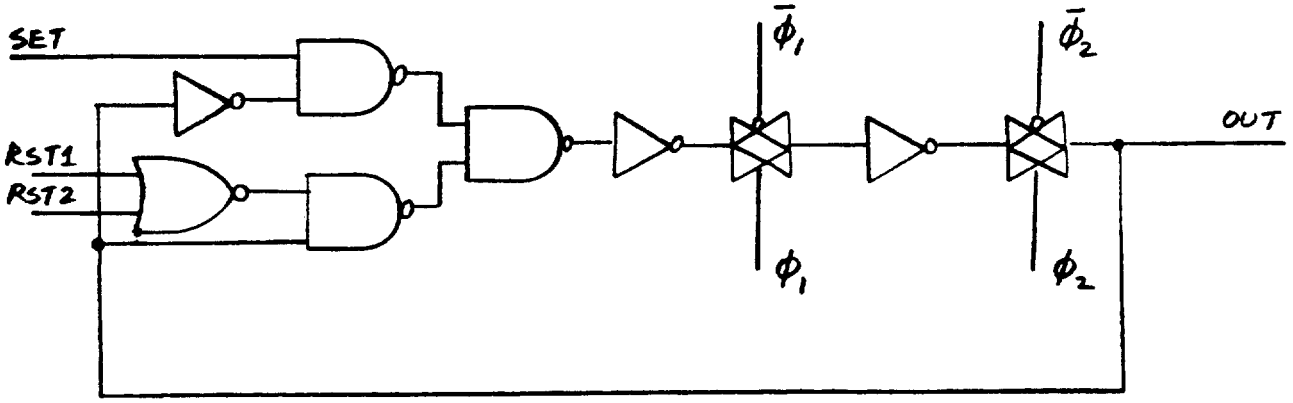
SEQMOD CONTROL PAGE 8

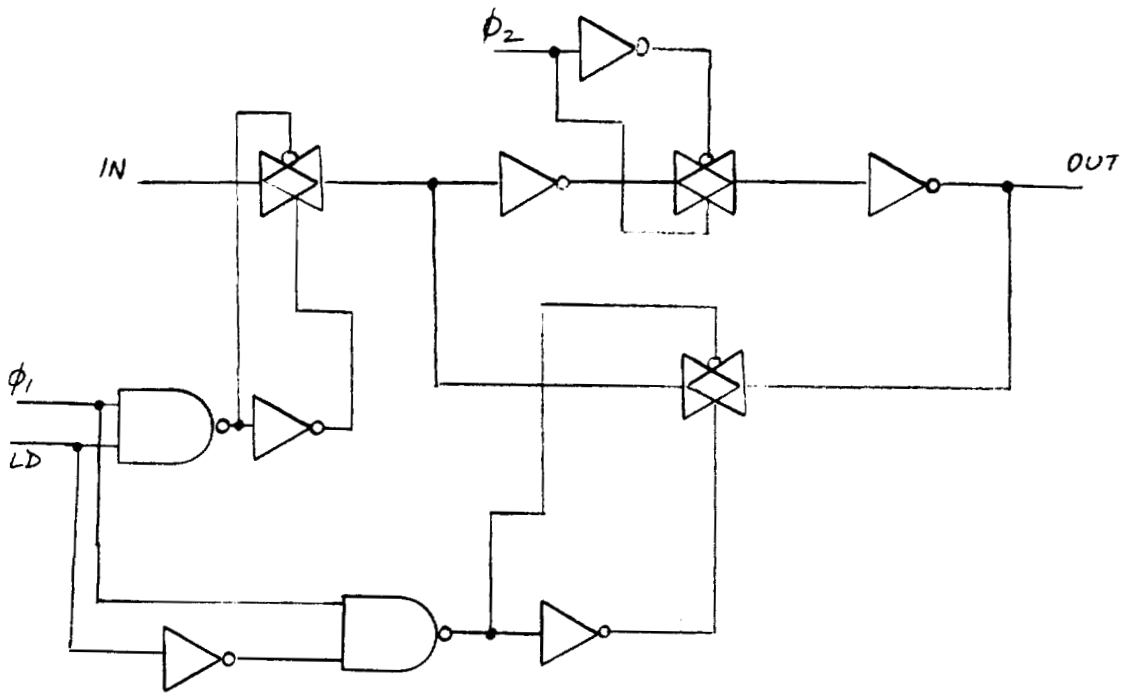


SEQMOD CONTROL PAGE9

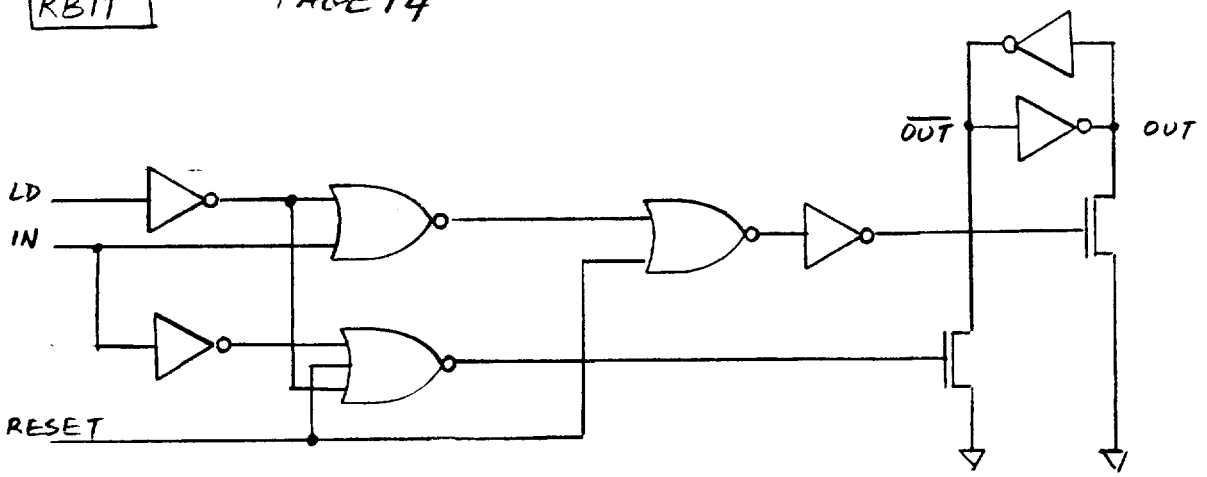



ϕ_1
ϕ_2
eiwr
wiwr
done
COUNT
ncl
ncφ
pcl
pcφ
cl
cφ
ϕ_1
ϕ_2
reset

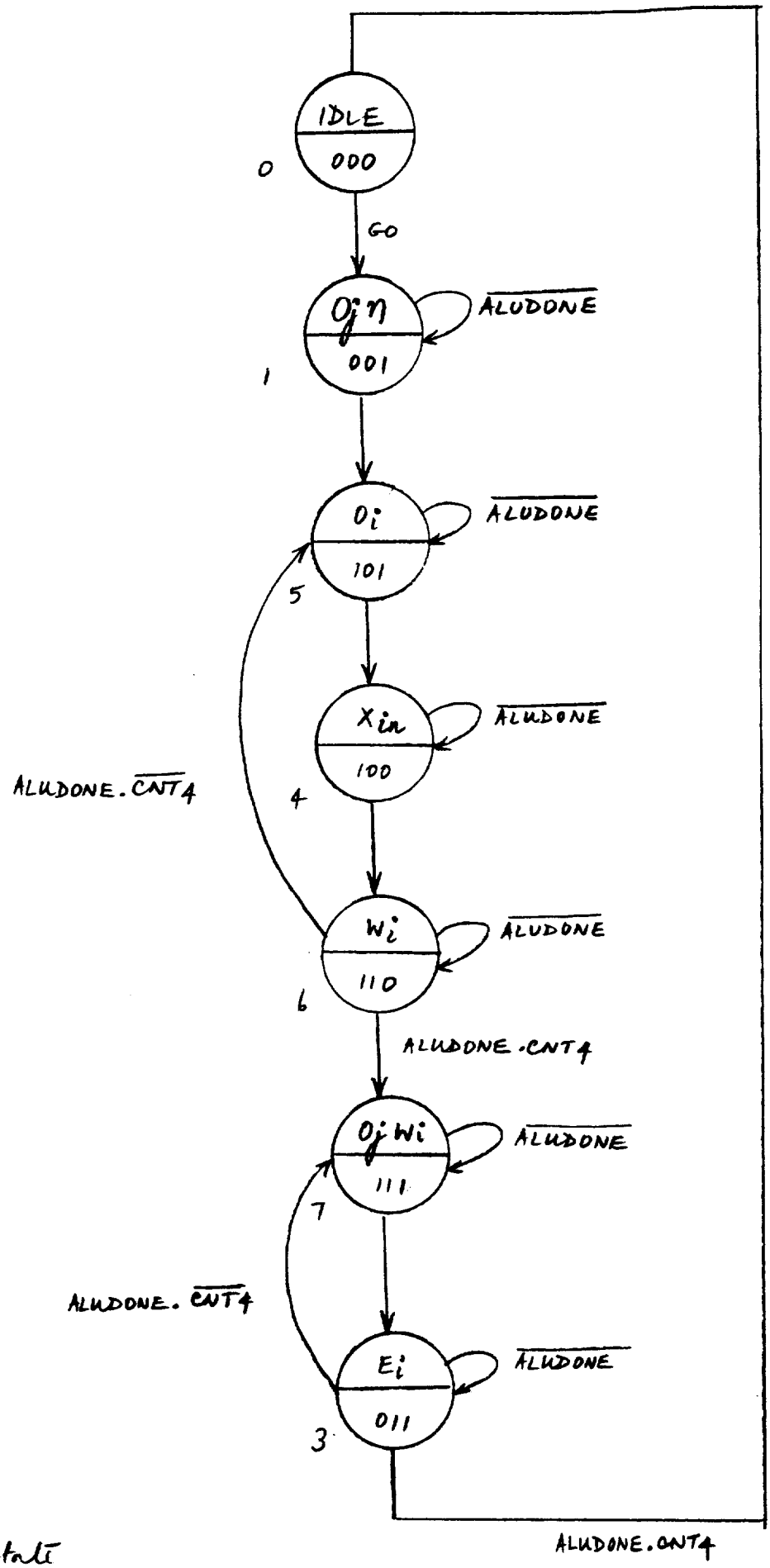

SET
RST1
RST2
$\bar{\phi}_1$
$\bar{\phi}_2$
ϕ_1
ϕ_2
OUT


ϕ_2
IN
OUT
ϕ_1
LD

RBIT



SEQUENCER



Define 8th state

CONTROL

$O_jRD = O_j\eta + O_jW_i$

$\eta RD = O_j\eta$

$XRD = O_j\eta + O_jW_i$

$R1WR = O_j\eta.DONE$

$R2WR = O_i.DONE +$

$R3WR = X_i.DONE$

$R1RD = O_i$

$R2RD = X_i.\overline{COMPL} + E_i.\overline{COMPL}$

$R2IPD = X_i.COMPL + E_i.COMPL$

$R3RD = W_i$

$ONERD = X_i.COMPL + E_i.COMPL$

$ZERORD = X_i.\overline{COMPL} + E_i.\overline{COMPL}$

$O1RD = O_i.CNT1$

$O2RD = O_i.CNT2$

$O3RD = O_i.CNT3$

$O4RD = O_i.CNT4$

$W1RD = W_i.CNT1 + O_jW_i.CNT1$

$W2RD = W_i.CNT2 + O_jW_i.CNT2$

$W3RD = W_i.CNT3 + O_jW_i.CNT3$

$W4RD = W_i.CNT4 + O_jW_i.CNT4$

$W1WR = W_i.CNT1.DONE +$

$W2WR = W_i.CNT2.DONE$

$W3WR = W_i.CNT3.DONE$

$W4WR = W_i.CNT4.DONE$

$E1WR = E_i.CNT1.DONE$

$E2WR = E_i.CNT2\ DONE$

$E3WR = E_i.CNT3.DONE$

$E4WR = E_i.CNT4.DONE$

$MULT = (O_j\eta + O_i + O_jW_i).PULSE$

$ADD = (X_i + W_i + E_i).PULSE$

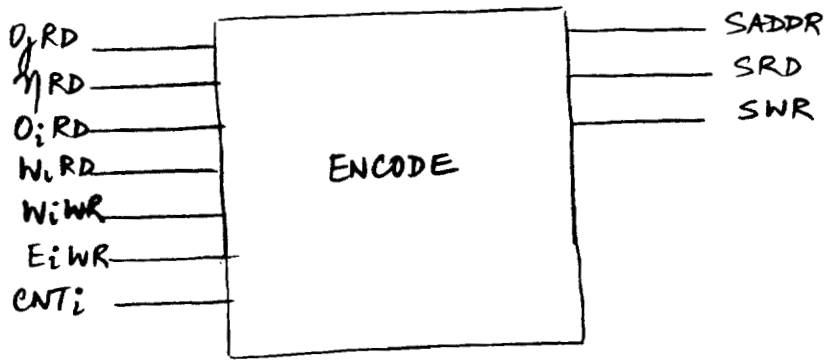
$PULSE = (GO + DONE + WDONE)$ Delayed 1 clock

$WDONE = W_i.CNT4.DONE$

$EDONE = E_i.CNT4.DONE$

ENCODE

O_1	0000
W_1	0001
O_2	0010
W_2	0011
O_3	0100
W_3	0101
O_4	0110
W_4	0111
WA	1000
η	1001
O_j	1010
X_{in}	1011
E_1	1100
E_2	1101
E_3	1110
E_4	1111



