Report Number: WUCS-87-9

1987-04-01

Design of a Broadcast Translation Chip

George H. Robbert

This paper describes the design of the Broadcast Translation Chip, one of the components of a high speed packet switch. The chip allows the packet switch to handle multi-point as well as point-to-point connections. It will be implemented in 1.5 Um CMOS technology.

DESIGN OF A BROADCAST TRANSLATION CHIP

George H. Robbert

WUCS-87-9

April 1987

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Abstract

This paper describes the design of the Broadcast Translation Chip, one of the components of a high speed packet switch. The chip allows the packet switch to handle multi-point as well as point-to-point connections. It will be implemented in 1.5 μm CMOS technology.

This work supported by Bell Communications Research, Italtel SIT, and National Science Foundation grant DCI-8600947.

DESIGN OF A BROADCAST TRANSLATION CHIP

George H. Robbert
ghr@wucs.UUCP

1. Introduction

This paper describes the design of the Broadcast Translation Chip (BTC) for use within a broadcast packet switching network. This network is described in [Tu85].

1.1. Packet Switch

The overall structure of the packet switch is shown in Figure 1. The switch terminates 7 fiber optic links. The packet switch is divided into five major sections. These are: the connection processor (CP), the Packet Processors (PPs), the Copy Network (CN), the broadcast translation chips (BTCs) and the routing network (RN). These are shown in figure 1. The CN, BTCs, and RN are collectively known as the Switch Fabric (SF). The PPs serve as buffers between the transmission links and also perform the address translation required by routing. The RN routes the packets passed through it

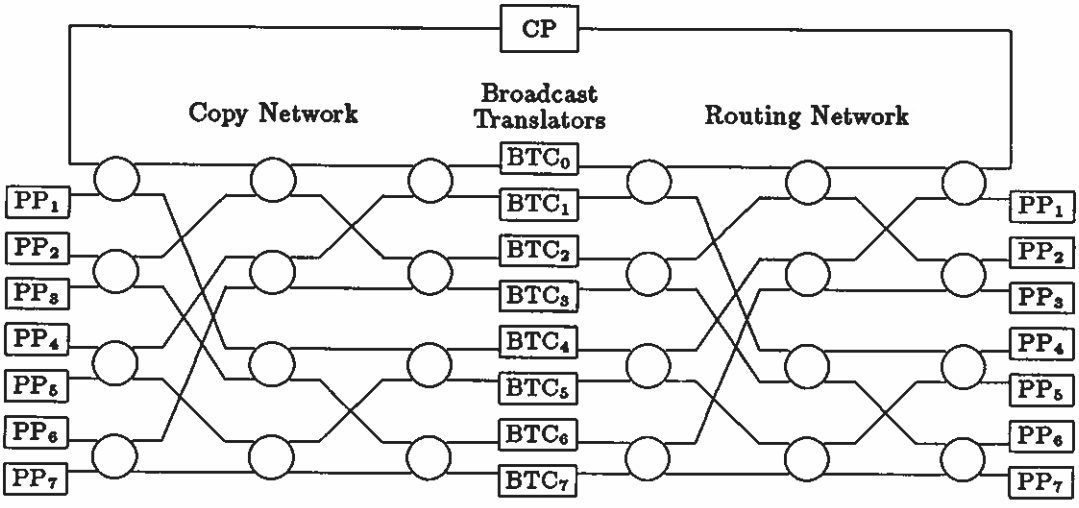


Figure 1: Switch Module

to the PP corresponding to the output link they are destined for. This information is encoded in the routing field (RF) of the packet. The CN and BTCs are described below.

The CN's function is to make copies of broadcast packets as they pass through. This is illustrated in Figure 3. The packet entering at left belongs to broadcast channel 35 and is to be sent to three different links. At the first stage, the packet is routed out the upper port. This is an arbitrary decision—the lower link could have been used at this point. At the second stage, the packet is sent out on both outgoing links and the fanout fields of the outgoing packets are modified. The upper packet will generate two copies and the lower one, one.

We now describe the CN routing algorithm for broadcast packets. Let BCN and FAN be the broadcast channel field and the number-of-copies field from the packet and let sn be the stage number of the node in the switch, where stages are numbered from right to left, starting with 0.

- If $\text{FAN} > 2^{sn}$ request both output links and when the links are available, simultaneously send the packet out on both.
  - If BCN is even, the FAN field of the upper packet is set to $\lfloor(\text{FAN}+1)/2\rfloor$ and the FAN field of the lower packet is set to $\lfloor\text{FAN}/2\rfloor$.
  - If BCN is odd, the FAN field of the upper packet is set to $\lfloor\text{FAN}/2\rfloor$ and the FAN field of the lower packet is set to $\lfloor(\text{FAN}+1)/2\rfloor$.
- If $\text{FAN} \leq 2^{sn}$, route packets alternately to each of the two output links. If the desired link is blocked, route to the other one.

Note that this algorithm delays copying packets as long as possible. Another option is to copy the packets early. However, this approach can lead to unbounded congestion in the CN. The late copying algorithm limits the potential for congestion. See [Bu85] for further details.

An important property of the copy algorithm is that given a particular combination of BCN and FAN, one can predict which of the FAN copies can appear at each output of the copy network. Suppose the copies of all broadcast packets were numbered from 0 to FAN with the indices increasing as the output port numbers increase. Consider what happens when a broadcast packet is processed by the copy network. Some CN output ports may receive a copy while others do not, depending on how the arbitrary routing decisions are made. However, if an output port receives a copy, the number of that copy is completely determined. The number of the copy that can appear at a particular output port is called the *broadcast copy index* or *bci* for that port. The *bci* is a function of the output port number, the FAN field and the low order bit of the BCN and is denoted $bci_k(\text{FAN}, \text{BCN})$. An algorithm to calculate $bci_k(\text{FAN}, \text{BCN})$ appears in Figure 2.

When the copies of a broadcast packet emerge from the CN, their final destinations are yet to be determined. The principal function of the Broadcast Translation Chips (BTC) is to assign a new routing field to each copy of a broadcast packet in such a way that a copy is received by every PP that is supposed to receive one. This is accomplished by a simple table lookup based on the broadcast channel number (BCN). Each BTC contains a *Broadcast Translation Table* (BTT), used for this purpose. The BTT is indexed by the BCN of the incoming packet and contains four nibble entries. When BTC_k receives a copy of a broadcast packet, it replaces the packet's routing field with the contents of $\text{BTT}_k[\text{BCN}]$.

Note that two BTCs need not have identical entries in their BTTs for a given BCN, since their *bci* values may be different. This point is illustrated in Figure 3 which shows the translation process for two packets copied from a single broadcast packet.

The addition of a new destination to a broadcast channel can radically change the mapping required in the BTTs. In general, when a new destination is added to a particular broadcast channel,

```
integer function bci_k(FAN, BCN)
    integer s, f;
    Let the binary representation of k be b_{n-1} ... b_1 b_0,
     where n is number of stages in copy network.
    s := 0; f := FAN;
    p := the least significant bit of BCN.
    for i := n - 1 downto 0 →
        if f > 2^i →
            if b_i = 0 →
                f := ⌊(f + 1 - p)/2⌋;
             | b_i = 1 →
                s := s + ⌊(f + 1 - p)/2⌋;
                f := ⌊(f + p)/2⌋;
            fi;
        fi;
    rof;
    return s;
end;
```

Figure 2: Computation of Broadcast Copy Index

the CP must calculate a new *bic* for each of the BTCs and use that information to update all their BTTs. Since this can happen frequently, it appears likely to present a substantial computational burden for the CP. Fortunately, there is a simple solution to the problem. It requires that for $0 \leq k \leq 7$, BTC_k contains a table, the *Broadcast Copy Index Table* (BCIT), that it can use to compute $bci_k(\text{FAN}, \text{BCN})$. Since, $\text{FAN} \leq 8$ and only the least significant bit of the BCN is needed to compute bci, the table has 32 entries, each one nibble long. Note that this table is static and is different for each BTC. Now, when the CP wishes to update the BTTs for a particular broadcast channel, it sends a control packet of the form shown in Figure 10. The CN replicates the packet so that each BTC receives a copy. When BTC_k receives its copy, it extracts FAN and BCN from the packet, then uses its lookup table to compute $j = bci_k(\text{FAN}, \text{BCN})$ and finally copies RF_j from the packet to $\text{BTT}_k[\text{BCN}]$.

Using this scheme, the CP keeps a copy of the BTT_*update* packet in its memory. To add a new destination, it increments the FAN field, adds a new RF to the end of the packet and sends it out to the RN. To remove a destination, it decrements the FAN field, removes the proper RF from the packet and sends it out.

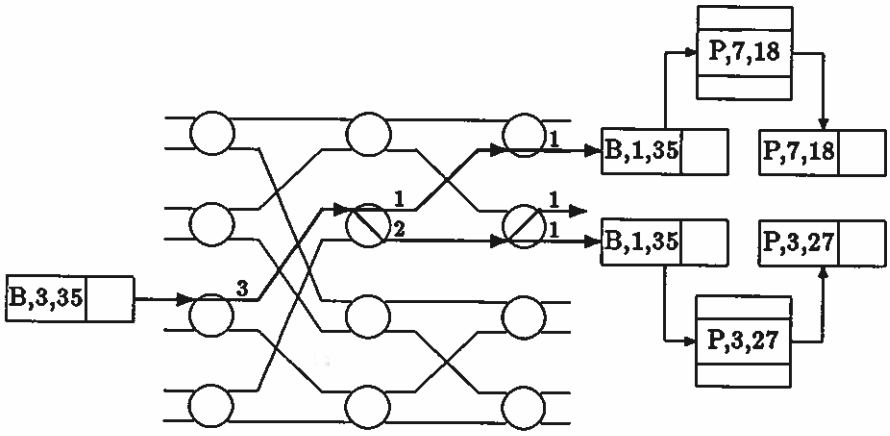


Figure 3: Example of Broadcast Processing

1.2. Packet Format

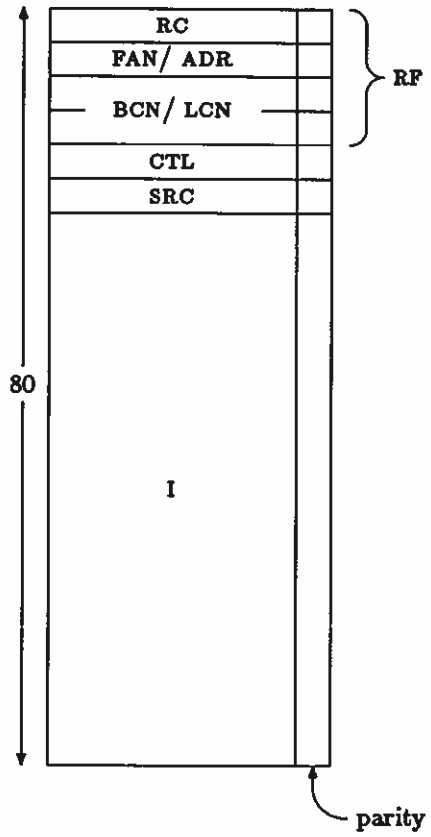


Figure 4: Packet Format

The format of the packets processed by the BTC is shown in Figure 4. The packet is organized as a sequence of four bit wide nibbles. Each packet contains exactly 80 nibbles, the first six of which constitute the packet *header*. The meanings of the fields are given below.

- *Routing Control* (RC). This field determines how the packet is processed by the nodes. 0000 signifies an empty packet slot, 0001 signifies a normal point-to-point data packet, 0011 signifies a normal broadcast packet, 0111 signifies a test packet.
- *Fanout* (FAN). If RC = 0011, the second nibble of the packet is taken to be the fanout, that is the number of switch fabric output ports that require copies of the packet.
- *Address* (ADR). If RC = 0001, the second nibble of the packet is taken to be the address of the packet, that is the switch fabric output port to which the packet is to be delivered.
- *Broadcast Channel Number* (BCN). If RC = 0011, the third and fourth nibbles of the packet are taken to be the broadcast channel number. All packets within a particular multi-point connection have the same broadcast channel number.