E_1 O_j E_2 η E_3 X_{in} E_4

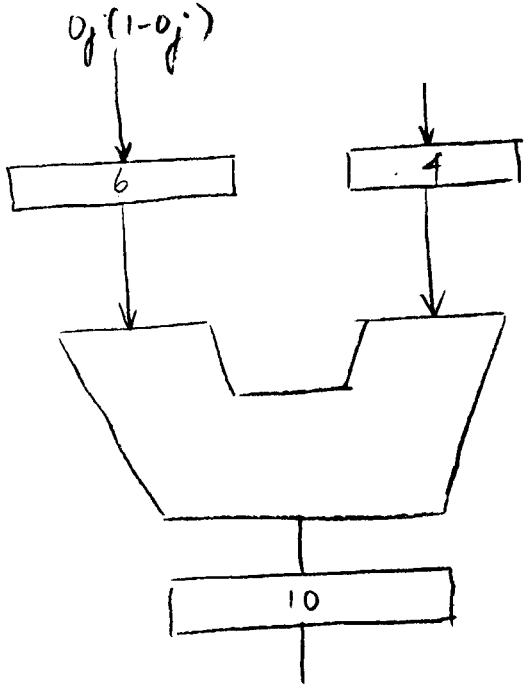
New

CNT4-1	OjRD	ηRD	OiRD	WiRD	WiWR	EiWR	SADDR	SRD	SWR
X	1	X	X	X	X	X	1010	1	0
X	X	1	X	X	X	X	1001	1	0
1	X	X	1	X	X	X	0000	1	0
2	X	X	1	X	X	X	0010	1	0
3	X	X	1	X	X	X	0100	1	0
4	X	X	1	X	X	X	0110	1	0
1	X	X	X	1	X	X	0001	1	0
2	X	X	X	1	X	X	0011	1	0
3	X	X	X	1	X	X	0101	1	0
4	X	X	X	1	X	X	0111	1	0
1	X	X	X	X	X	1	1000	0	1
2	X	X	X	X	X	1	1001	0	1
3	X	X	X	X	X	1	1010	0	1
4	X	X	X	X	X	1	1011	0	1
1	X	X	X	X	1	X	0001	1	0
2	X	X	X	X	1	X	0001	1	0
3	X	X	X	X	1	X	0101	1	0

$O_j(1-O_j)$

1111 × 0000	=	000000	0	
1110 × 0001	=	001110	3	
1101 × 0010	=	011100	7	
1100 × 0011	=	100100	9	
1011 × 0100	=	101100	11	
1010 × 0101	=	110010	12	1110
1001 × 0110	=	110110	13	1110
1000 × 0111	=	111000	14	1111
0111 × 1000	=	111000	14	1111
0110 × 1001	=	110110	13	1110
0101 × 1010	=	110010	12	1110
0100 × 1011	=	101100	11	1101
0011 × 1100	=	100100	9	1100
0010 × 1101	=	011100	7	1011
0001 × 1110	=	001110	3	1001
0000 × 1111	=	000000	0	1000

LOOKUP TABLE



$O_j(1-O_j) * \eta * O_i$

2 Multiplies

$\Delta = \overbrace{O_J(1-O_J)}^{LKUP} \eta \cdot (X) \cdot O_I$

$W_N = (W_{OLD} + \Delta)$

(0-1) ↑ ±1 ↑ (0-1) ↑

ALL VALUES EXCEPT X ARE +VE
SINCE $X = \pm 1$ WE CAN USE IT FOR DIRECTION OF CORRECTION TO WOLD

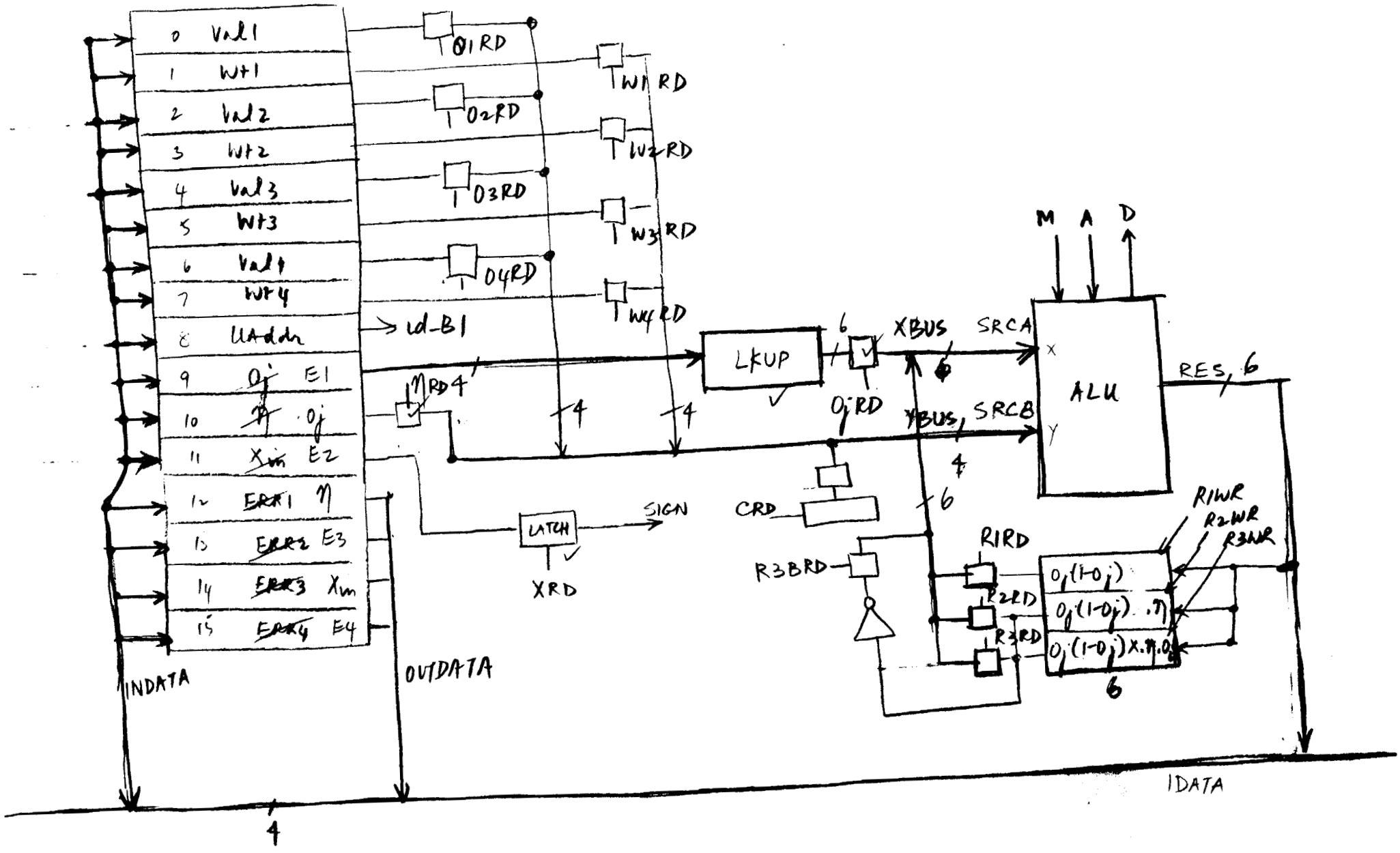
1 Multiply

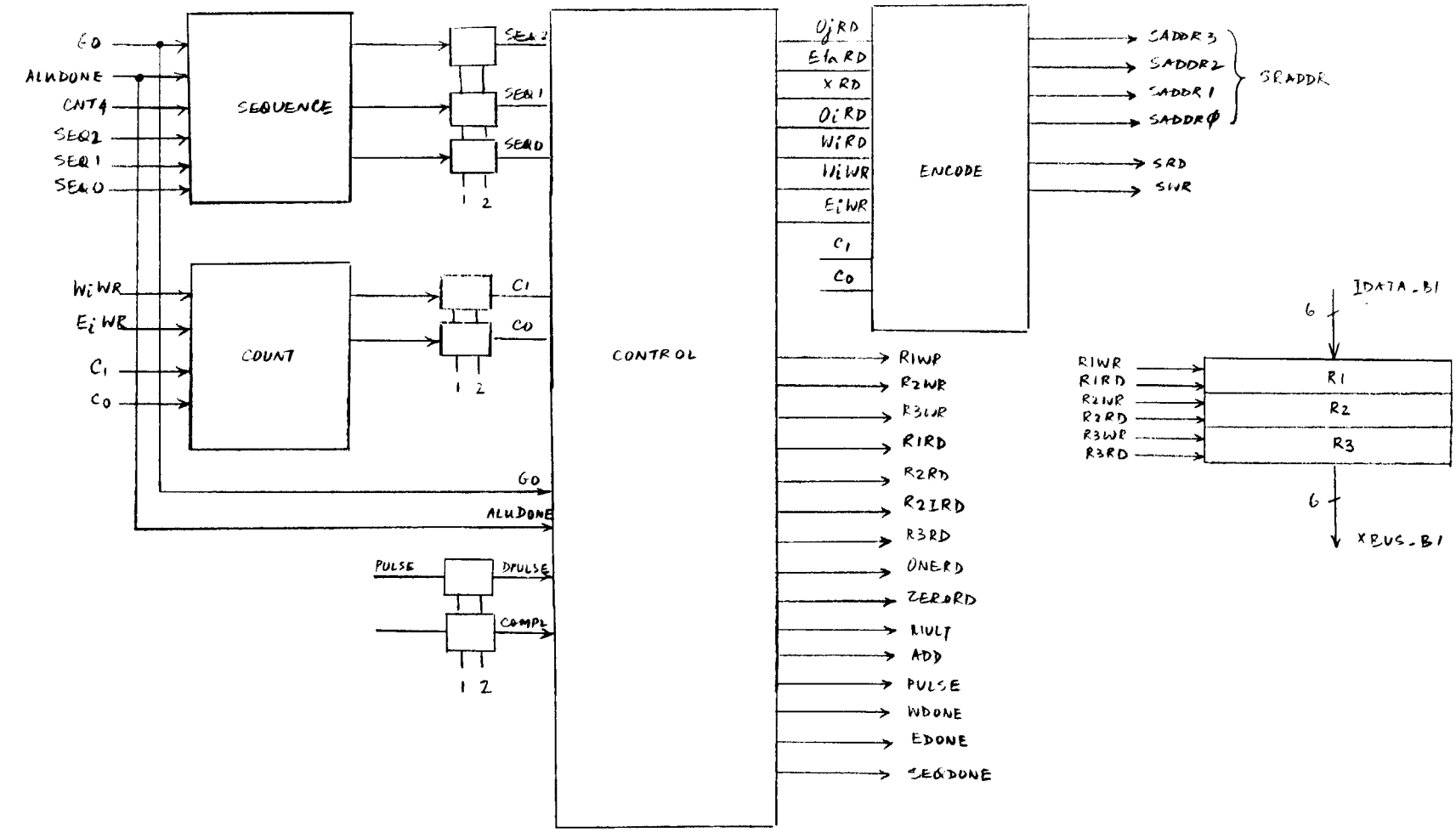
$ERR = O_J(1-O_J) \cdot X \cdot W_I$

ALL VALUES EXCEPT X AND W_I ARE +VE
USE X FOR DIRECTION OF CORRECTION

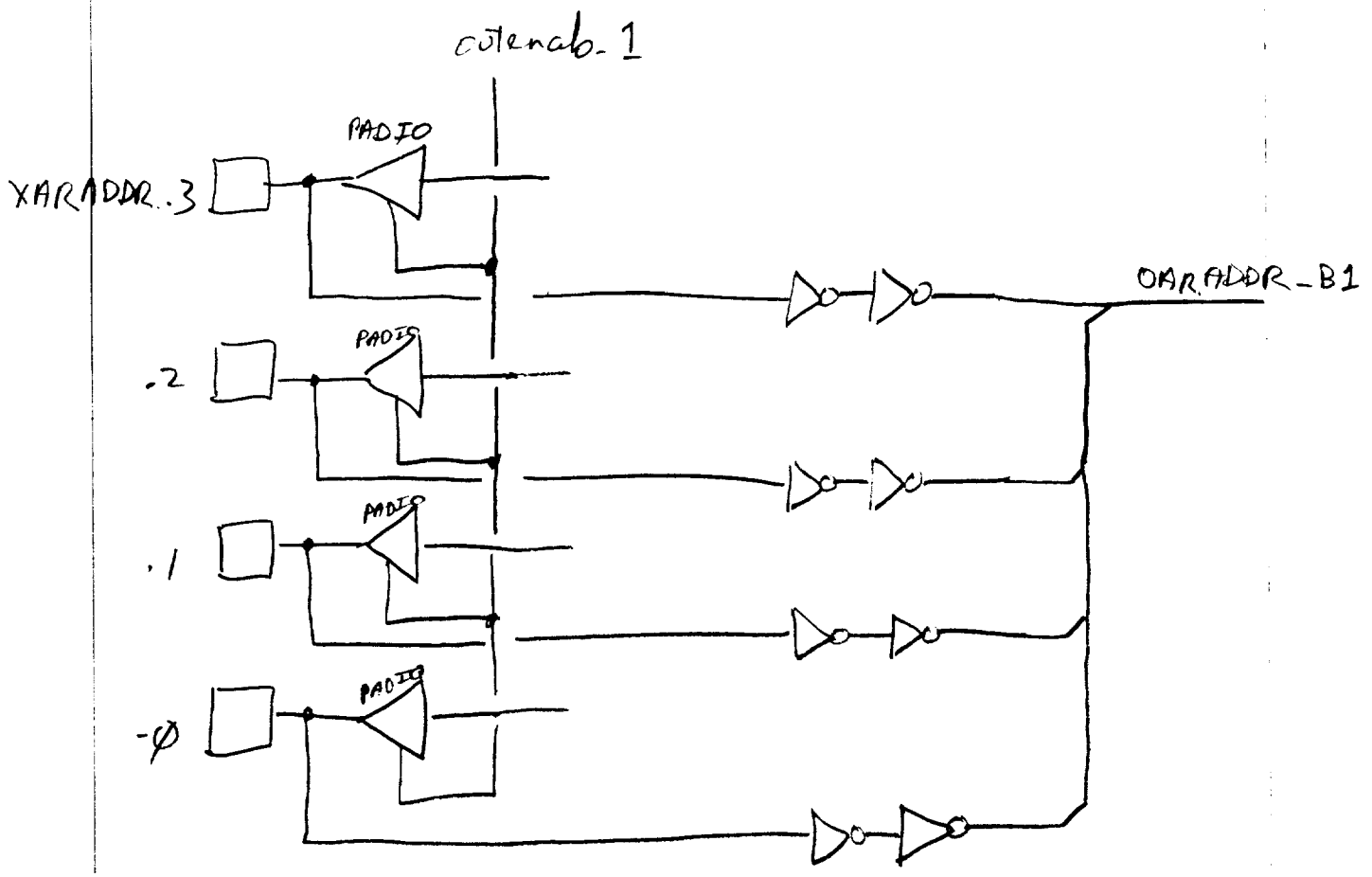
$$W_i = [O_j(1-O_j) \times \eta \cdot O_i + W_i]$$
$$E_i = [O_j \cdot (1-O_j) \times W_i]$$

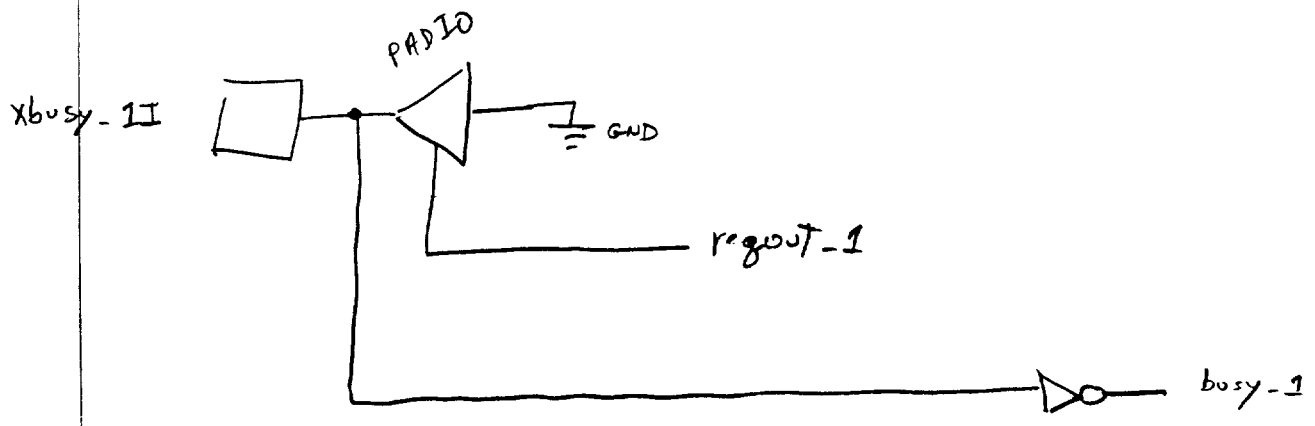
W1-W4 WR
E1-E4 WR



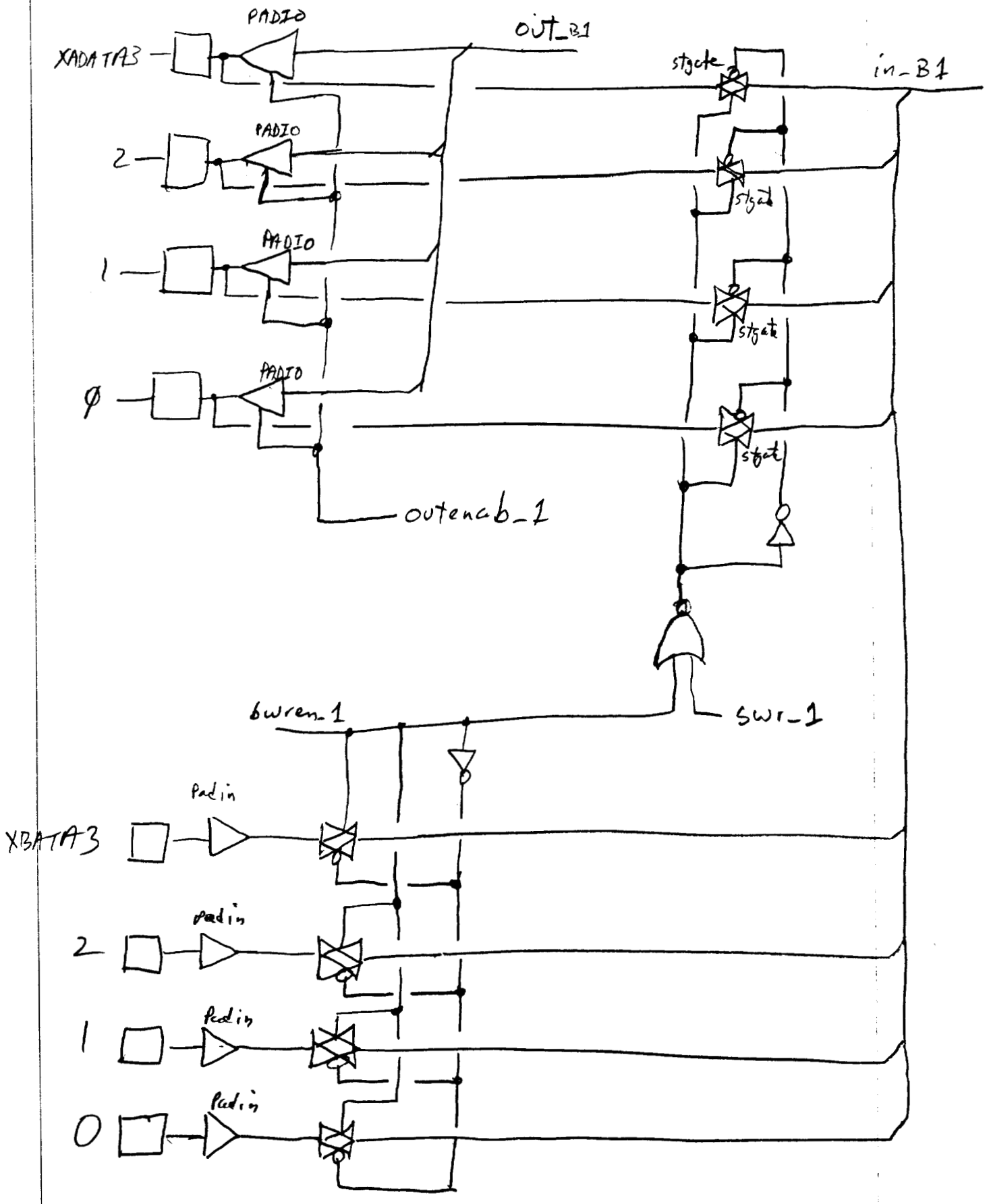

GO
ALUDONE
CNT4
SEQ2
SEQ1
SEQ0
SEQUENCE
SEQ2
SEQ1
SEQ0
1 2
WiWR
EiWR
C1
C0
COUNT
C1
C0
1 2
GO
ALUDONE
PULSE
DPULSE
COMPL
1 2
CONTROL
OjRD
EtaRD
XRD
OiRD
WiRD
WiWR
EiWR
C1
C0
ENCODE
SADDR3
SADDR2
SADDR1
SADDR0
SRADDR
SRD
SIWR
R1WR
R2WR
R3WR
R1RD
R2RD
R2IRD
R3RD
ONERD
ZERORD
MULT
ADD
PULSE
WDONE
EDONE
SEQDONE
IDATA_B1
6
R1
R2
R3
6
XBUS_B1
IDATA_B1
YBUS_B1
4
TABLE
XBUS_B1
XIN_B1
ETA_B1
YBUS_B1