- *Logical Channel Number* (LCN). If RC = 0001, the third and fourth nibbles of the packet are taken to be the outgoing logical channel number. On the external fiber optic links, logical channel numbers are used to identify which connection a packet belongs to.
- *Control Field* (CTL). This field identifies various types of control packets. The possible values of the field and the corresponding functions are listed below.

  0 *Ordinary data packet.*

  1–4 *Packet Processor Control Packet.*

  5 *Switch Test Packet.*

  6 *Read* BTT *Block.* Directs BTC to read and return a block of 16 entries from the BTT. Nibble 0 of the I field specifies which of four blocks to read. The data is written into nibbles 1–64 of the I field.

  7 *Write* BTT *Block.* Directs BTC to write information into a block 16 entries of the BTT. Nibble 0 of the I field specifies which of four blocks to write to. The data to be written appears in nibbles 1–64 of the I field.

  8 *Update* BTT. Directs BTC to update one entry in the BTT. See operation below.

  9 *Read* BCIT. Directs the BTC to read the contents of the BCIT into nibbles 0–31 of the packet.

  A *Write* BCIT. Directs the BTC to write the information in nibbles 0–31 of the packet into the BCIT.

  B–F *Reserved.*

- *Source* (SRC). Identifies which switch fabric input port the packet came from.
- *Information* (I). Normally contains user information. In the case of control packets, may contain additional control information. Individual nibbles are denoted I[0], I[1], I[2], ... with I[0] being the first nibble of the I field.

2. Chip Specification

2.1. Interface

Here is a specification of the interface for the Broadcast Translation Chip (BTC) described above. The external leads of the BTC are shown in Figure 5 and described briefly below. This figure, as well as the others describing major pieces of the BTC use a form of dependency notation. A description of this notation may be found in [IE]

- *Upstream data leads* ($ud_0 - ud_3$) Incoming data from upstream neighbors. Four bits wide (one nibble).
- *Upstream parity* (up) Odd parity on incoming data from upstream neighbor.
- *Downstream data leads* ($dd_0 - dd_3$) Outgoing data to downstream neighbors. Four bits wide (one nibble).
- *Downstream parity* (dp) Odd parity on outgoing data to downstream neighbor.
- *Packet Time* (pt). Goes high when first nibble of packet is present on ud leads.

- *Reset* (rst) Initialize internal state machines.
- *Soft Reset* (srst) Reset error flags on chip.
- *Parity Error* (e0) Report parity error.
- *Control Error* (e1) Report bad CTL field of packet
- *Framing Error* (e2) Report inappropriate pt assertion.
- *Error* (err) Report parity violation or other error. This signal is the logical OR of e0 through e2.
- *Test Data Leads* ($\texttt{srt}_0 - \texttt{srt}_3$) Chip Testing outputs. These are the outputs of the last stage of the packet header storage register. Four bits wide (one nibble).
- *Test Data Parity* (srtp) Odd parity on Chip Testing outputs.
- *Clock* (ϕ_1, ϕ_2). Two-phase, non-overlapping clock.
- *Power and Ground* (P, G).

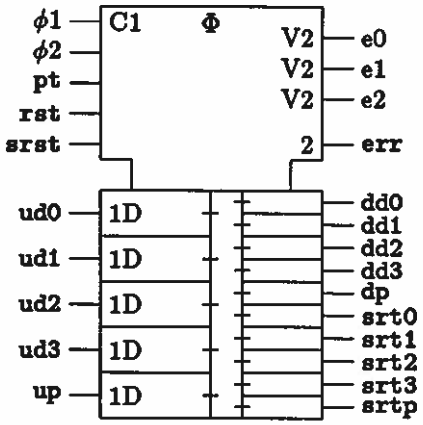


Figure 5: BTC Connections

2.2. Operation

The BTC operates on the basis of a *packet cycle* which starts when the pt lead goes high. This indicates that the first nibble of an incoming packet is present on the upstream data leads ud. Successive nibbles of the packet then arrive on successive clock cycles.

The chip is clocked by a two phase non-overlapping clock. The desired clock rate is 10 MHz giving a clock period of $100ns$. When ϕ_1 is high, the signals on the ud and up leads of the chip are valid. The pt lead goes high while ϕ_1 is low and stays high for one clock cycle. The first nibble of an incoming packet is present when both ϕ_1 and pt are high. Successive up-going transitions of pt are at least 80 clock cycles apart.

Odd parity is computed across every nibble as it is received and checked against the parity bit on the up lead. Any discrepancies cause the error lead to be asserted. Parity is also computed across every nibble as it leaves the chip and the new parity value is transmitted on the dp lead.

The BTC contains two internal tables. The *Broadcast Translation Table* (BTT) is used to modify the headers of broadcast packets as they pass through the BTT. It has 64 entries, each of which is four nibbles wide. The i^{th} entry is denoted BTT[i]. Parity is also stored for each nibble and checked when data is read from the BTT. Parity errors cause the error lead to be asserted. The *Broadcast Copy Index Table* (BCIT) is used for updating the BTT. It consists of 32 entries, each of which is one nibble wide. Again parity is stored for each nibble and checked when data is read.

Normal point-to-point data packets (those with RC = 0001 and CTL = 0) pass through the BTC unchanged, experiencing a delay of 16 clock cycles. The same goes for control packets that don't affect the BTC. The actions taken by the BTC on broadcast data packets and BTC control packets are described in figure 6. Note that, in some cases (writes to BTC memory), the BTC does not pass the packet through. In all other cases, the (possibly modified) packet is passed through, with a delay of 16 clock cycles.

3. High Level Design

The BTC may be divided into three basic sections: The shift register, the memory, and the associated control circuitry. A block diagram is shown in figure 7.

3.1. Shift Registers

The heart of the BTC is a shift register 5 bits wide and 16 stages long. It provides sufficient buffering to allow the proper action to be taken on the packet. Various stages of this shift register may be either read or written to get information from the packet or to modify it. A second shift register is used to store the packet header and the first nibble of the I field. The first 7 nibbles of a packet are shifted into this shift register and stored since they may be used by the control circuitry throughout the packet cycle.

3.2. Memory

There are two memories in the BTC. The larger one is the BTT. It stores the new routing information for the broadcast packets that this BTC is translating. The smaller one, the BCIT, is used for updating the BTT. It is used to calculate the *Broadcast Copy Index* via table lookup. Since the *Broadcast Copy Index* is unique for each BTC, this table is implemented in RAM. All the BTC chips are identical and the BCIT in each is loaded with the unique function values for that chip. Both of these memories are

```
if CTL = 0 ∧ RC = 0011 →
    Replace the first four nibbles of the packet with BTT[BCN].
    If new ADR field equals SRC field, do not propagate the packet.
| CTL = 6 →
    i := 16 * I[0];
    for j ∈ [0, 15] →
        Copy BTT[i + j] to I[4j + 1], I[4j + 2], I[4j + 3], I[4j + 4].
    rof;
| CTL = 7 →
    i := 16 * I[0];
    for j ∈ [0, 15] →
        Copy I[4j + 1], I[4j + 2], I[4j + 3], I[4j + 4] to BTT[i + j].
    rof;
    Do not propagate the packet.
| CTL = 8 →
    i := (I[0] << 1) | (BCN_L & 01);
    j := BCIT[i];
    Copy I[4j + 1], I[4j + 2], I[4j + 3], I[4j + 4] to BTT[BCN].
    Do not propagate the packet.
| CTL = 9 →
    for j ∈ [0, 31] → Copy BCIT[j] to I[j]. rof;
| CTL = A →
    for j ∈ [0, 31] → Copy I[j] to BCIT[j]. rof;
    Do not propagate the packet.
| else
    Pass packet unchanged.
fi;
```

Figure 6: BTC Packet Actions

accessed from the main shift register. To improve reliability, both memories store parity for each nibble. This is checked when the memory is read.

3.3. Control

The control section of the BTC is distributed into four functional blocks. Three of these control specific sections of the packet processing while the fourth coordinates their actions. The sections controlled by these three are:

- memory refresh
- block reads and writes of the BTT and BCIT
- controlling packet propagation

There is also a counter to keep track of the position within the current packet. Another counter is used to determine which of the "new packet headers" to write to the BTT in a single entry update.

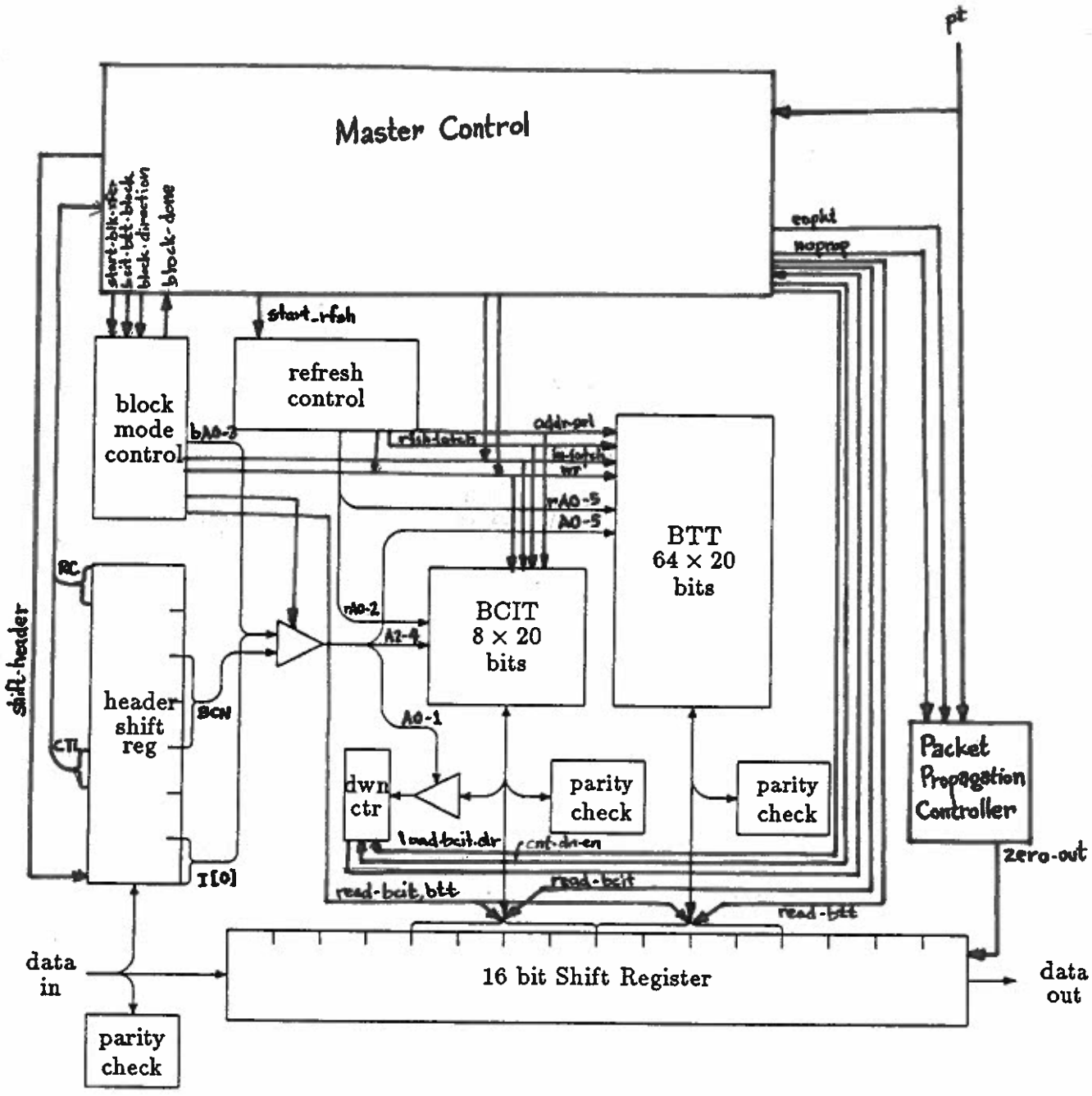


Figure 7: Circuit Diagram

4. Node Operation

All packets begin processing in the same manner. The header is shifted into both the header storage and main shift registers. At this point the BTC determines what type of packet it is and proceeds accordingly. Point to point packets and all others that the BTC doesn't recognize are merely passed through the main shift register to appear unchanged at the output. Other types of packets require special processing. Two examples are given below.

4.1. Normal Broadcast Packet

Figures 8 and 9 illustrate the processing of a simple broadcast packet. In figure 9, the thick lines indicate which datapaths are used in broadcast translation. The BCN field of the packet header is used to index the BTT and obtain new routing information for the packet. This replaces the original packet header in the main shift register. If the new destination is the same as the originator of the packet (SRC), the packet is not propagated. Otherwise, the modified packet is output.

4.2. Load Single BTT Entry

Figures 10 and 11 illustrate the processing of a *update single* BTT *entry* packet. In figure 11, the thick lines indicate which datapaths are used in BTT updates. The *broadcast copy index* is calculated by looking it up in the BCIT. The low order bit of the BCN and the new FAN (held in I[0]) are used to address the BCIT. The low order 2 bits of this address select the correct nibble from the slice of 4 nibbles read from the BCIT. The rest is used to address the BCIT memory array. This selects the correct *bci* value from the table. A map of how BCIT holds the *bci* values is shown in figure 12. This value read from the BCIT (ie. $bci(\text{FAN}', \text{BCN})$) is loaded into the down counter which is decremented every 4 nibble times. When it reaches 0, the appropriate RF is ready to be loaded into the BTT. The BCN is the address of the location in the BTT where the new RF is to be written.

The block reads and writes of the BTT and BCIT are done similarly. However, the calculation of the *broadcast copy index* is not required in these cases. Also the memory addresses come from the `block mode control` section instead of from the packet header.

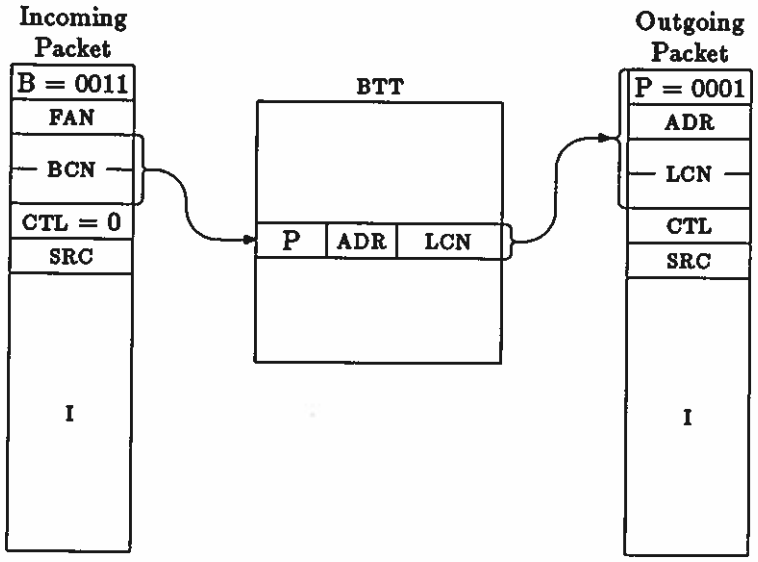


Figure 8: Broadcast Packet Translation

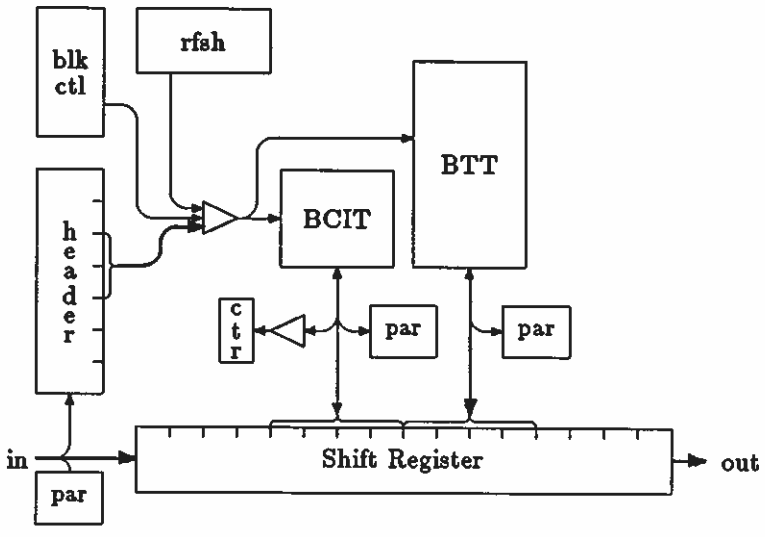


Figure 9: Broadcast Packet Translation

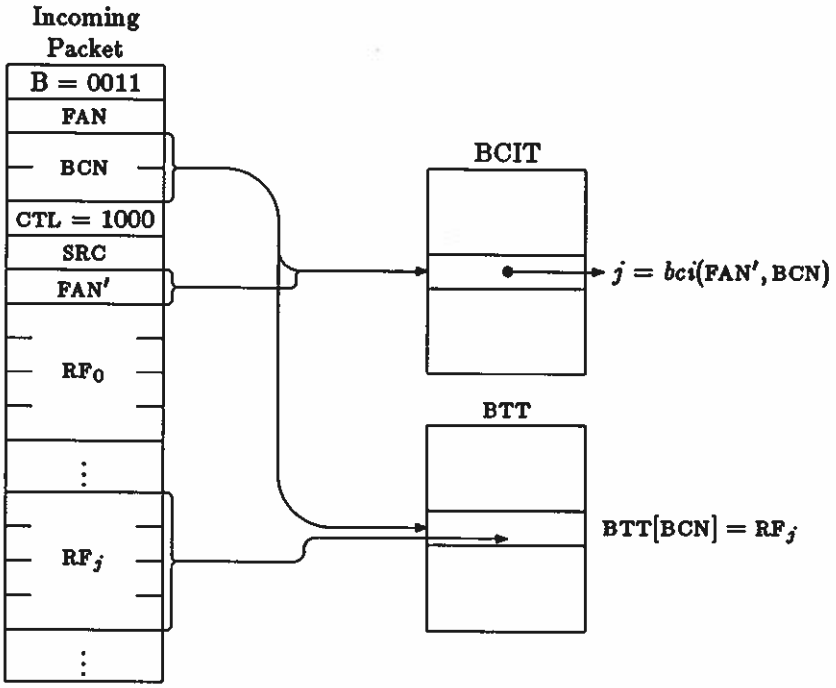


Figure 10: BTT Update (single entry)

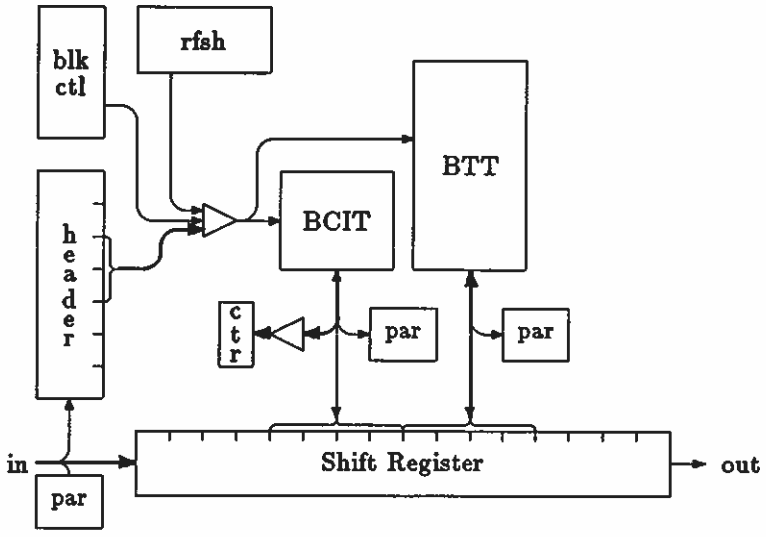


Figure 11: BTT Update (single entry)

7	$bci(14, even)$	$bci(14, odd)$	$bci(15, even)$	$bci(15, odd)$
6	$bci(12, even)$	$bci(12, odd)$	$bci(13, even)$	$bci(13, odd)$
5	$bci(10, even)$	$bci(10, odd)$	$bci(11, even)$	$bci(11, odd)$
4	$bci(8, even)$	$bci(8, odd)$	$bci(9, even)$	$bci(9, odd)$
3	$bci(6, even)$	$bci(6, odd)$	$bci(7, even)$	$bci(7, odd)$
2	$bci(4, even)$	$bci(4, odd)$	$bci(5, even)$	$bci(5, odd)$
1	$bci(2, even)$	$bci(2, odd)$	$bci(3, even)$	$bci(3, odd)$
0	$bci(16, even)$	$bci(16, odd)$	$bci(1, even)$	$bci(1, odd)$

Figure 12: Map of BCIT Entries

5. Medium Level Design

This section contains a more detailed description of each of the functional blocks described previously.

5.1. Shift Register

Both of the shift registers use the same basic dynamic latch cells. This 8 transistor 2 phase latch was chosen for its simplicity, size and speed. These are described below.

5.1.1. Main Shift Register. The main shift register is 16 stages long and 5 bits wide. Since data is constantly being shifted, there is no penalty for using dynamic circuitry here. A simple multiplexor allows some stages of the main shift register[1] to be loaded from either the previous stage (for shifting) or from an external source (ie. the memory).

The interface of bit slice n of the main shift register is shown in figure 13. Note: to match figure 7, this figure must be turned on its side (rotated 90° counterclockwise). Five such slices compose the main shift register. Here are the interface signals.

- *Data Inputs* (data_in$_0$ – data_in$_4$) The 5 bits of serial input to the shift register. These are the upstream data leads plus the upstream parity lead (ud$_0$ – ud$_3$ and up).
- *Data Outputs* (data_out$_0$ – data_out$_4$) The 5 bits of output from the shift register. These are the downstream data leads plus the downstream parity lead (dd$_0$ – dd$_3$ and dp).
- *Load From* BCIT (read_bcit) Load stages 4 through 7 of the shift register with the value read from the BCIT instead of the shifted in value. Connected to BLOCKMODE_BLK/bcit_read.
- *Load From* BTT (read_btt) Load stages 8 through 11 of the shift register with the value read from the BTT instead of the shifted in value. Input from BLOCKMODE_BLK/btt_read + MASTER_BLK/btt_read.
- *Input Data From* BCIT (bcit_in$_{n+0}$ – bcit_in$_{n+15}$) Data output by the BCIT. In these labelings n is the number of this bit slice. So, for bit slice 0, these would be labeled bcit_in$_0$, bcit_in$_5$, bcit_in$_{10}$ and bcit_in$_{15}$. The other inputs and outputs from memory are labeled similarly. These inputs are loaded into the shift register during BCIT block reads. Input From BCIT/bit_out$_0$ – BCIT/bit_out$_{19}$.
- *Input Data From* BTT (btt_in$_{n+0}$ – btt_in$_{n+15}$) Data output by the BTT. Loaded into the shift register during BTT block reads and broadcast packet translation. Input from BTT/bit_out$_0$ – BTT/bit_out$_{19}$.
- *Output Data To* BCIT (bcit_out$_{n+0}$ – bcit_out$_{n+15}$) Data to be written to the BCIT. These outputs are used during BCIT block writes. Output to BCIT/bit_in$_0$ – BCIT/bit_in$_{19}$.
- *Output Data To* BTT (btt_out$_{n+0}$ – btt_out$_{n+15}$) Data to be written to the BTT. These outputs are used during BTT updates and block writes. Output to BTT/bit_in$_0$ – BTT/bit_in$_{19}$.
- *Clock* (ϕ1 and ϕ2) Two-phase, non-overlapping clock. Data is loaded into the shift register and shifted on ϕ1. Outputs are latched on ϕ2.
- *Propagate Packet* (propagate) If this lead is true, the packet is propagated. If false, 0 nibbles are output. Input from PKTPROP_BLK/zero_out.