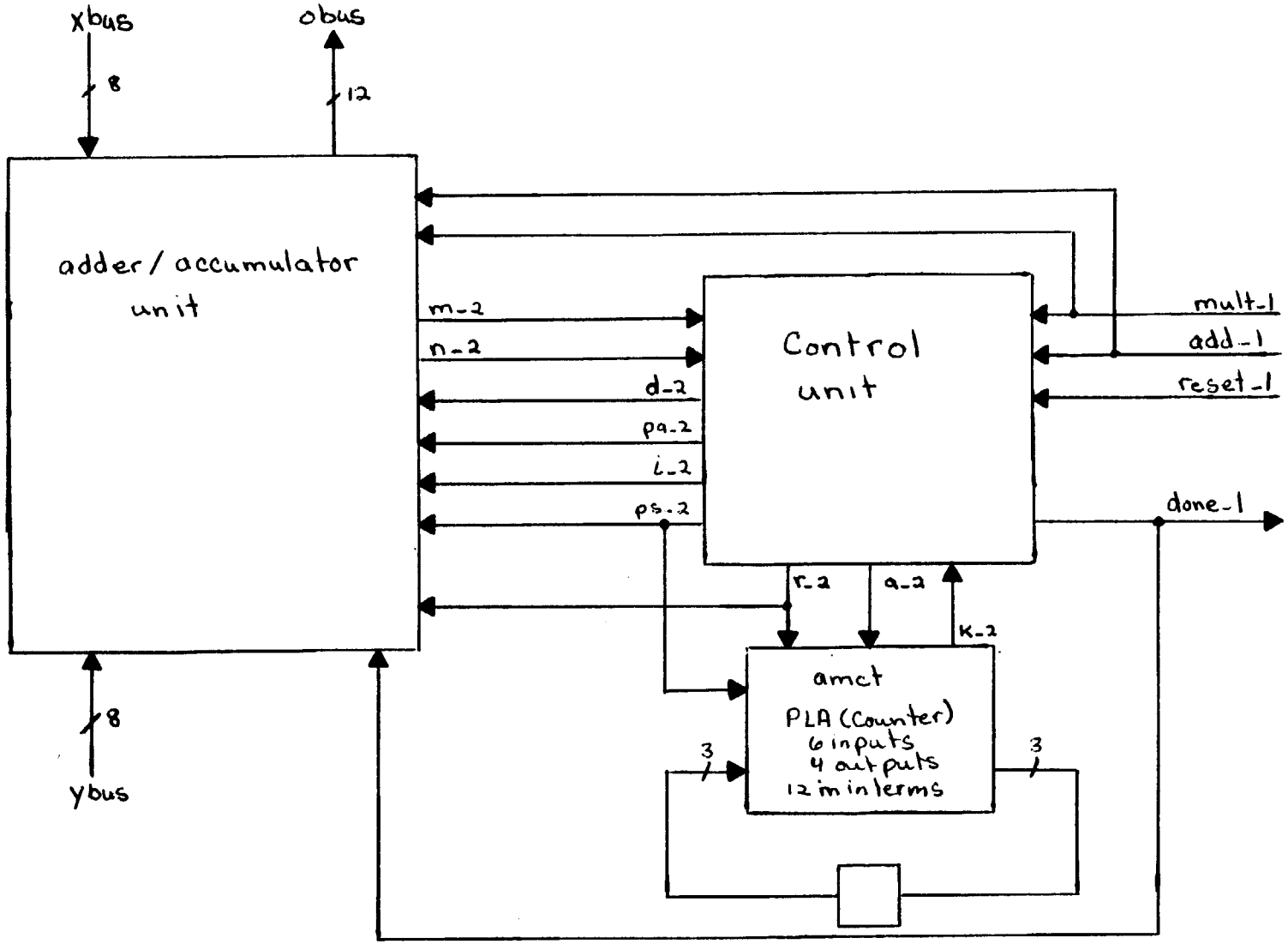
TRISTATE PADS



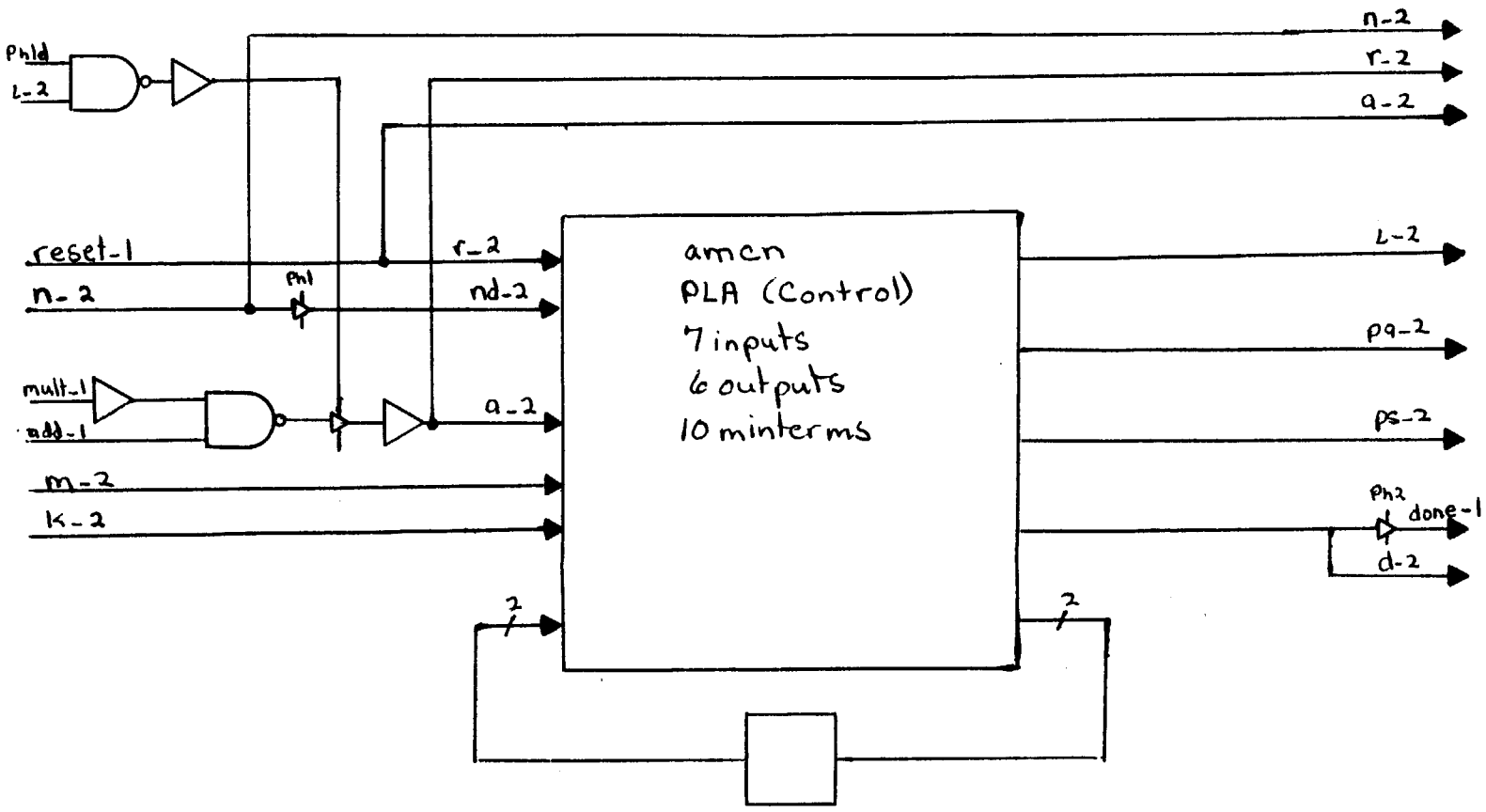


Data Busses

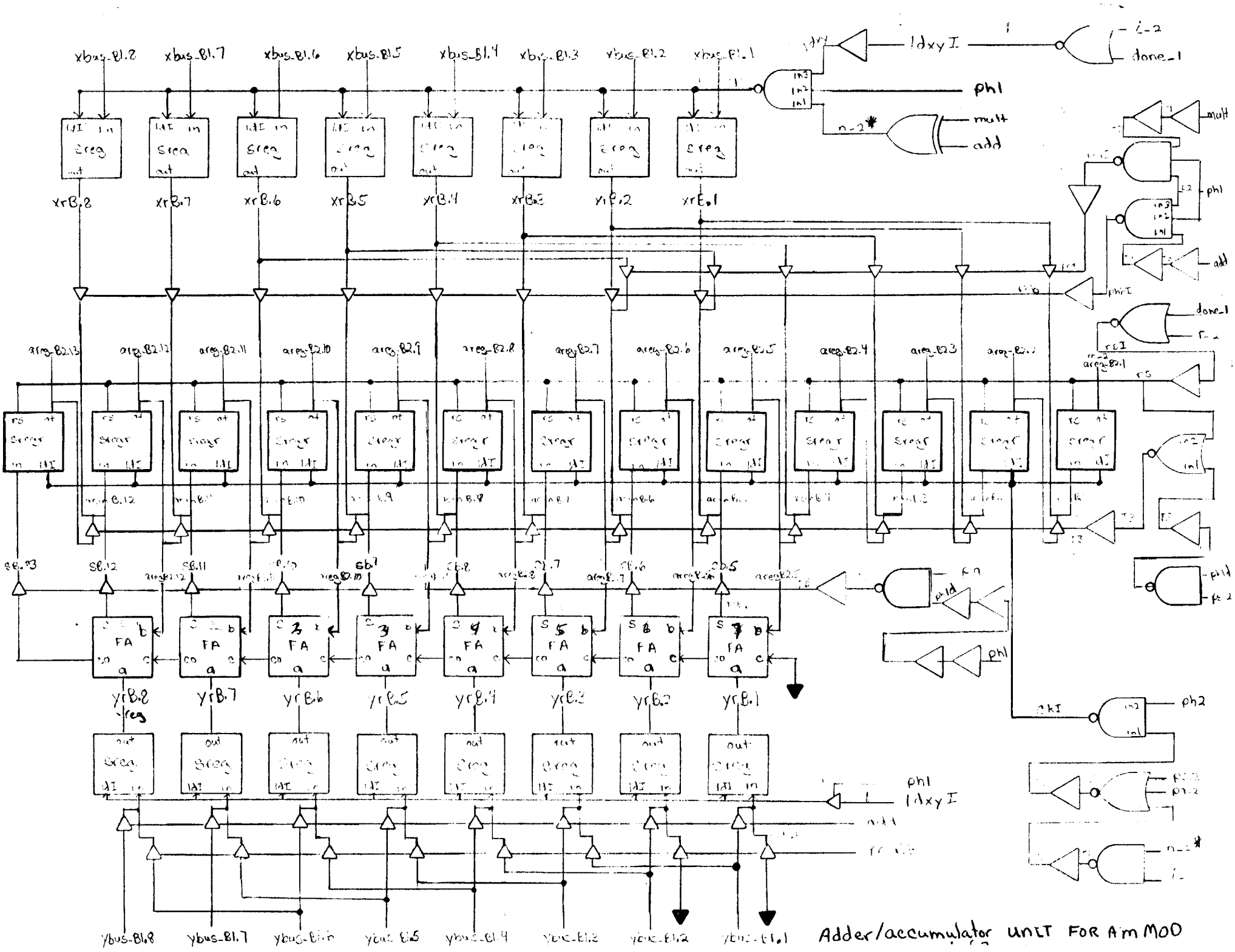

PADIO
XADATA3
OUT_B1
stgate
in_B1
PADIO
2
stgate
PADIO
1
stgate
PADIO
0
stgate
outenab_1
bwren_1
SWR_1
Padin
XBATA3
padin
2
Padin
1
Padin
0

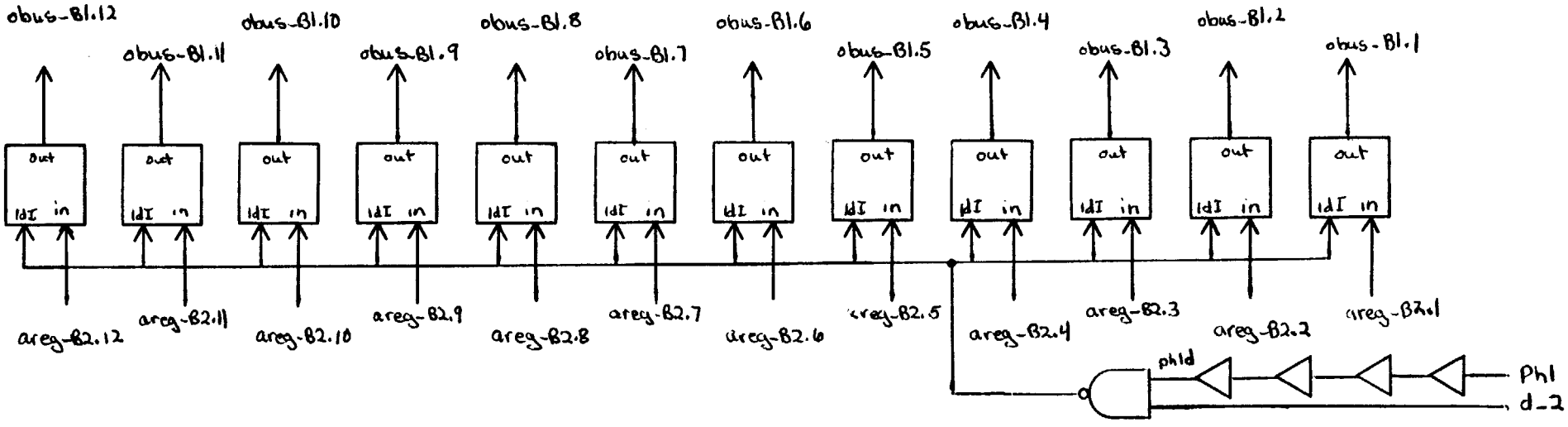

xbus
8
obus
12
adder / accumulator unit
m_2
n_2
d_2
pa_2
i_2
ps_2
Control unit
mult_1
add_1
reset_1
done_1
r_2
a_2
K_2
amct
PLA (Counter)
6 inputs
4 outputs
12 minterms
3
3
8
ybus

Am MOD

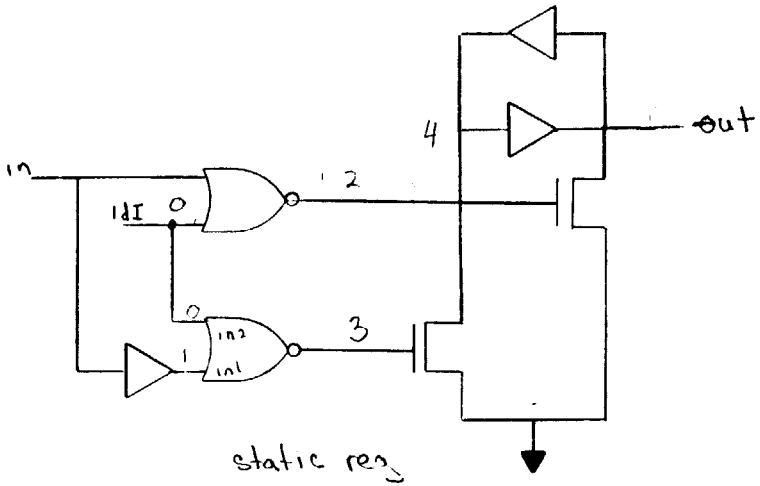


Control unit for AmMOD


xbus_B1.8
xbus_B1.7
xbus_B1.6
xbus_B1.5
xbus_B1.4
xbus_B1.3
xbus_B1.2
xbus_B1.1
xrB.8
xrB.7
xrB.6
xrB.5
xrB.4
xrB.3
xrB.2
xrB.1
ldxyI
phI
mult
add
done_1
FA
yrB.8
yrB.7
yrB.6
yrB.5
yrB.4
yrB.3
yrB.2
yrB.1
ybus_B1.8
ybus_B1.7
ybus_B1.6
ybus_B1.5
ybus_B1.4
ybus_B1.3
ybus_B1.2
ybus_B1.1
ph2
Adder/accumulator UNIT FOR AimMOD

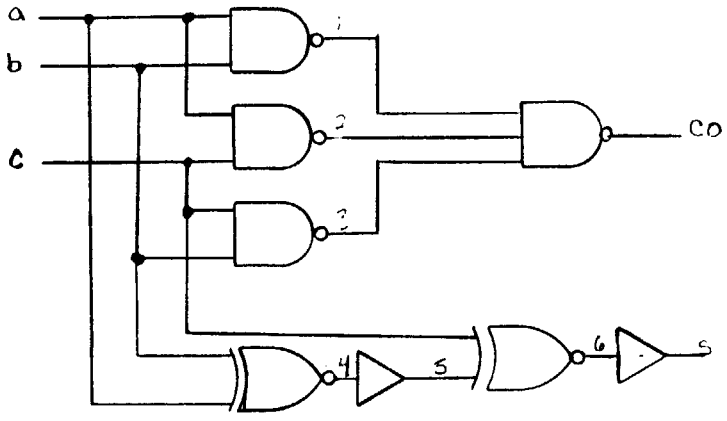


Adder/accumulator unit for Arm MOD 2of2 pages

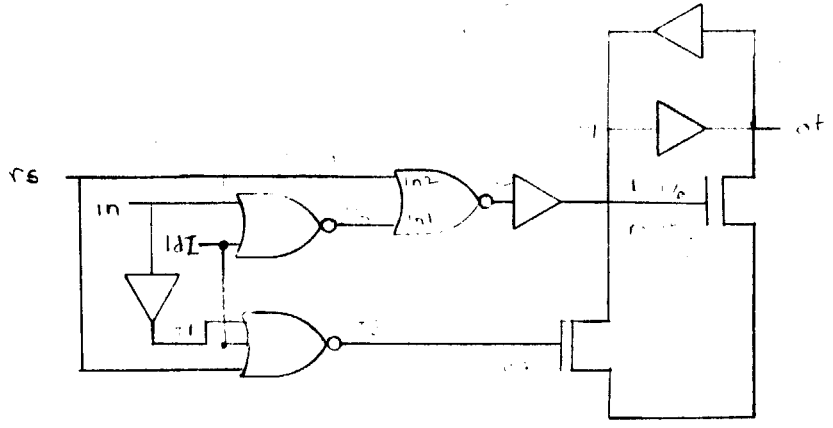


static reg
sreg

1.16



full adder FA



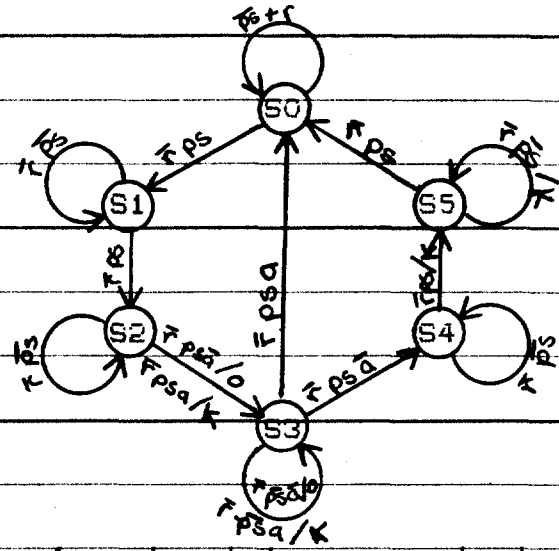
static reg with reset
sregr

subunits for adder/accumulator unit
cells for AM module

AmMOD Counter Unit State Machine

Inputs
ps=shift areg
r=reset
a=add + multbar

Outputs
k=1 at count max



if r=1 go to state zero output zero

count	inputs	next count	outputs
C B A	r ps a	G F E	k
0 0 0	0 0 X	0 0 0	0
0 0 0	0 1 X	0 0 1	0
0 0 1	0 0 X	0 0 1	0
0 0 1	0 1 X	0 1 0	0
0 1 0	0 0 X	0 1 0	0
0 1 0	0 1 0	0 1 1	0
0 1 0	0 1 1	0 1 1	1
0 1 1	0 0 0	0 1 1	0
0 1 1	0 0 1	0 1 1	1
0 1 1	0 1 0	1 0 0	0
0 1 1	0 1 1	0 0 0	0
1 0 0	0 0 X	1 0 0	0
1 0 0	0 1 X	1 0 1	1
1 0 1	0 0 X	1 0 1	1
1 0 1	0 1 X	0 0 0	0
X X X	1 X X	0 0 0	0

ncnt0=E=rbar(ps(Cbar(Bbar)Abar+Cbar(B)Abar+C(Bbar)Abar) +
psbar(Cbar(Bbar)A+Cbar(B)A+C(Bbar)A))

ncnt1=F=rbar(Cbar(Bbar)A(ps) + Cbar(B)Abar + Cbar(B)A(psbar))