[1] Specifically stages 4 through 11.

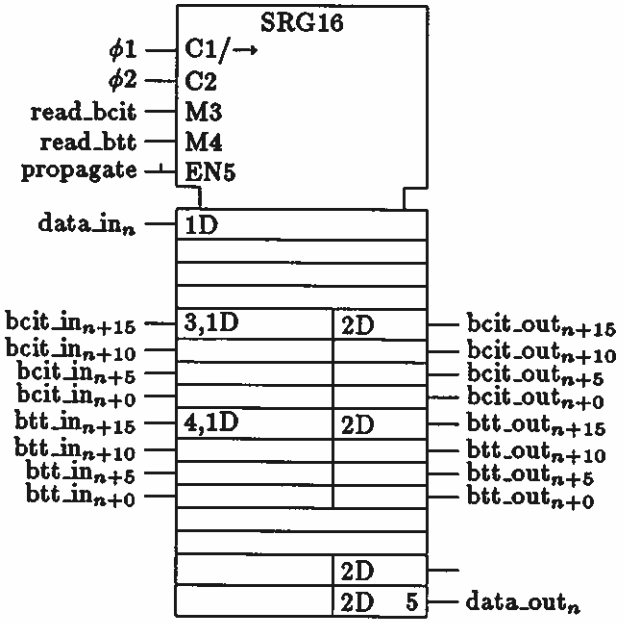


Figure 13: One Bit Slice of Main Shift Register

5.1.2. Packet Header Shift Register. The packet header shift register is 8 stages long and 5 bits wide. Since the maximum length of time data is to be held (without shifting) is one packet time, dynamic circuitry may also be used here.

The interface of bit slice n of the packet header shift register is shown in figure 14. This figure is inverted from its orientation in figure 7. Five such slices compose the packet header shift register. Here are the interface signals.

- *Data Inputs* ($\texttt{data\_in}_0 - \texttt{data\_in}_4$) The 5 bits of serial input to the shift register. These are the upstream data leads plus the upstream parity lead ($\texttt{ud}_0 - \texttt{ud}_3$ and up).
- *Stage 0 outputs* ($\texttt{i[0]}_0$ - $\texttt{i[0]}_4$) These are the I[0] inputs to the master control block.
- *Stage 2 outputs* ($\texttt{ctl}_0 - \texttt{ctl}_4$) These are the CTL inputs to the master control block.
- *Stage 3 outputs* ($\texttt{bcn\_l}_0 - \texttt{bcn\_l}_4$) These are the low order 4 bits of the BCN input to the Memory Address Multiplexor.
- *Stage 4 outputs* ($\texttt{bcn\_h}_0 - \texttt{bcn\_h}_4$) These are the high order 4 bits of the BCN input to the Memory Address Multiplexor.
- *Stage 6 outputs* ($\texttt{rc}_0 - \texttt{rc}_4$) The 5 bits of output from the shift register. These are the test data leads plus the test data parity lead ($\texttt{srt}_0 - \texttt{dd}_3$ and srt). These are also the RC inputs to the master control block.
- *Shift Enable* (shift_ena) This input enables the shifting for this shift register. When it is true, data is shifted on $\phi 1$, When it is false, data is held.
- *Clock* ($\phi 1$ and $\phi 2$) Two-phase, non-overlapping clock. Data is shifted on $\phi 1$, if shifting is enabled. Outputs are latched on $\phi 2$.

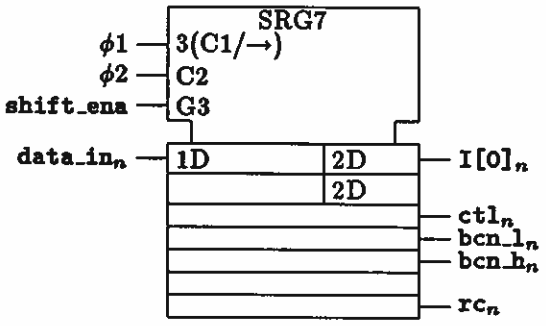


Figure 14: One Bit Slice of Packet Header Shift Register

5.2. Memory

Both memories are implemented using the same 3 transistor dynamic RAM design. Our design gives a cell size of $30 \times 30\lambda$ for 1 bit. The row decoders are implemented using a dynamic nor. Since a three transistor cell is used, the sense amplifiers can be simple inverters. The sense amplifiers/column drivers contain the circuitry to precharge the bit lines in the array, a latch for data to be written to the memory and the sense amplifying inverter.

The memory has two basic operating cycles: read and write. Refresh is handled by a read immediately followed by a write. Hand calculation of the delays through the larger of the two arrays (BTT) yields an access time of about $275ns$. Since we need $< 400ns$ access time on each of the memories, we have not tried for a more precise estimate of the delay through it.

Both memories are read and written in 4 nibble words in order to obtain the needed throughput. Since the BTT is only accessed 4 or more nibbles at a time it poses no problem. In operation, the entire BCIT is read or written, or a single nibble is read. When a single nibble is of interest, 4 nibbles are read from the BCIT and the desired nibble is selected by a multiplexor.

A timing diagram for the memory interface is shown in figure 17. Note: for highest throughput states S0 and S4 may be overlapped (i.e. S4 of transfer n is the same as S0 of transfer $n+1$).

5.2.1. BTT Interface. The interface of the BTT is shown in figure 15. Here are the interface signals for the BTT.

- *Address Lines* ($A_0 - A_5$). The address lines for the memory.
- *Refresh Address Lines* ($rA_0 - rA_5$). The address lines for the memory refresh address. Input from REFRESH_BLK/rA_0 – REFRESH_BLK/rA_5.
- *Address Select* (addr_sel). This selects which set of address lines is used. When it is high, the refresh address is used, otherwise the normal address is used. Input from BCIT/addr_sel and BTT/addr_sel.
- *Input Data lines* ($btt\_in_0 - btt\_in_{19}$). The value to be written into the memory. Input from SHREG/btt_out_0 - SHREG/btt_out_{19}.
- *Output Data lines* ($btt\_out_0 - btt\_out_{19}$). The value read from the memory. Output to SHREG/btt_in_0 - SHREG/btt_in_{19}.
- *Start Memory Cycle* (memrc) This line indicates that the memory should begin a read or write cycle. Input from MASTER_BLK/memrc + BLOCKMODE_BLK/memrc + REFRESH_BLK/memrc.
- *Precharge Clock* (prech) This clock precharges the dynamic logic in the memories.
- *Evaluate Clock* (eval) This clocks the various functions within the memories.
- *Clock* ($\phi 1$ and $\phi 2$) Two-phase, non-overlapping clock. Prech and eval are generated from these and memrc.
- *Input latch* (in_latch) Clock for input data latches.
- *Refresh Latch* (rfsh_latch) Clock for refresh data latches. These latches hold the word just read from memory in the refresh cycle. Input from REFRESH_BLK/rfsh_latch.
- *Write Cycle* (wr') The *read/*$\overline{write}$ line. This indicates which type of memory cycle to perform. When it is high, the memory is configured for reading. When it is low, the memory is configured for writing.

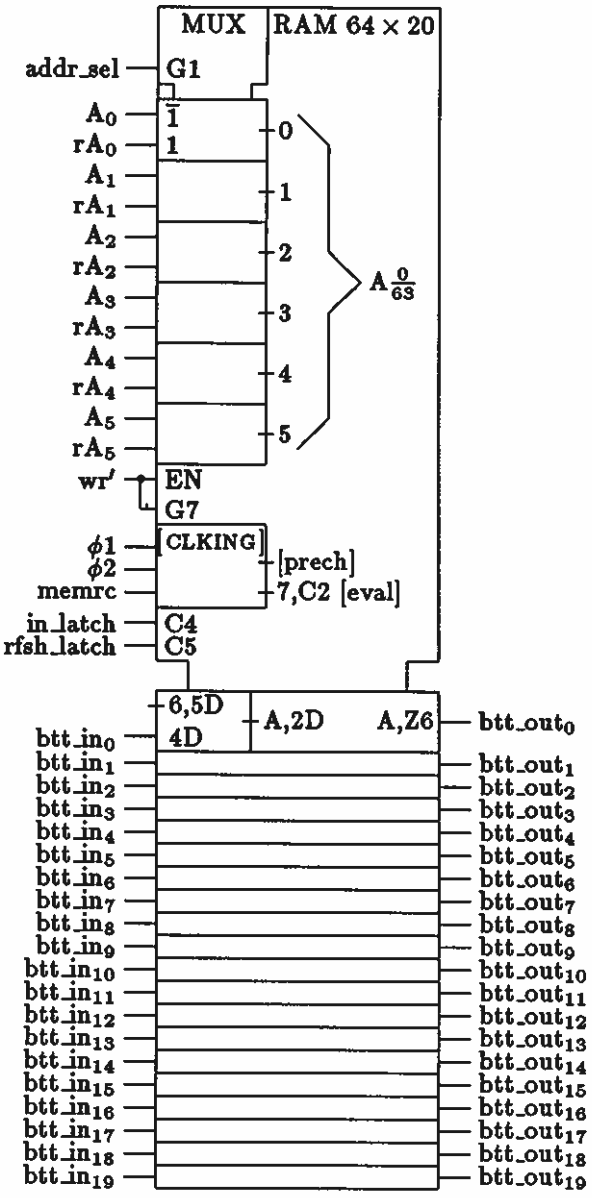


Figure 15: Broadcast Translation Table

5.2.2. BCIT Interface. The interface of the BCIT is shown in figure 16. Here are the interface signals for the BCIT.

- *Address Lines* ($A_0 - A_2$). The address lines for the memory.
- *Refresh Address Lines* ($rA_0 - rA_2$). Same as BTT.
- *Address Select* (addr_sel). Same as BTT. Input from BCIT/addr_sel.
- *Input Data lines* ($bcit\_in_0 - bcit\_in_{19}$). The value to be written into the memory. Input from $SHREG/bcit\_out_0$ - $SHREG/bcit\_out_{19}$.
- *Output Data lines* ($bcit\_out_0 - bcit\_out_{19}$). The value read from the memory. Output to $SHREG/bcit\_in_0$ - $SHREG/bcit\_in_{19}$.
- *Start Memory Cycle* (memrc) Same as BTT.
- *Precharge Clock* (prech) Same as BTT.
- *Evaluate Clock* (eval) Same as BTT.
- *Clock* ($\phi 1$ and $\phi 2$) Same as BTT.
- *Input latch* (in_latch) Same as BTT.
- *Refresh Latch* (rfsh_latch) Same as BTT.
- *Write Cycle* (wr') Same as BTT.

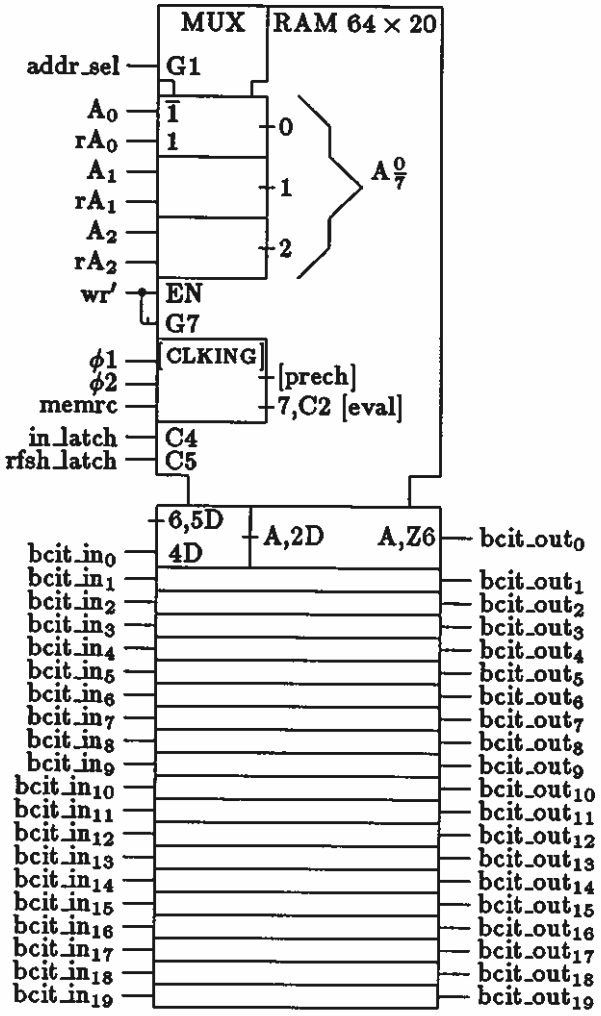


Figure 16: Broadcast Copy Index Table

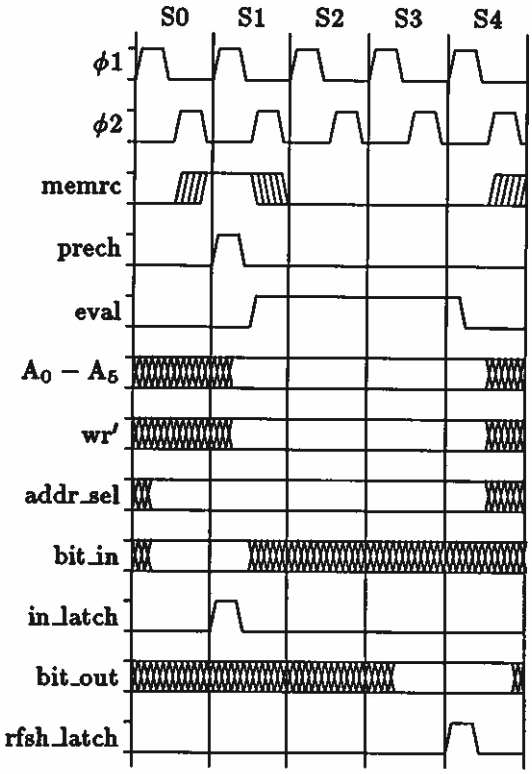


Figure 17: Memory Signal Timing

5.3. Control

This section contains a more detailed description of each of the control blocks mentioned previously.

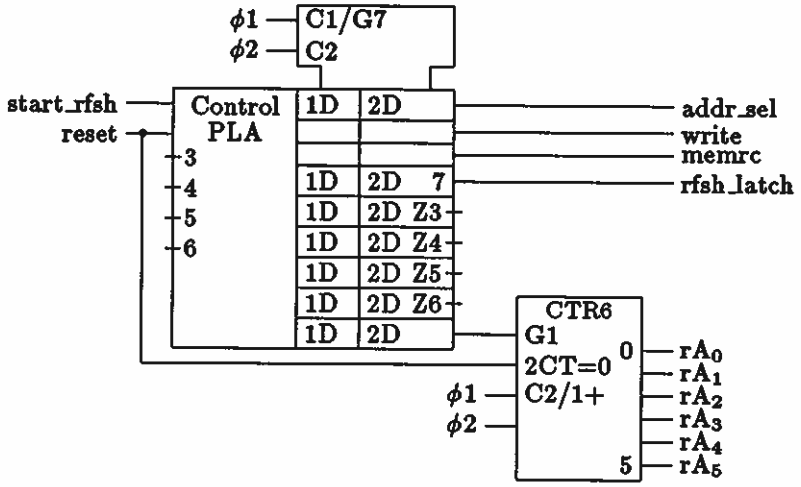


Figure 18: Memory Refresh Control Block

5.3.1. Memory Refresh Controller. A diagram of the refresh control block is shown in figure 18. This block controls the refresh for both the BTT and BCIT. Here are the interface signals. A timing diagram of them is shown in figure 19.

- *Start Refresh* (`start_rfsh`) This signal indicates that the refresh controller should begin a refresh cycle. Input from `MASTER_BLK/start_refresh`.
- *Master Reset* (`reset`) This is the master reset signal for the whole chip. It aborts any refresh cycle currently in progress and initializes the refresh controller.
- *Address Select* (`addr_sel`) This line is asserted when the two memories (BTT and BCIT) should use the address supplied by the refresh controller. Output to `BCIT/addr_sel` and `BTT/addr_sel`.
- *Write Enable* (`write`) This line indicates that the memory should treat this cycle as a write cycle. It is ORed with `BLOCKMODE_BLK/bcit_write` and output to `BCIT/wr'`, and it is ORed with `BLOCKMODE_BLK/btt_write` and `MASTER_BLK/btt_write` and output to `BTT/wr'`.
- *Start Memory Cycle* (`memrc`) This line indicates that the memory should begin a cycle. It is ORed with `MASTER_BLK/memrc` and `BLOCKMODE_BLK/memrc` and output to `BCIT/memrc` and `BTT/memrc`.
- *Refresh Latch* (`rfsh_latch`) This is the clock for the refresh data latches in the memory. It is output to `BCIT/rfsh_latch` and `BTT/rfsh_latch`.

The function of the refresh controller is to generate the signals for the refresh sequence upon receiving `start_rfsh`. It loops in state S0 until this signal is asserted. States S0 and S8 are overlapped if two refresh cycles immediately follow one another. In other words; upon leaving state S8, if `start_rfsh` is asserted, the next state will be S1.

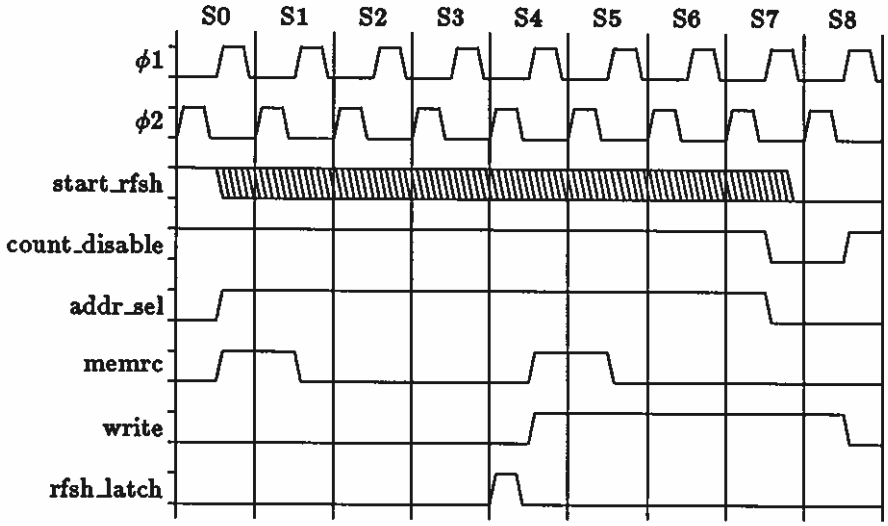


Figure 19: Refresh Control Signal Timing

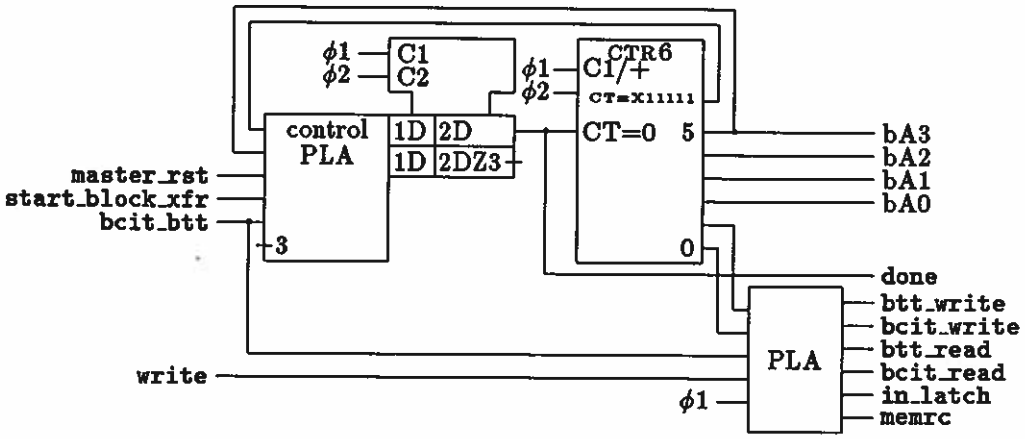


Figure 20: Block Transfer Control Block

5.3.2. Block Transfer Controller. A diagram of the block transfer controller is shown in figure 20. Here are the interface signals. A timing diagram of them is shown in figure 23.

- *Start Block Transfer* (`start_block_xfr`) This signal indicates that the block transfer controller should begin a block transfer. Input from `MASTER_BLK/start_block_xfr`.
- *Master Reset* (`reset`) This is the master reset signal for the whole chip. It aborts any block transfer currently in progress and initializes the refresh controller.
- *BCIT or BTT* (`bcit_btt`) This signal indicates which of the two memories (BCIT and BTT) should be involved in this transfer. When it is high, the BTT is selected, when it is low, the BCIT is selected. Input from `MASTER_BLK/bcit_btt_block`.
- *Transfer Direction* (`write`) This signal indicates whether the block transfer will be to or from the memory. Input from `MASTER_BLK/block_direction`.
- *Write BTT* (`btt_write`) This indicates that the BTT should treat this cycle as a write cycle. It is ORed with `MASTER_BLK/btt_write` and `REFRESH_BLK/write` and output to `BTT/wr'`.
- *Write BCIT* (`bcit_write`) This indicates that the BCIT should treat this cycle as a write cycle. It is ORed with `REFRESH_BLK/write` and output to `BCIT/wr'`.
- *Load Shift Register from BCIT* (`bcit_read`) This indicates that the shift register should load data from the BCIT in stages 4 through 7 instead of shifting on through. Output to `SHREG/read_bcit`.
- *Load Shift Register from BTT* (`btt_read`) This indicates that the shift register should load data from the BTT in stages 8 through 11 instead of shifting on through. Output to `SHREG/read_btt`.

- *Start memory cycle* (*memrc*) This line indicates that the memory should begin a cycle. It is ORed with MASTER_BLK/memrc and REFRESH_BLK/memrc and output to BCIT/memrc and BTT/memrc.
- *Input Data Latch* (in_latch) This line latches the data to be stored in memory from the shift register. Output to BCIT/in_latch and BTT/in_latch.
- *Block Transfer Done* (block_done) This line is asserted when the block transfer controller has completed the block transfer requested. Output to MASTER_BLK/block_done and ADDRESS_MUX/select.
- *Block Transfer Address* (bA0 - bA3) These are the 4 low order address lines for block transfers. For transfers to and from the BTT, the high order 2 address lines are taken from I[0].