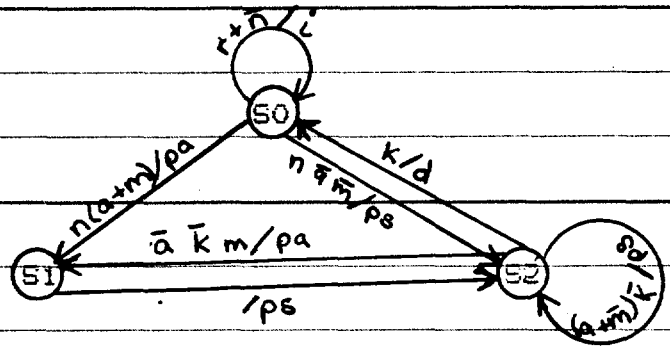
ncnt2=G=rbar(C(Bbar)Abar + C(Bbar)A(psbar) + Cbar(B)A(ps)abar)

k=rbar(a(Cbar(B)Abar(ps)+Cbar(B)A(psbar)) + C(Bbar)(Abar(ps)+A(psbar)))

AmMOD Control Unit State Machine

Inputs
n=mult ⊕ add
a=add + multbar
k=1 at count max
m=lsb areg
r=reset

Outputs
ps=shift areg right 1
pa=add pulse
d=done
i=system idle



if r=1 go to state zero set i=1

current state B A	inputs r n a m k	next state F E	outputs ps pa d i
0 0	0 0 X X X	0 0	0 0 0 1
0 0	0 1 1 X X	0 1	0 1 0 0
0 0	0 1 0 1 X	0 1	0 1 0 0
0 0	0 1 0 0 X	1 0	1 0 0 0
0 1	0 X X X X	1 0	1 0 0 0
1 0	0 X 1 X 0	1 0	1 0 0 0
1 0	0 X 1 X 1	0 0	0 0 1 0
1 0	0 X 0 0 0	1 0	1 0 0 0
1 0	0 X 0 1 0	0 1	0 1 0 0
1 0	0 X 0 X 1	0 0	0 0 1 0
X X	1 X X X X	0 0	0 0 0 1

nxt0=E=rbar(Bbar(Abar)n(a+abar(m)) + B(Abar)abar(m)kbar)

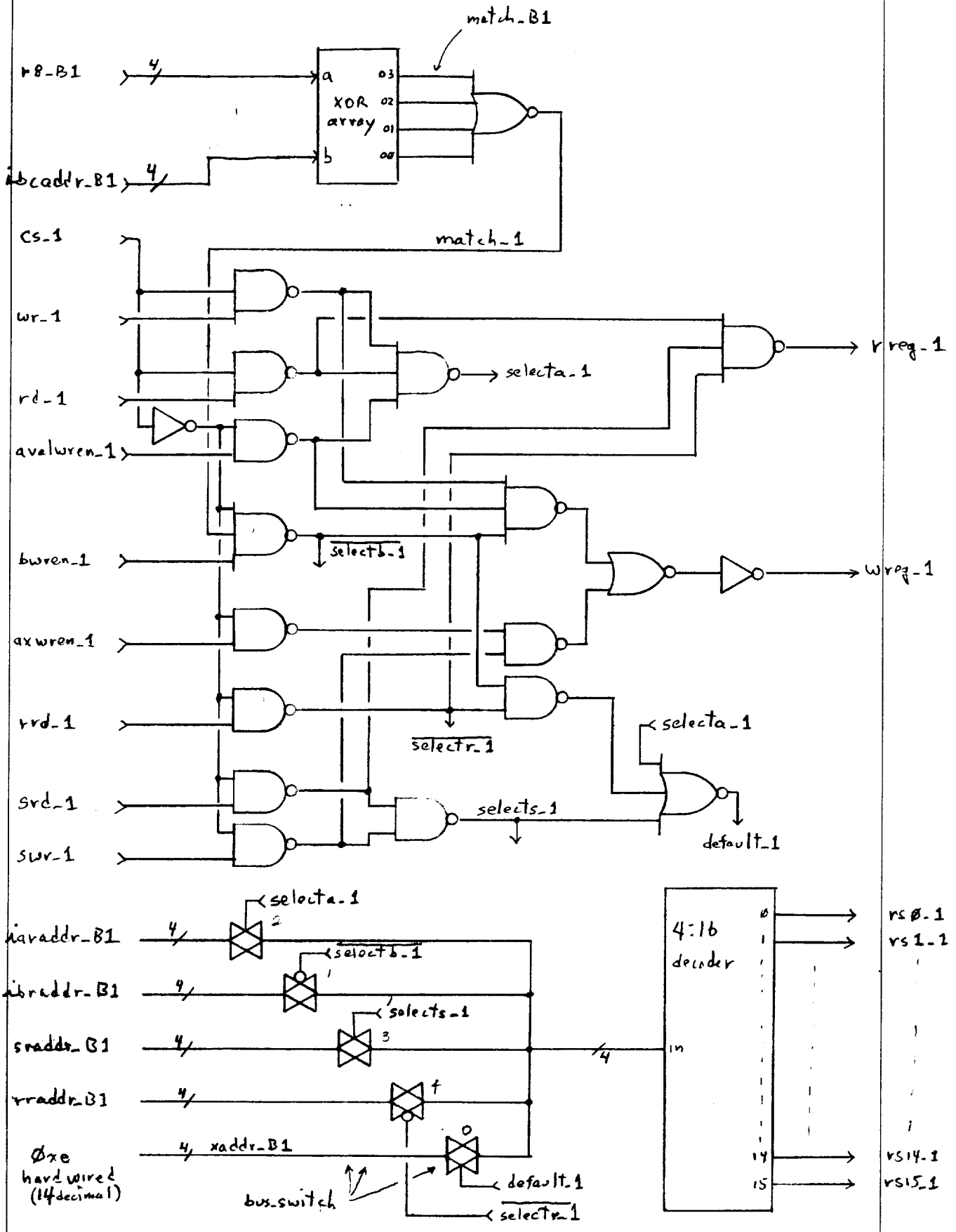
nxt1=F=rbar(Bbar(Abar)n(abar)mbar + Bbar(A) + B(abar)((a+(mbar))kbar))

ps=rbar(Bbar(Abar)n(abar)mbar + Bbar(A) + B(Abar)kbar(a+mbar))

pa=rbar(n(Bbar(Abar)(a+m)) + B(Abar)abar(m)kbar)

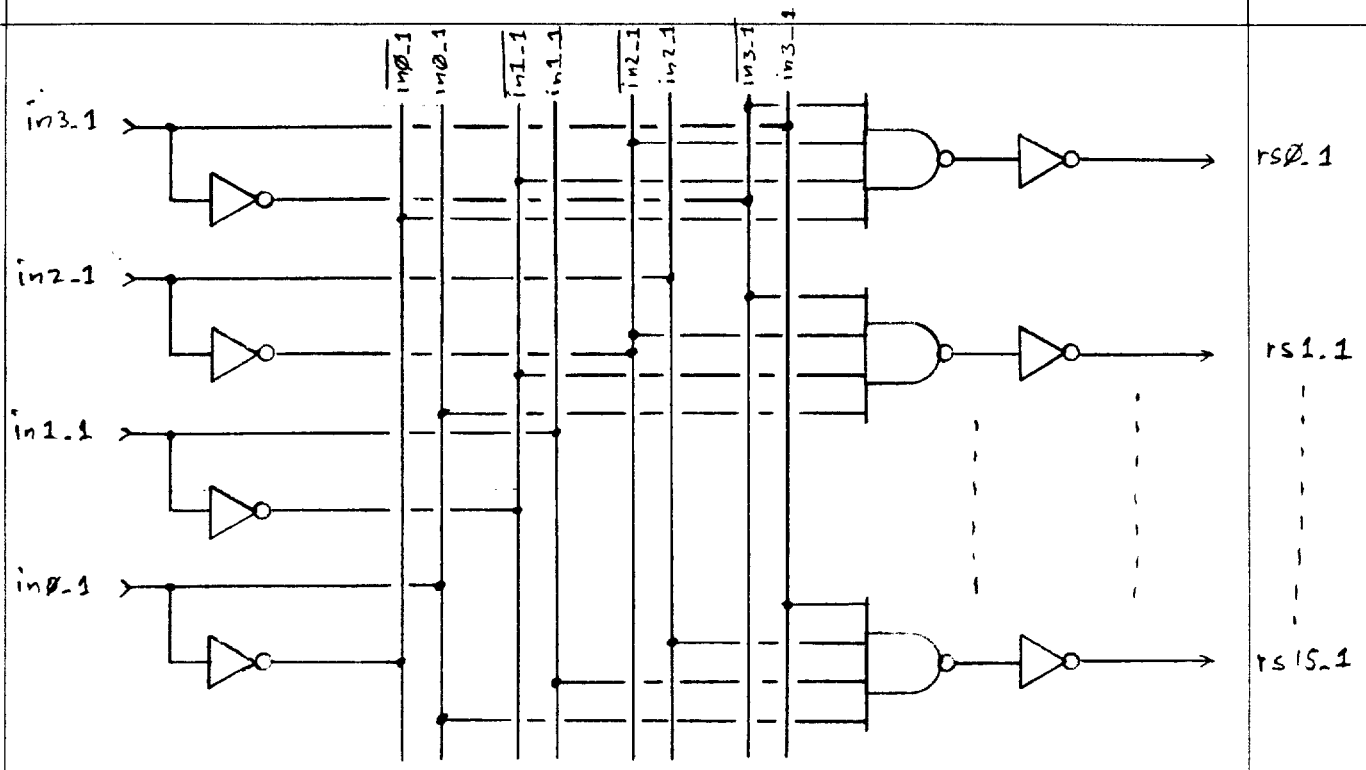
d=rbar(B(Abar)k)