The function of the block mode controller is to generate the signals for the block transfers to and from the BTC memory. It loops in state Sw until start_block_xfr is detected. It then performs 8 or 16 memory transfers and asserts done upon completion. The algorithm is shown in figure 21. The logic equations for the output signals are shown in figure 22.

```
integer i;
do start_block_xfr = 0 →
    wait;
od
i := 0;
Negate done;
if (bcit_btt = 0 ∧ write = 1) →
    Assert btt_write;
| (bcit_btt = 1 ∧ write = 1) →
    Assert bcit_write;
fi
do →
    if write = 0 →
        Strobe memrc and in_latch;
        wait;
        wait;
        wait;
    | write = 1 →
        wait;
        wait;
        wait;
        if bcit_btt = 0 →
            Assert btt_read;
        | bcit_btt = 1 →
            Assert bcit_read;
        fi
    fi
until (i = 7 ∧ bcit_btt = 1) ∨ (i = 15 ∧ bcit_btt = 0) od
if (bcit_btt = 0 ∧ write = 1) →
    Negate btt_write;
| (bcit_btt = 1 ∧ write = 1) →
    Negate bcit_write;
fi
Assert done;
```

Figure 21: Blockmode Controller Algorithm

$$
\begin{aligned}
\texttt{btt\_write} &= \overline{\texttt{done}} \wedge \texttt{write} \wedge \overline{\texttt{bcit\_btt}} \\
\texttt{bcit\_write} &= \overline{\texttt{done}} \wedge \texttt{write} \wedge \texttt{bcit\_btt} \\
\texttt{btt\_read} &= \overline{\texttt{done}} \wedge \overline{\texttt{write}} \wedge \overline{\texttt{bcit\_btt}} \wedge \texttt{counter}_0 \wedge \texttt{counter}_1 \\
\texttt{bcit\_read} &= \overline{\texttt{done}} \wedge \overline{\texttt{write}} \wedge \texttt{bcit\_btt} \wedge \texttt{counter}_0 \wedge \texttt{counter}_1 \\
\texttt{in\_latch} &= \overline{\texttt{done}} \wedge \texttt{write} \wedge \overline{\texttt{counter}_0} \wedge \overline{\texttt{counter}_1} \wedge \phi 1 \\
\texttt{memrc} &= \overline{\texttt{done}} \wedge \overline{\texttt{counter}_0} \wedge \overline{\texttt{counter}_1}
\end{aligned}
$$

Figure 22: Blockmode Controller Output Signals

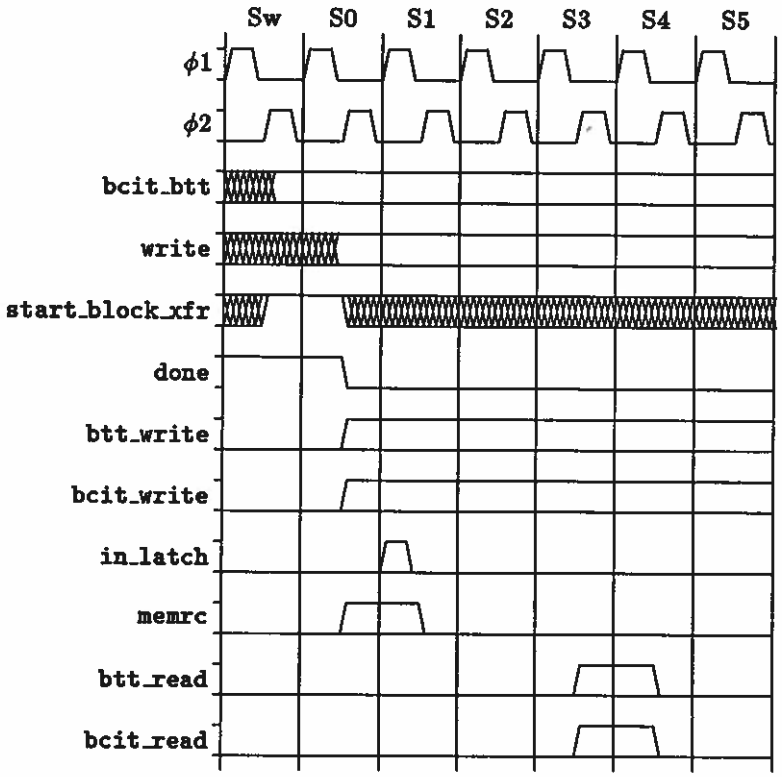


Figure 23: Blockmode Controller Signal Timing

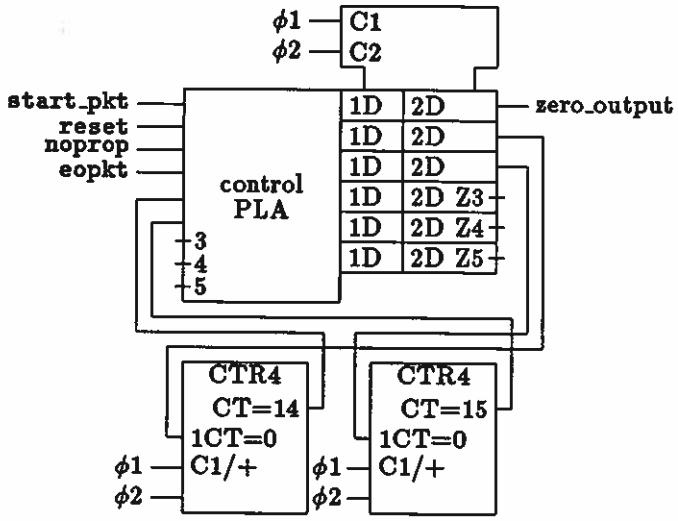


Figure 24: Packet Propagation Control Block

5.3.3. Packet Propagation Controller. A diagram of the Packet Propagation Controller is shown in figure 24. It keeps track of whether there is a packet in the shift register, where it is, and whether to propagate it or not. The Packet Propagation Controller is in one of 5 states depending on the shift registers occupancy. These are shown in figure 25. The transitions between states are shown in figure 26. Here are the interface signals.

- *Start Packet* (start_pkt) This signal indicates that a packet is about to enter the main shift register. Input from INPUT/pt.
- *Master Reset* (reset) This is the master reset signal for the whole chip. It stops propagation of any packet currently in the shift register and resets the Packet Propagation Controller to having no packet in the shift register.
- *Don't Propagate This Packet* (noprop) This input indicates that the packet currently in (at the front of) the main shift register should not be propagated. Input from MASTER_BLK/noprop.
- *End of Packet* (eopkt) This signal indicates that the current packet's end is at the beginning of the main shift register. Input from MASTER_BLK/eopkt.
- *Don't Propagate Output* (zero_output) This output tells the main shift register to output zero nibbles instead of whatever is coming through it. Output to SHREG/propagate.

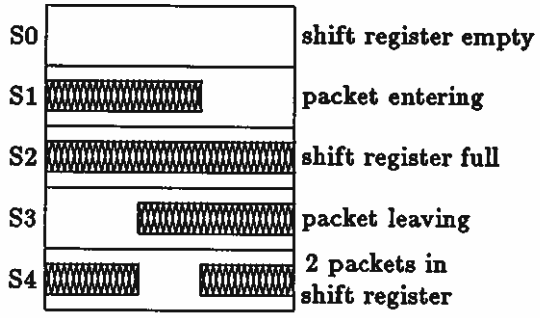


Figure 25: States of Packet Propagation Controller

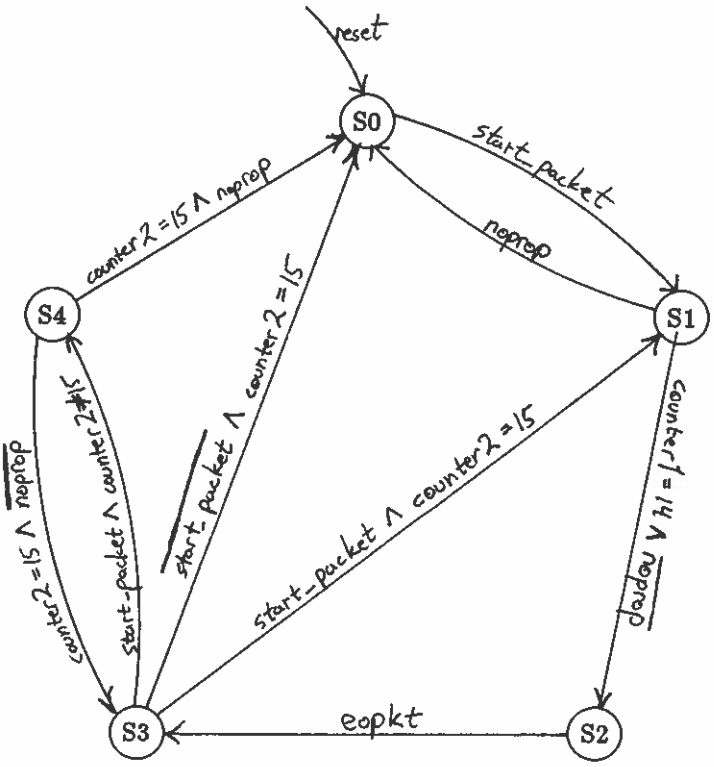


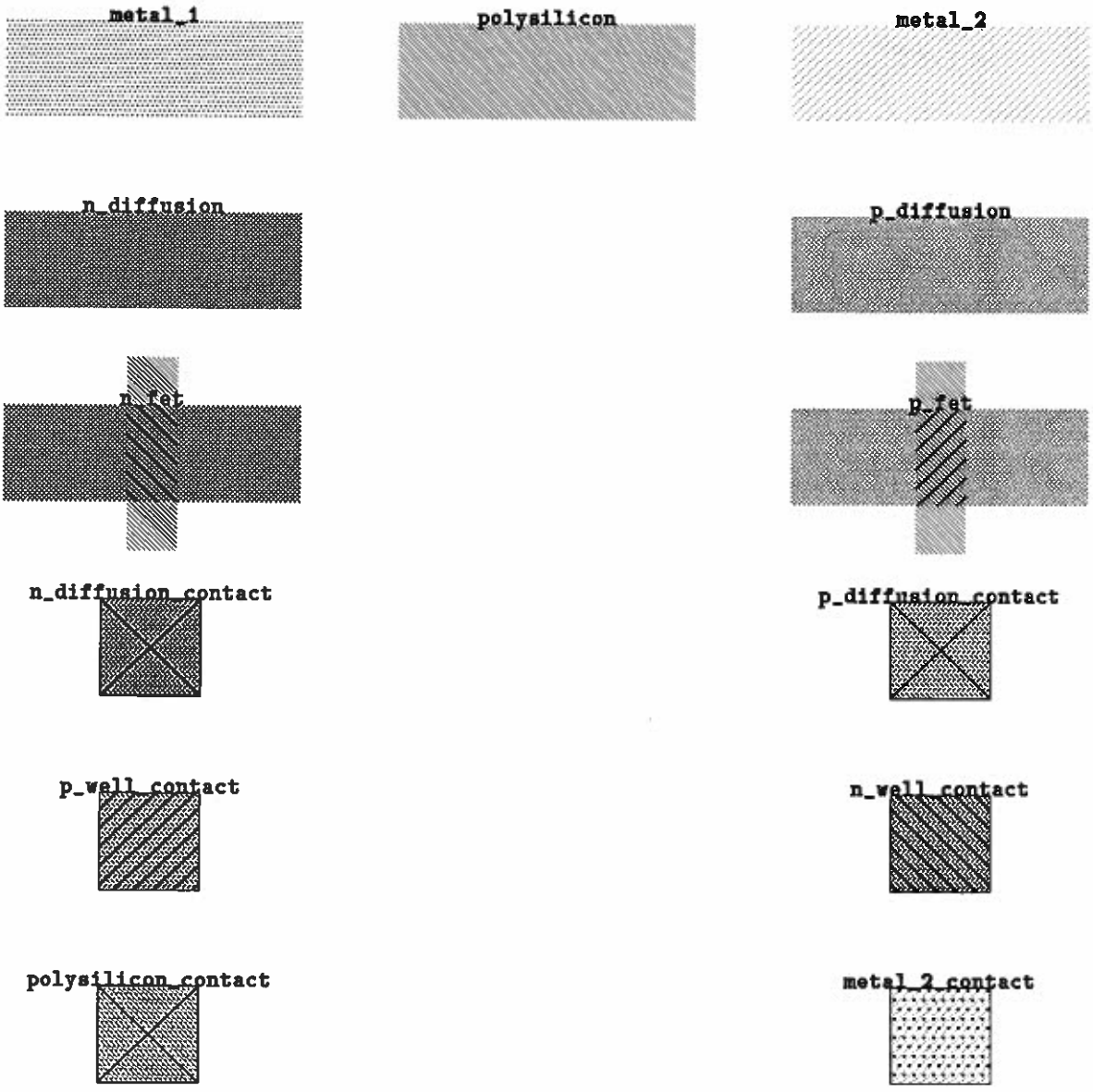
Figure 26: State Machine for Packet Propagation Controller