i=r + nbar(Bbar)Abar



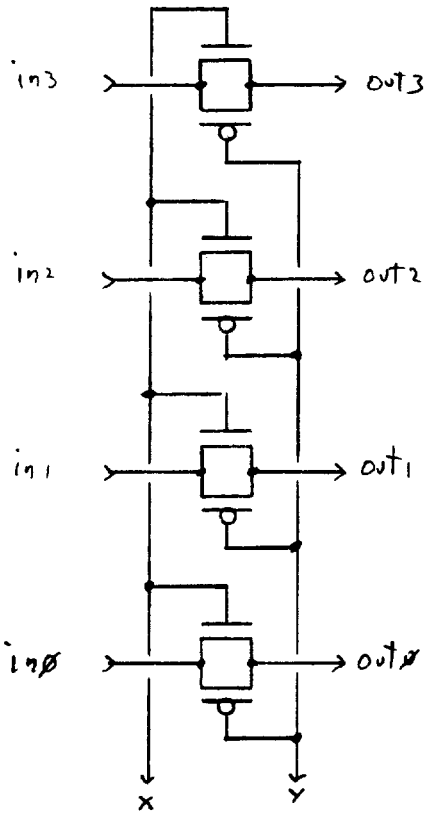
DecMOD - address decoder module

73 placements
398 transistors
200 wires

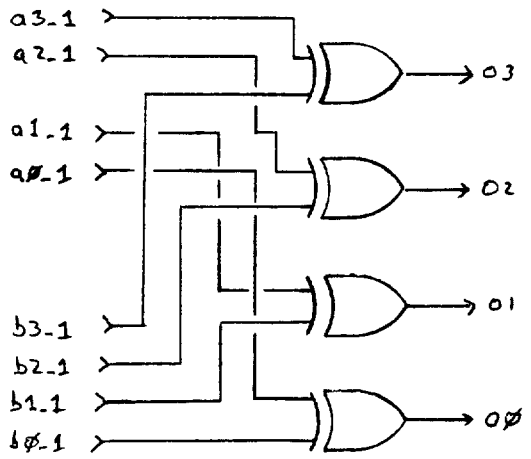


4 : 16 decoder

86 wires



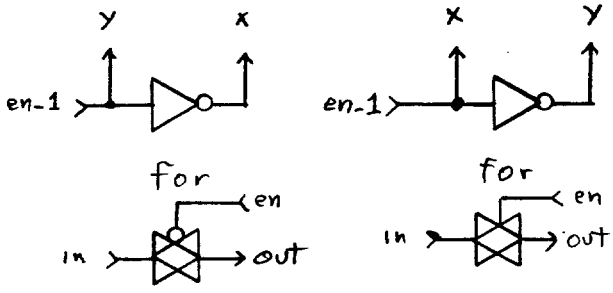
12½ wires



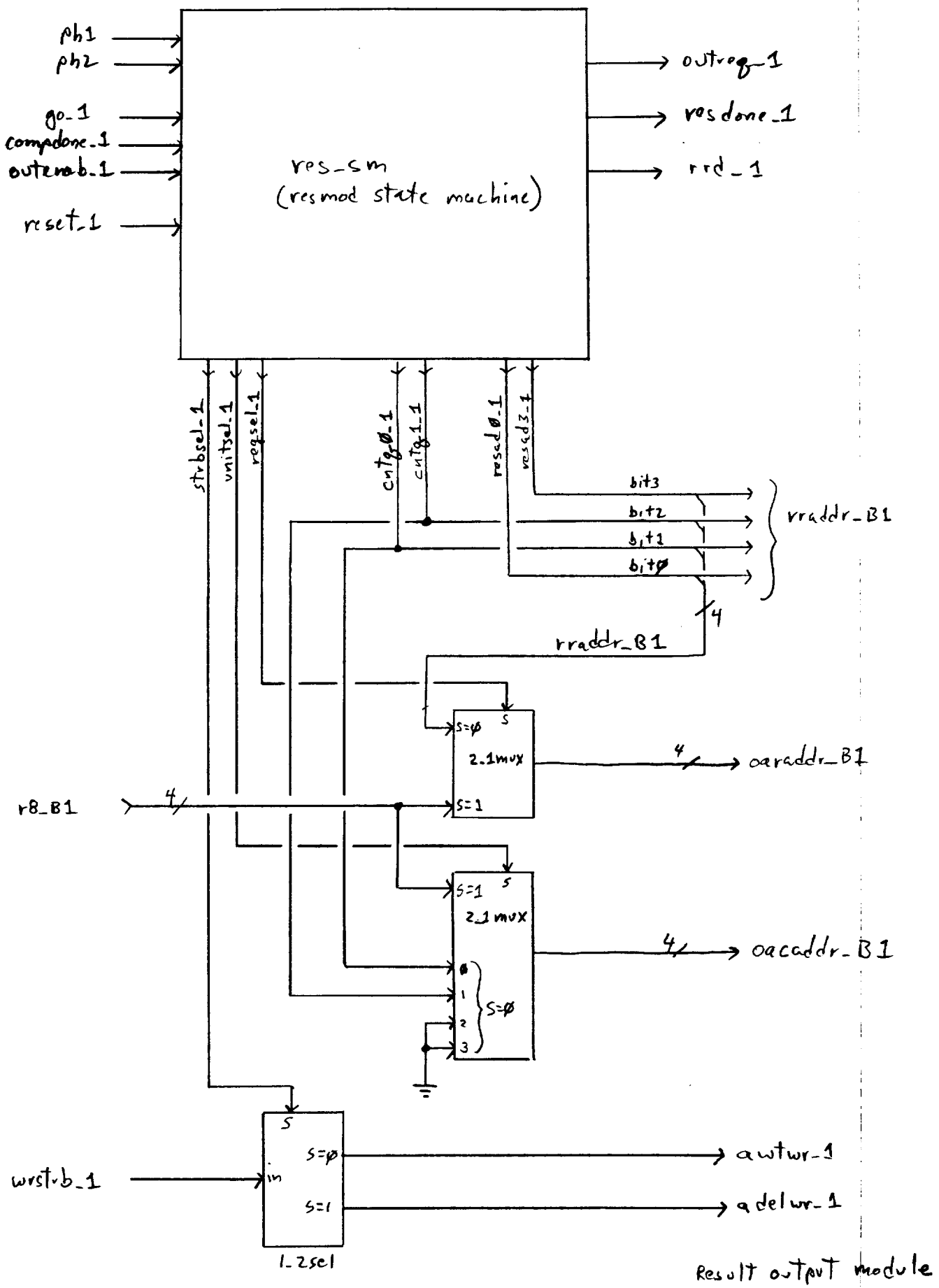
XOR ARRAY

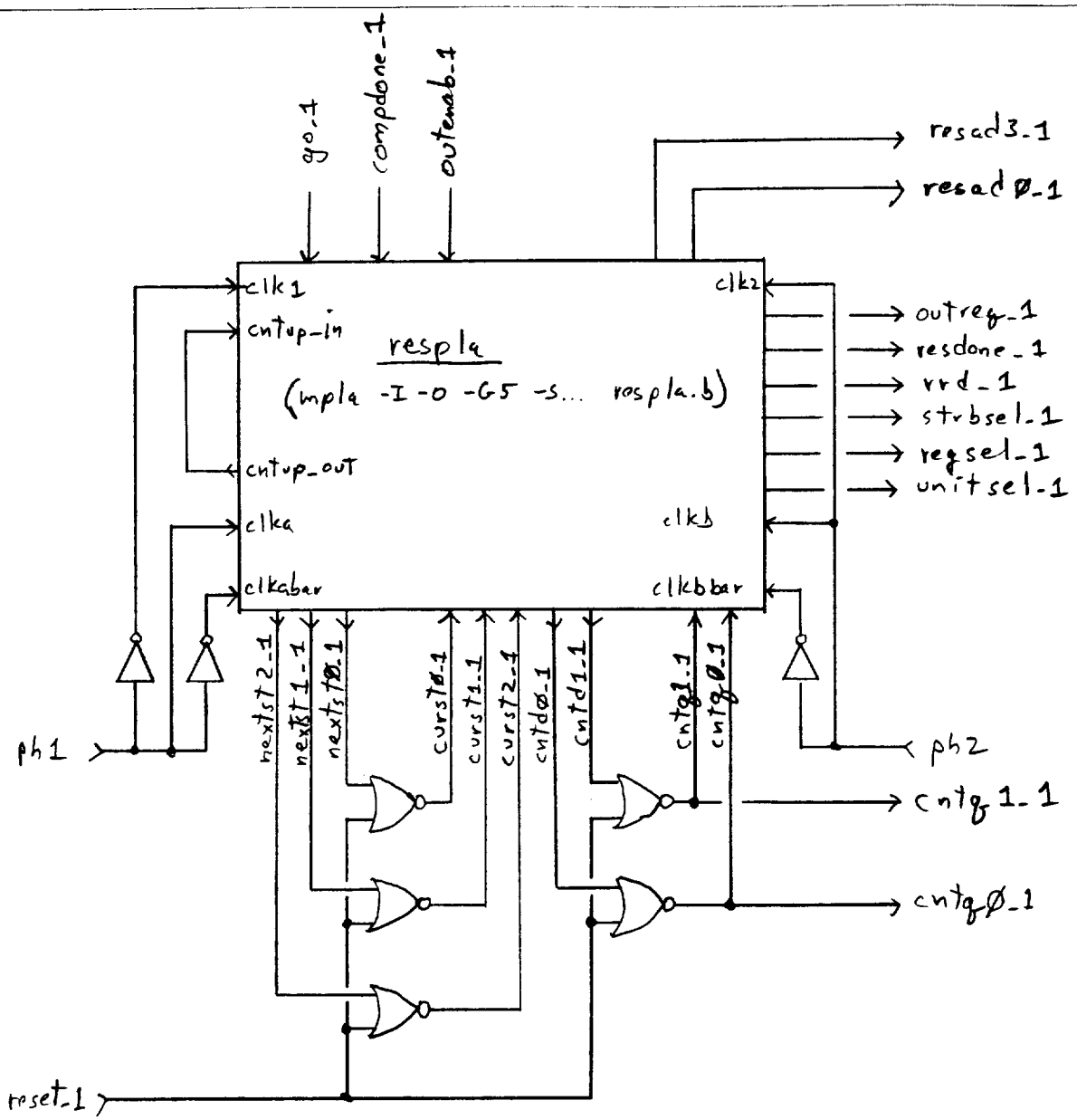
6 wires

BUS_SWITCH



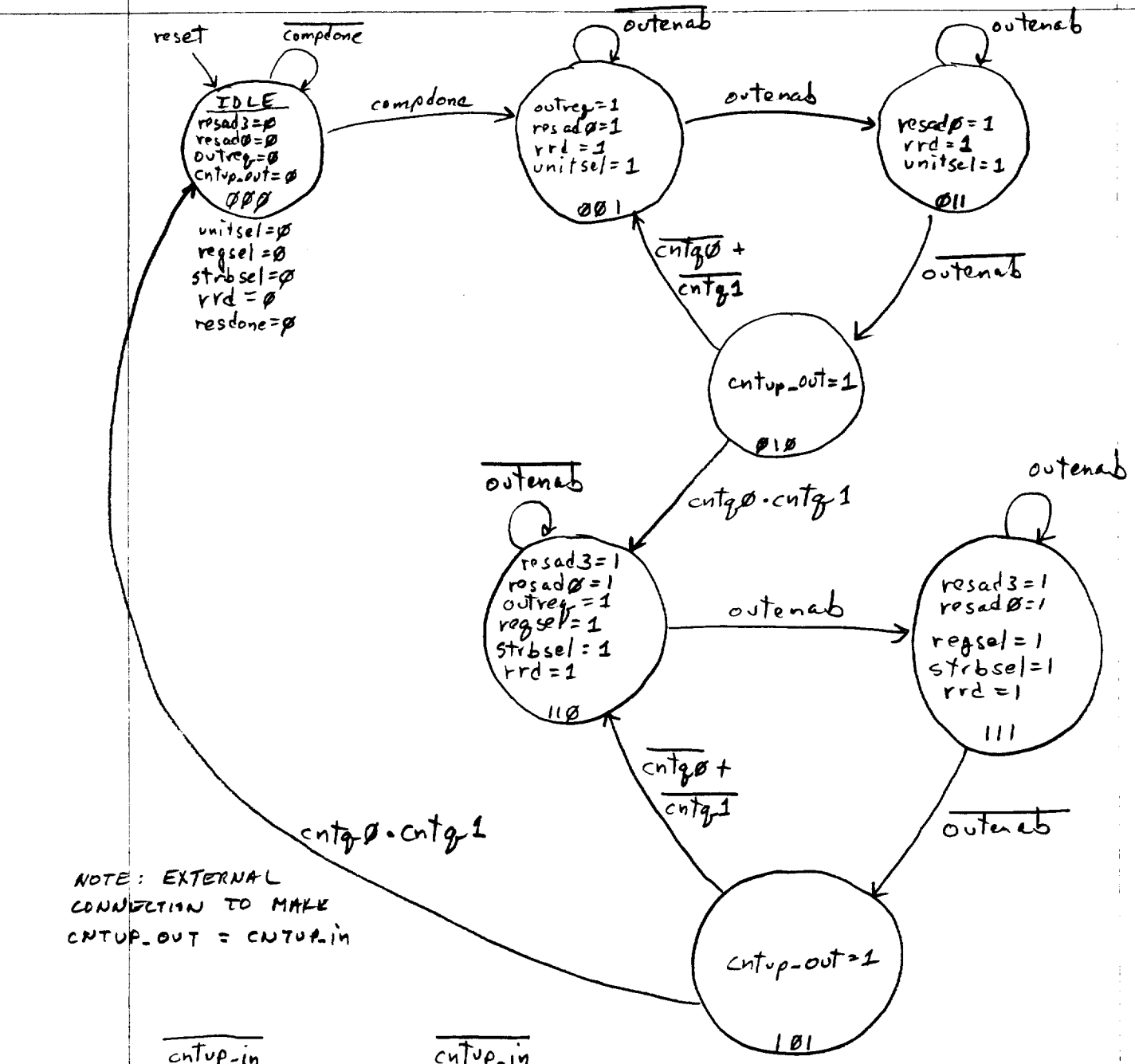
BSM units


ph1
ph2
go_1
compdone_1
outenab_1
reset_1
res_sm
(resmod state machine)
outreq_1
resdone_1
rrd_1
strbsel_1
vnitsel_1
reqsel_1
cntr0_1
cntr1_1
resd0_1
resd3_1
bit3
bit2
bit1
bit0
rraddr_B1
4
rraddr_B1
S=0
S
2_1mux
S=1
4
oaraddr_B1
r8_B1
4
S=1
S
2_1 mux
S=0
0
1
2
3
4
oacaddr_B1
S
wrstrb_1
in
S=0
S=1
awtwr_1
adelwr_1
1_2sel
Result output module
RESMOD

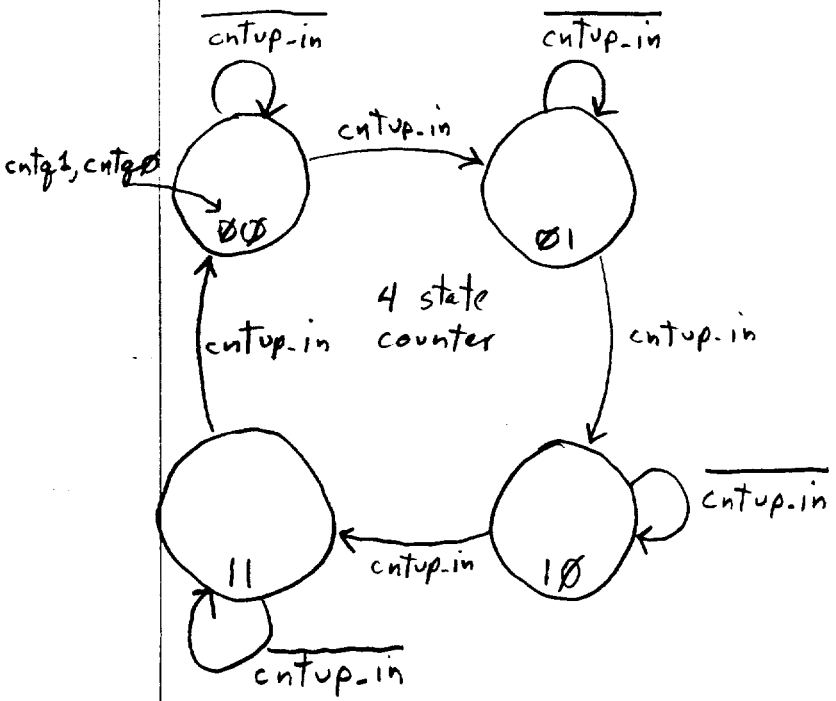


Res_sm

Resmod state machine



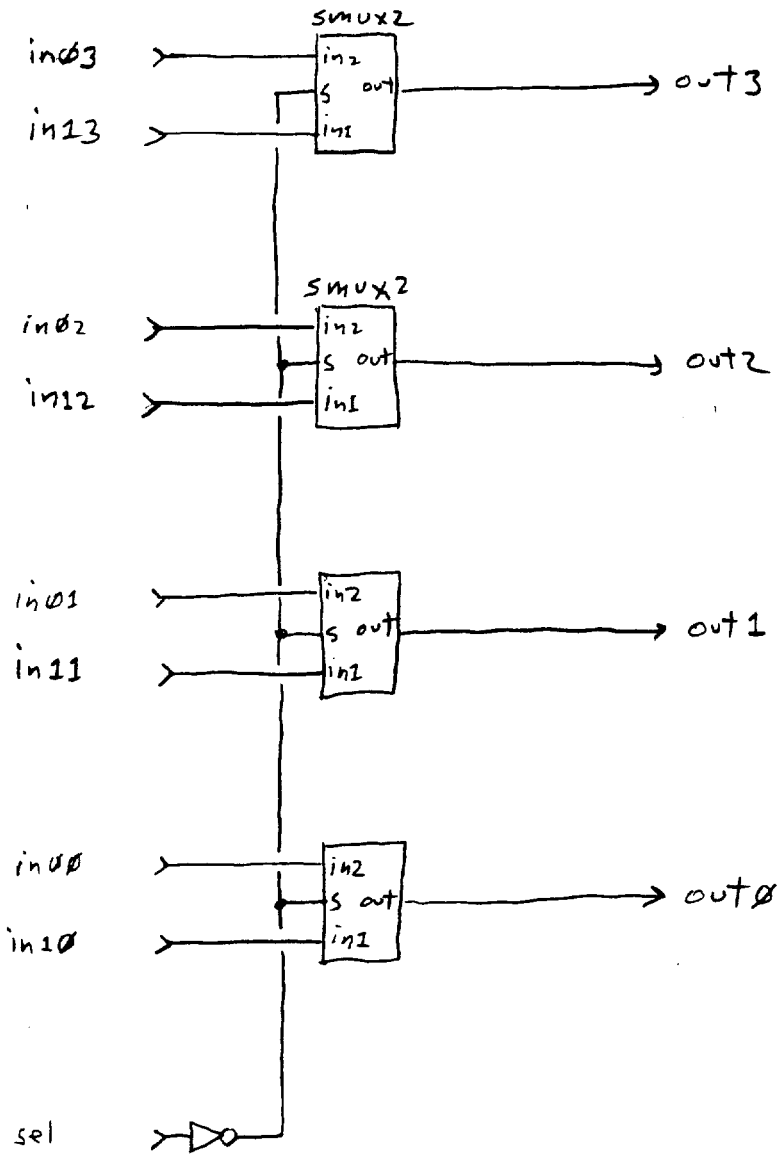
NOTE: EXTERNAL CONNECTION TO MAKE CNTUP_OUT = CNTUP_in