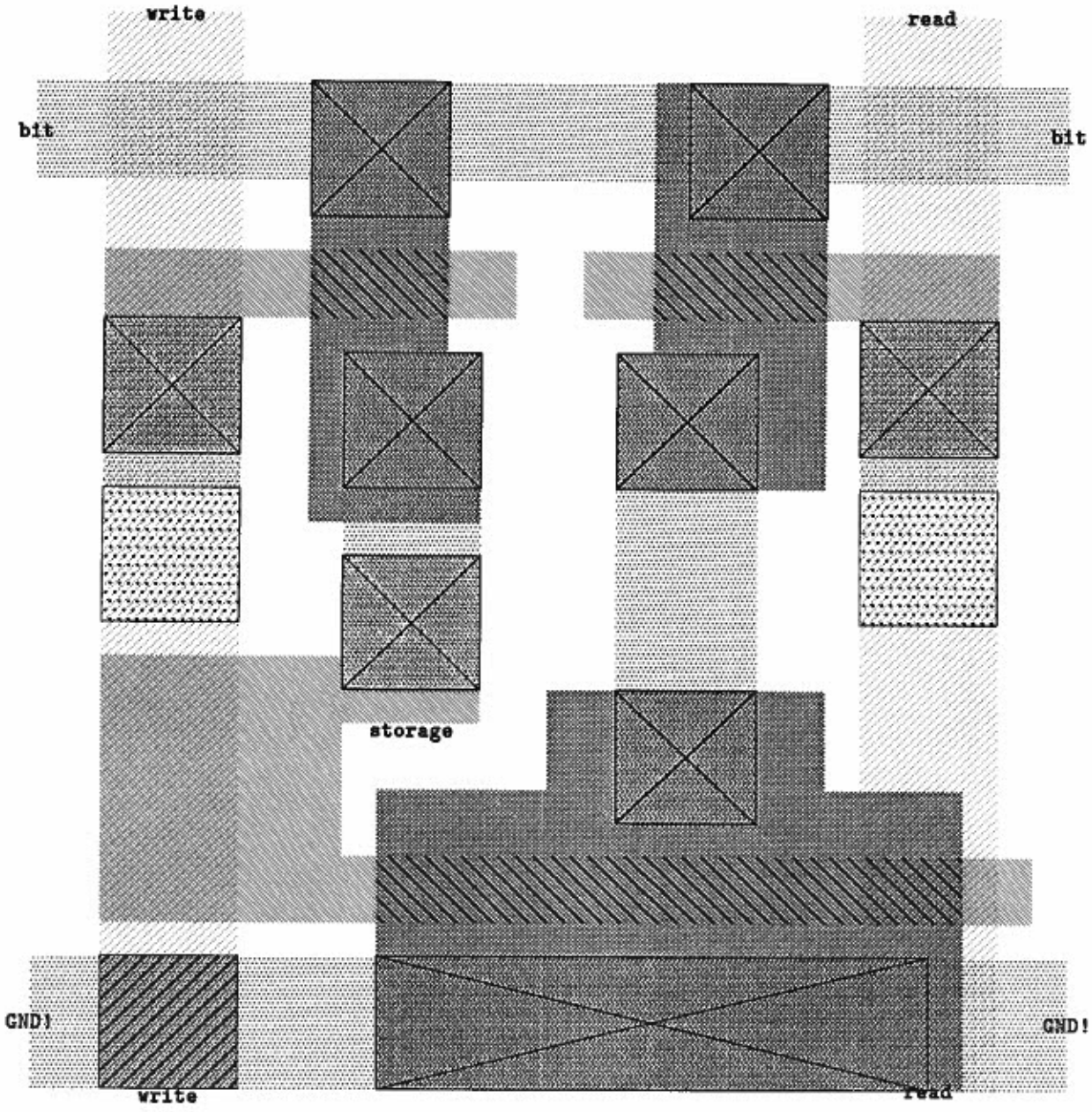
5.3.4. Master Controller. A diagram of the Master Control Block is not shown in a Figure yet. The Master Control Block co-ordinates the actions of the other control blocks. It also handles simple broadcast translation and single entry BTT updates. Here are the interface signals.

- *Master Reset* (reset). This is the master reset signal for the whole chip. It aborts any action in progress and resets all controllers.
- *Start Refresh Cycle* (start_refresh). This signal tells the memory refresh controller to begin a refresh cycle.
- *Start Block Transfer* (start_blk_xfr). This signal tells the block transfer controller to begin a block transfer.
- *BCIT block or BTT block* (bcit_btt_block). This signal tells the block transfer controller which of the two memories is involved in the block transfer.
- *Block Transfer Direction* (block_direction). This signal tells the block transfer controller in which direction (to or from memory) the block transfer will be.
- *Block Transfer Done* (block_done). This input from the block transfer controller tells that the requested block transfer is done.
- *Don't Propagate this Packet* (noprop). This signal tells the packet propagation controller not to propagate the current packet.
- *End of Packet* (eopkt). The master controller asserts this signal when the last nibble of a packet is at the beginning of the main shift register.
- *Down Counter = 0* (dncnt_0). This input is true when the down counter is equal to zero.
- *Enable Down Counter* (cnt_dn_en). This signal tells the down counter to decrement by 1 on the next clock cycle.
- *Load Down Counter* (load_bcit_ctr). This signal indicates that the down counter should be loaded from the output of the BCIT output mux.
- *Shift/Hold for Header Shift Register* (shift_header). This output indicates that the header shift register should shift the input data on through this cycle instead of holding it.
- *Start Memory Cycle* (memrc). This output indicates that the memory should begin a cycle. It is Ored with BLOCKMODE_BLK/memrc and REFRESH_BLK/memrc and output to BCIT/memrc and BTT/memrc.
- *Load Shift Register from BTT* (read_btt). This output indicates that the shift register should load data from the BTT in stages 8 through 11 instead of shifting on through. It is ORed with BLOCKMODE_BLK/btt_read and output to SHREG/read_btt.
- *BTT Input Data Latch* (btt_inlatch). This line latches the data to be stored in memory from the shift register. Output to BTT/in_latch.
- *Write BTT* (btt_write). This output indicates that the BTT should treat this cycle as a write cycle. It is ORed with BLOCKMODE_BLK/btt_write and REFRESH_BLK/write and output to BTT/wr'.

6. Cell Designs

This section provides manual pages for all of the low level cells used in the BTC. The manual page for each cell describes the cell's function and physical characteristics. Each cell also is documented with a layout and and circuit diagram. The stipple paterns used in the layouts are shown below.




write
read
bit
bit
storage
GND!
GND!
write
read

NAME

MEMCELL2 – 3-transistor RAM cell

SYNOPSIS

The basic 3-T RAM cell.

PROPERTIES

Size

$30 \times 30\lambda$

Interface

	label	name	layer	cap
inputs	read	read line	metal2	.022pf
	write	write line	metal2	.018pf
bidirectional	bit	bit line	metal1	.083pf

Simulation

esim

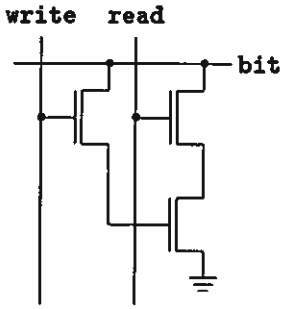
DESCRIPTION

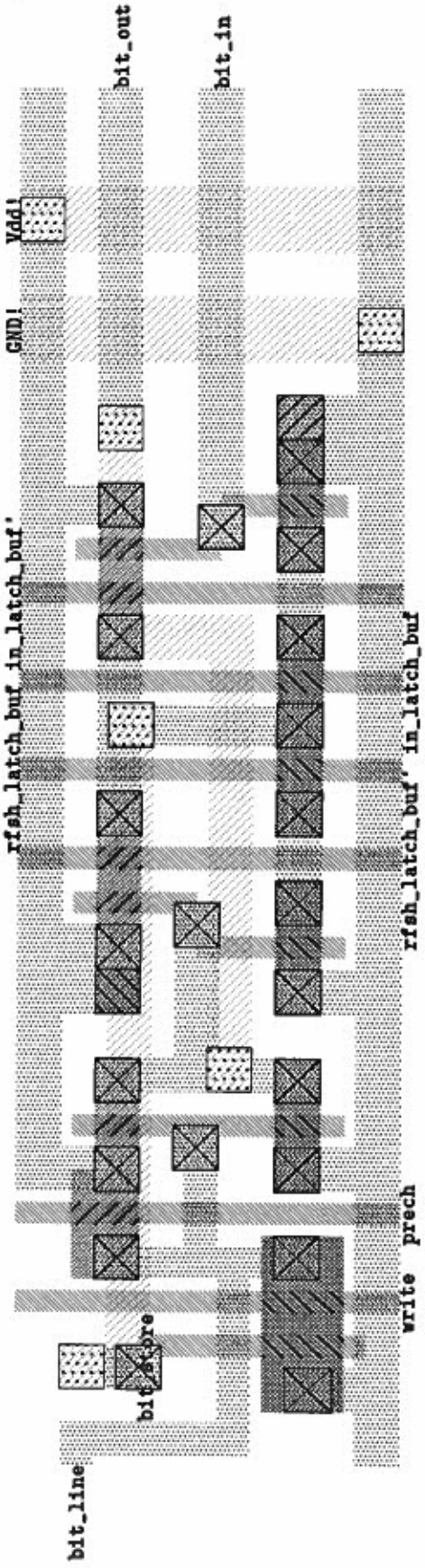
MEMCELL2 is a 3-T RAM cell. To write into the cell, bit is driven to the desired value and write is strobed. To read from the cell, bit is first precharged to Vdd. Then read is strobed. The complement of the value stored is placed on the bit. Since both reading and writing entail charge sharing between bit and the cell, bit's capacitance must be high enough to maintain good logic levels.

TESSELATION

Abuts horizontally. Forms mirrored pairs vertically, overlapping 4λ to share ground; pairs then abut vertically.

LOGIC DIAGRAM




bit_out
bit_in
Vdd!
GND!
rfsh_latch_buf_in_latch_buf'
rfsh_latch_buf' in_latch_buf
write
prech
bit_line

NAME

COL_DRIVE – column driver/sense amplifier

SYNOPSIS

The column driver, sense amplifier and refresh data latch for the dynamic memory.