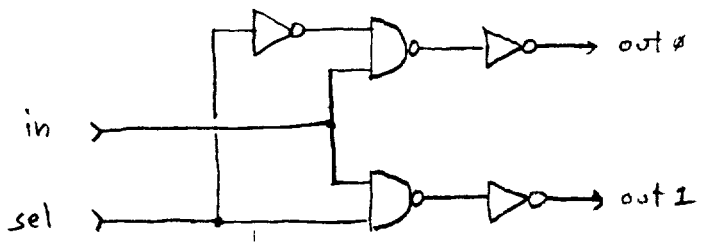
Address Map

O_{i0} - 0000	UnitAd - 1000
W_0 - 0001	Error0 - 1001
O_{i1} - 0010	O_j - 1010
W_1 - 0011	Error1 - 1011
O_{i2} - 0100	Σta - 1100
W_2 - 0101	Error2 - 1101
O_{i3} - 0110	X_{in} - 1110
W_3 - 0111	Error3 - 1111

Resmod State Machine State Diagram

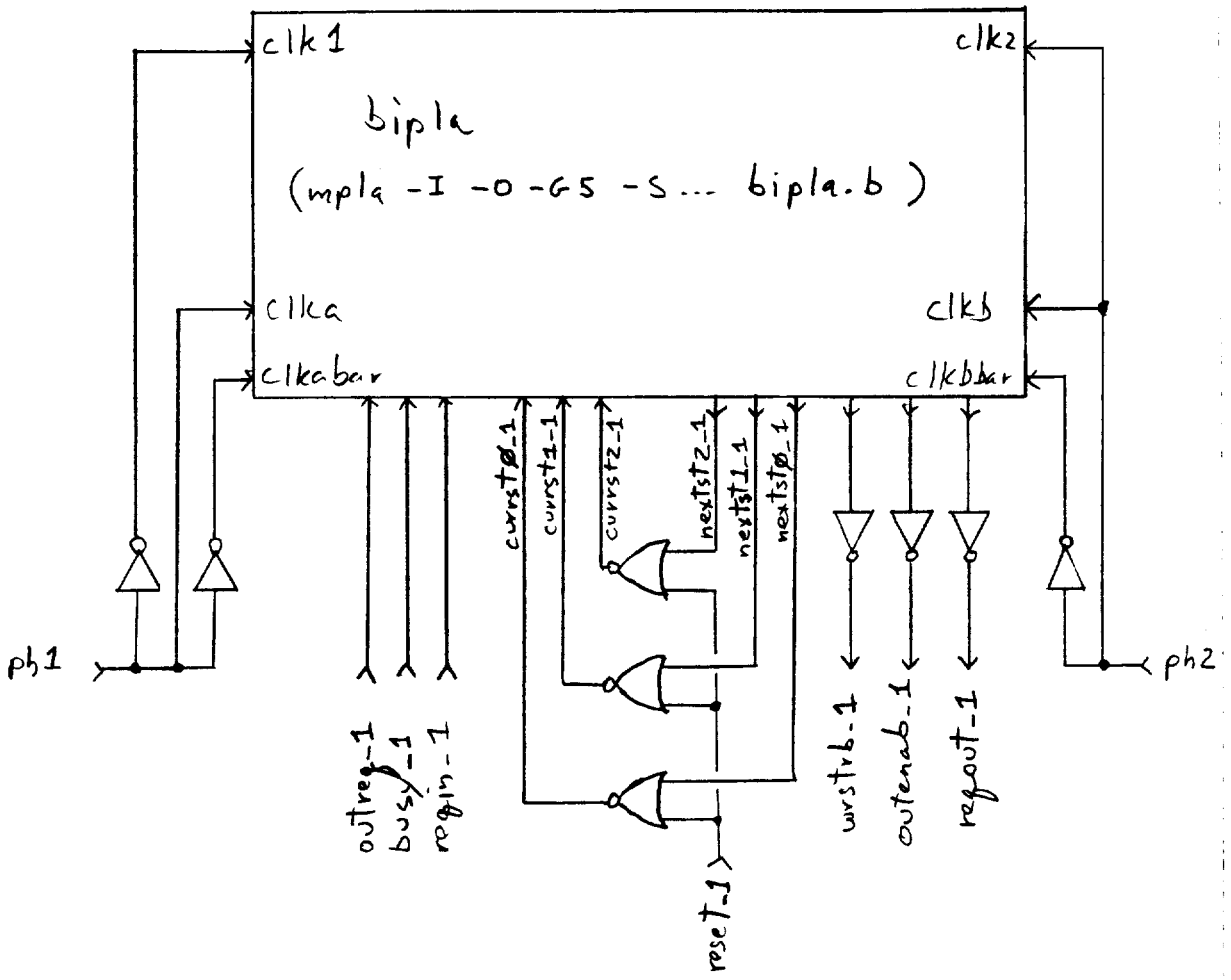


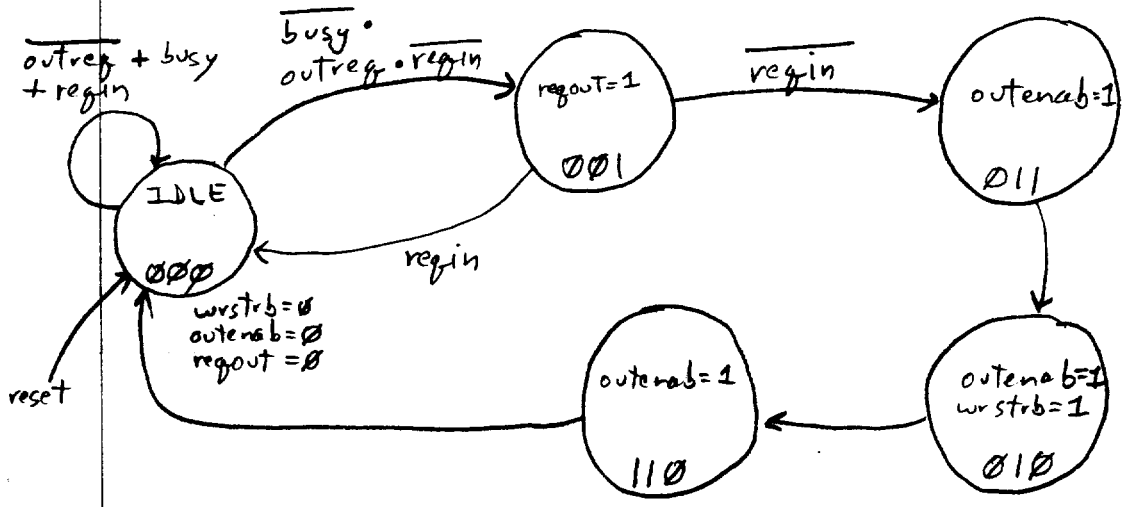
2_1MUX 4 BIT 2:1 MULTIPLEXER



1_2SEL
DATA SELECTOR

units for BSM

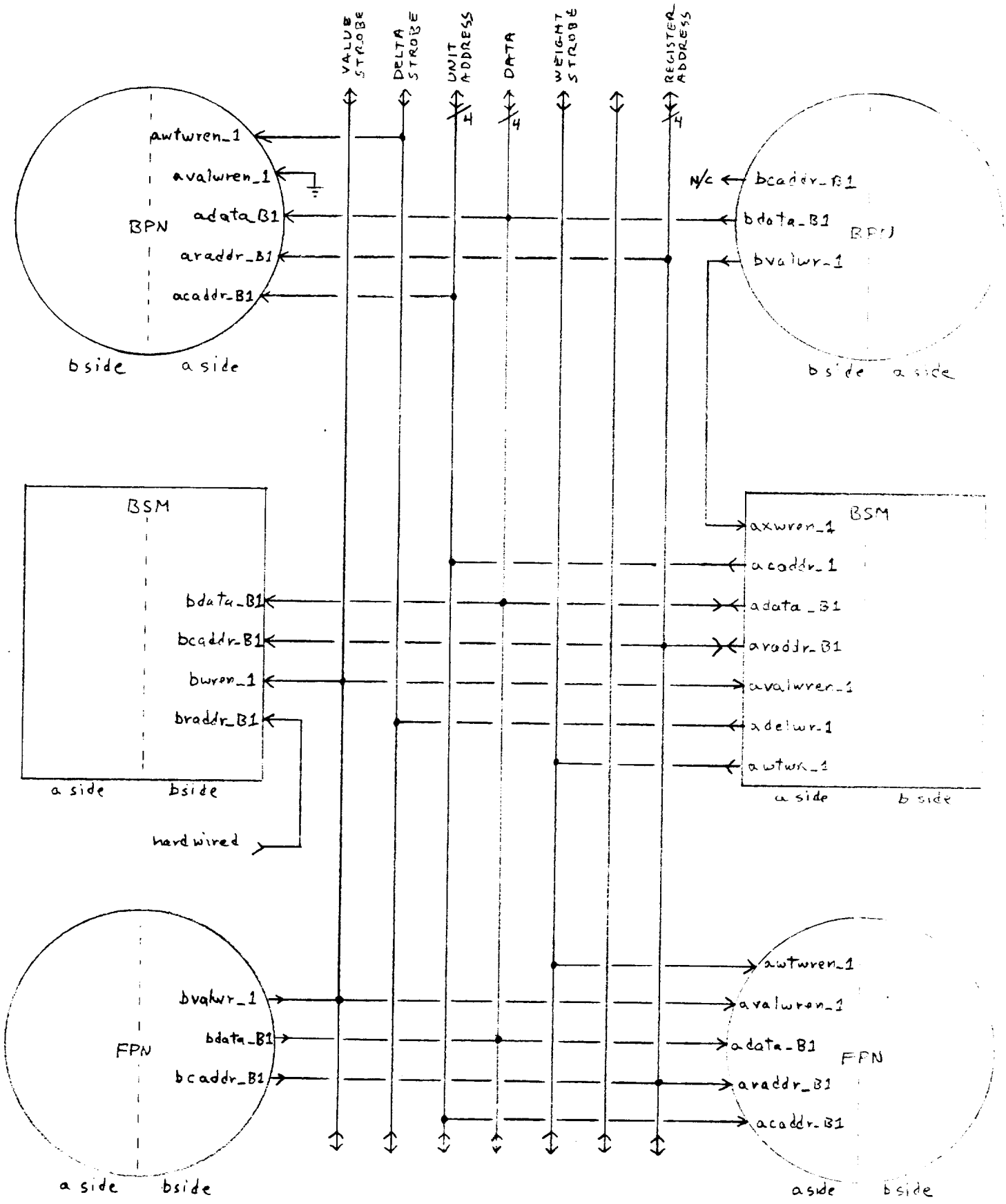




Bus Interface Module

BiMOD

STATE DIAGRAM



NOTES: PN & BSM a side must ignore unit address on value write but not on weight write.
BUS handshaking signals not shown.

FIG. 2-2
SYSTEM BUS INTERCONNECTION
MARCH 2, 1987

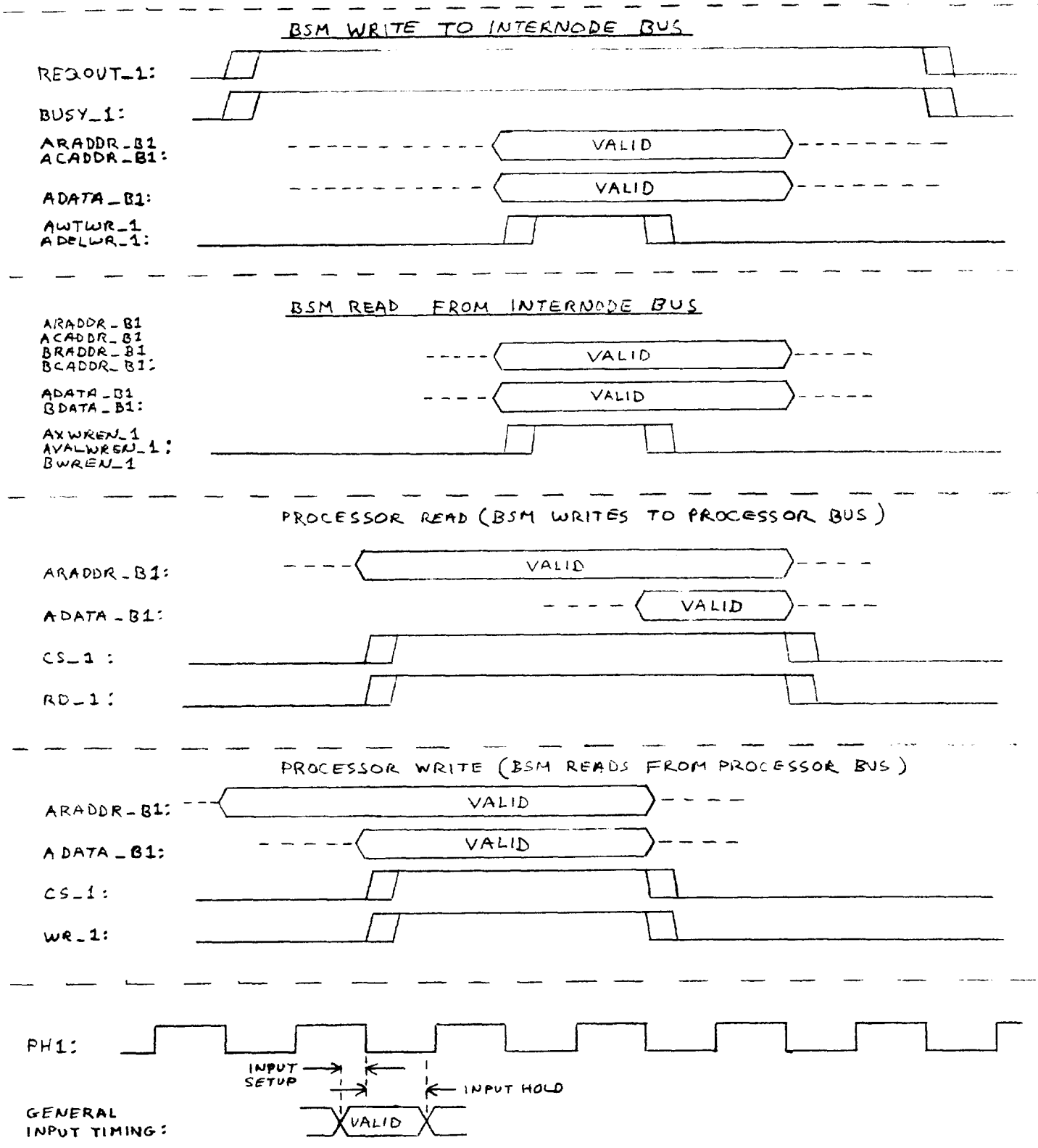
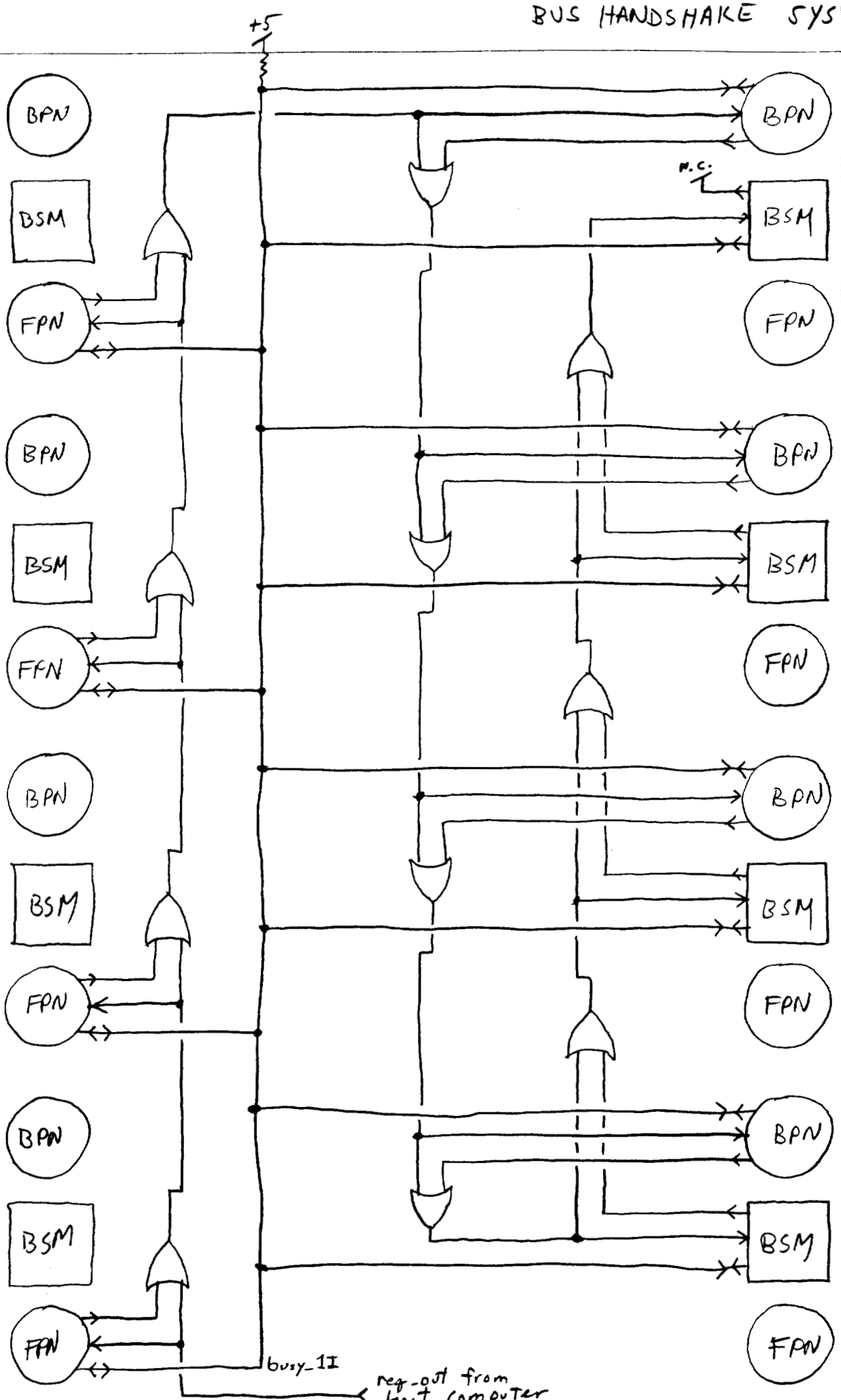


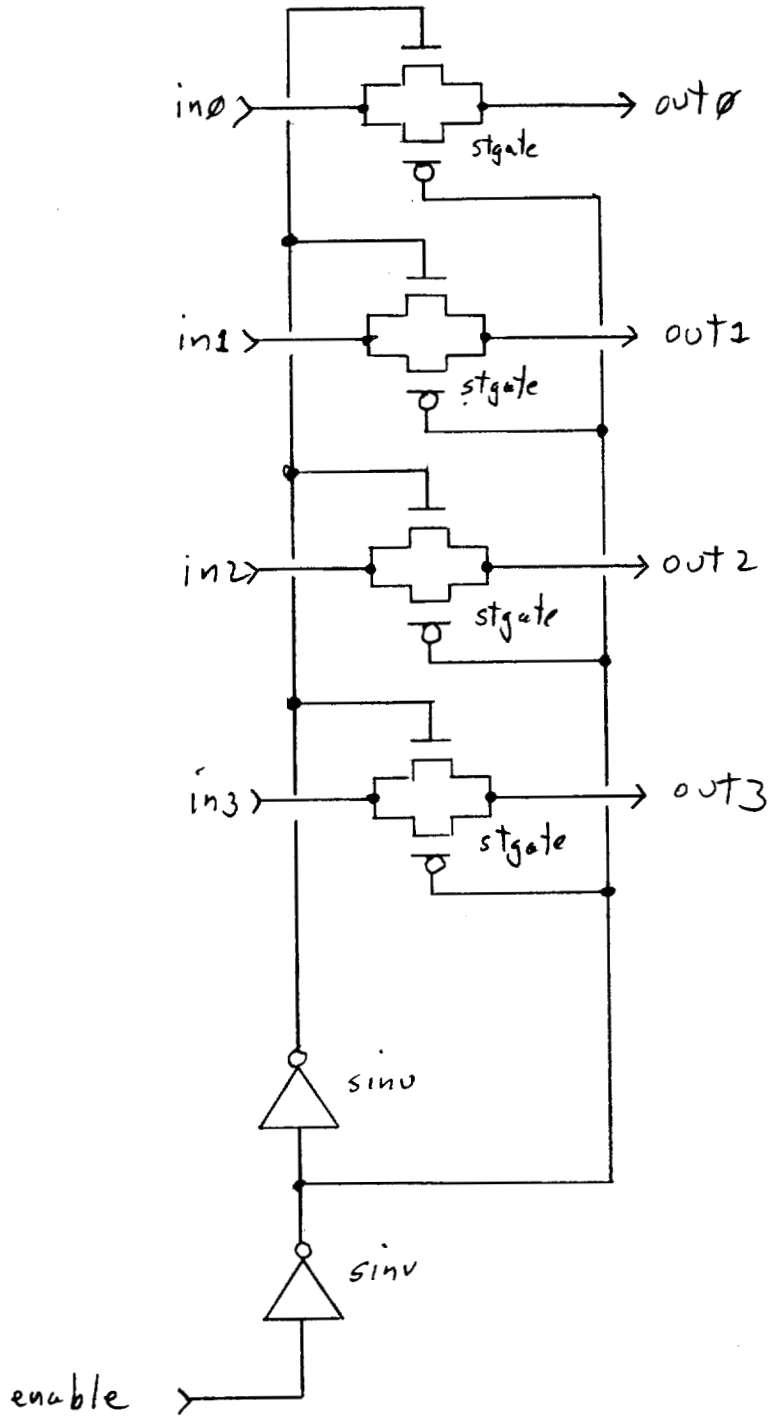
FIG. 4.1

BSM BUS TIMING

MARCH 2, 1987

BUS HANDSHAKE SYSTEM




in0
out0
stgate
in1
out1
stgate
in2
out2
stgate
in3
out3
stgate
sinv
sinv
enable

BUSBUF

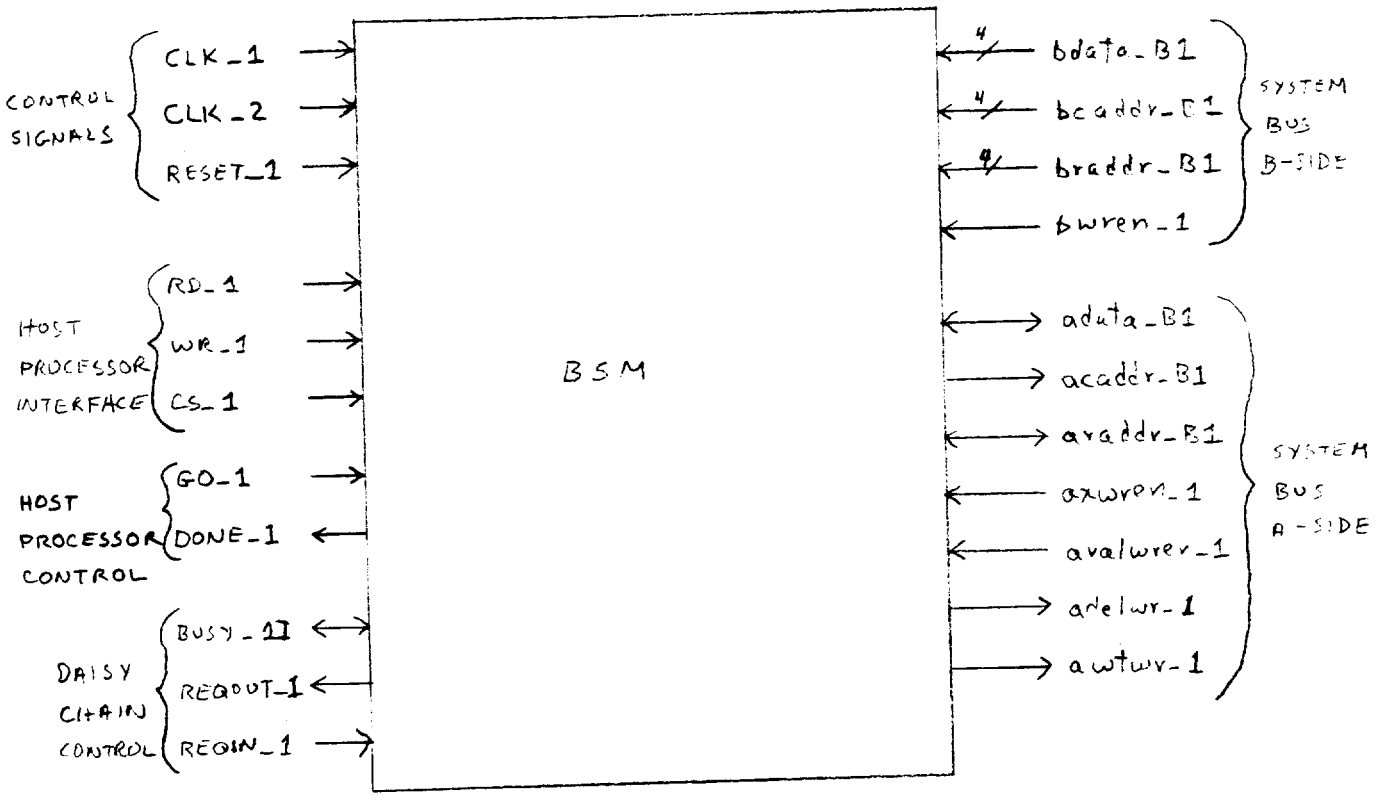


FIG. 2.3

BSM EXTERNAL INTERFACE
MARCH 2, 1987

BSM BLOCK DIAGRAM