PROPERTIES

Size

$125 \times 34\lambda$

Interface

	label	name	layer	cap
inputs	bit_in	data input	metal1	
	prech	precharge clock	polySi	
	write	write enable	polySi	
	rfsh_latch_buf	refresh latch clock	polySi	
	rfsh_latch_buf'	refresh latch clock	polySi	
	in_latch_buf	input latch clock	polySi	
	in_latch_buf'	input latch clock	polySi	
outputs	bit_out	data output	metal1	
bidirectional	bit_line	bit line	metal1	

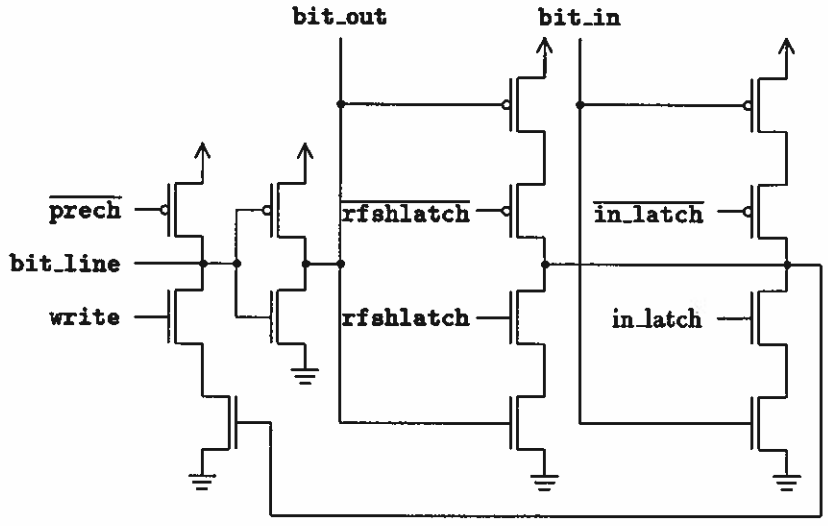
DESCRIPTION

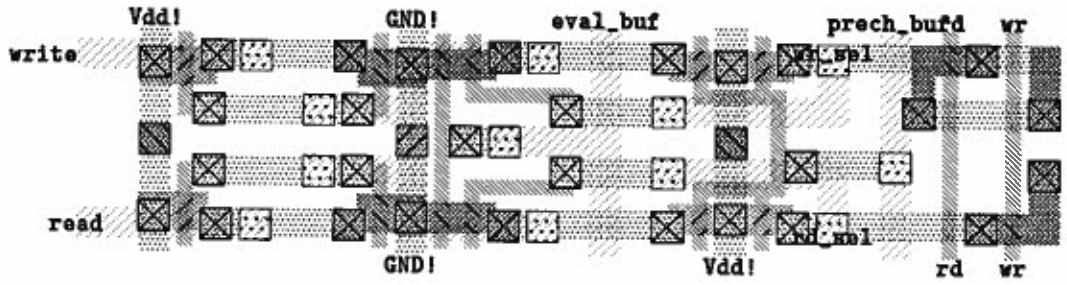
COL_DRIVE is the column driver and sense amplifier for the dynamic ram.

TESSELATION

Forms mirrored pairs vertically, overlapping 4λ to share ground; pairs then abut vertically.

LOGIC DIAGRAM



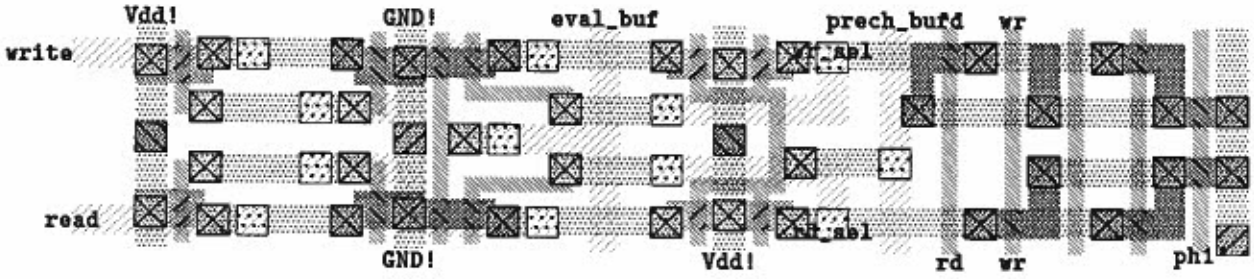


row_endriver

rowbit0

rowbit1

row_end



sample_row_driver

NAME

ROW_ENDRIVER – Row decoder/driver endpiece
ROWBIT0 – Row decoder low address bit
ROWBIT1 – Row decoder high address bit
ROW_END – Row decoder other endpiece

SYNOPSIS

Major part of dynamic NOR row decoder driver

PROPERTIES

Size

ROW_ENDRIVER	$126 \times 30\lambda$
ROWBIT0	$17 \times 30\lambda$
ROWBIT1	$17 \times 30\lambda$
ROW_END	$9 \times 30\lambda$

Interface

	label	name	layer	cap
inputs	rd	read input	polysilicon	.016pf
	wr	write input	polysilicon	.016pf
	an	address line	polysilicon	.026pf/.006pf
	an′	address line	polysilicon	.006pf/.026pf
	prech_buf′	precharge clock	metal2	.060pf
	eval_buf	evaluate clock	metal2	.032pf
outputs	read	read line	metal2	.099pf
	write	write line	metal2	.10pf

Timing

Simulation

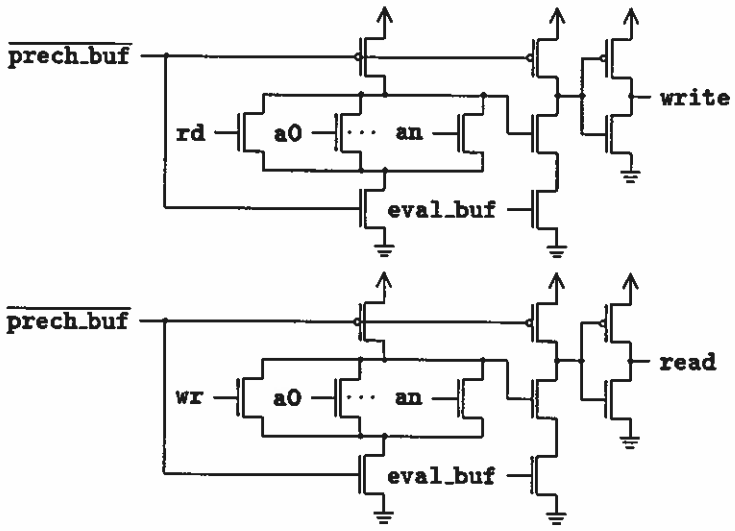
DESCRIPTION

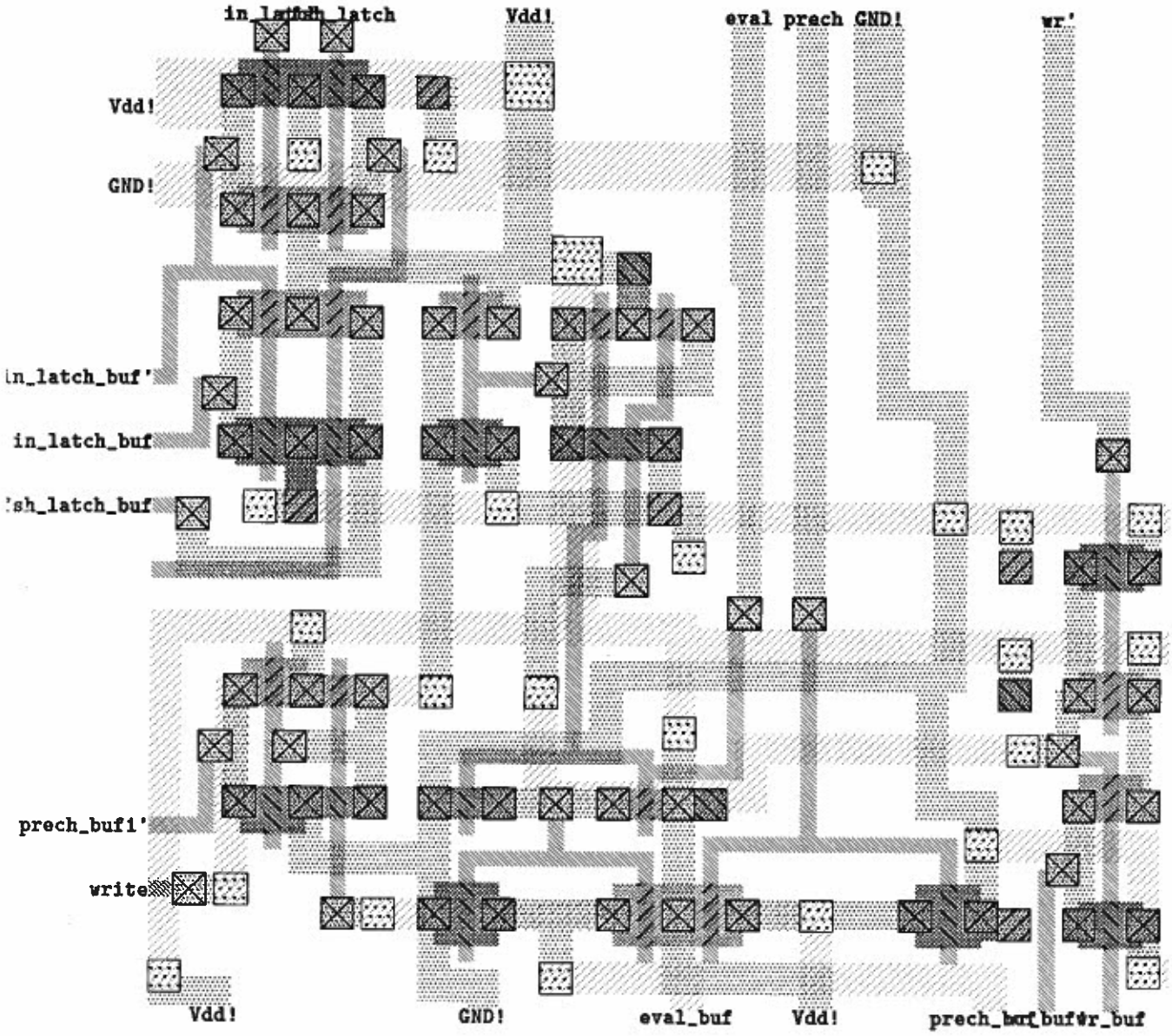
ROW_ENDRIVER is the basis for the row decoders and drivers of the dynamic memory. It uses a dynamic NOR structure for the address and read/write decoding. The NOR is precharged on phase prech and evaluated on phase eval. Read or write is selected by the pair of complementary signals rd and wr. The cells for selecting the address lines are ROWBIT0 and ROWBIT1 respectively. Also ROW_END handles the final pulling down of the NOR.

TESSELATION

All abut vertically.
ROW_ENDRIVER: Abuts horizontally with rowbit0 or rowbit1.
ROWBIT0, ROWBIT1: Abut horizontally with 1λ overlap.
ROW_END: Abuts horizontally with 1λ overlap with rowbit0 or rowbit1.

LOGIC DIAGRAM




Vdd!
eval
prech
GND!
wr'
Vdd!
GND!
in_latch_buf'
in_latch_buf
prech_buf1'
write
Vdd!
GND!
eval_buf
Vdd!

NAME

RADRV – Driver for row and column drivers

SYNOPSIS

memory row and column driver driver

PROPERTIES

Size

$125 \times 124\lambda$

Interface

	label	name	layer
inputs	prech	precharge clock	metal1
	eval	evaluate clock	metal1
	wr′	read/$\overline{\text{write}}$	metal1
	in_latch	input latch clock	polycontact
	rfsh_latch	refresh latch clock	polycontact
outputs	prech_buf′	precharge clock	metal2
	prech_buf1′	precharge clock	polySi
	eval_buf	evaluate clock	metal2
	rd_buf	read select	polySi
	wr_buf	write select	polySi
	write	write strobe	polySi
	rfsh_latch_buf	refresh latch clock	polySi
	rfsh_latch_buf′	refresh latch clock	polySi
	in_latch_buf	input latch clock	polySi
	in_latch_buf′	input latch clock	polySi

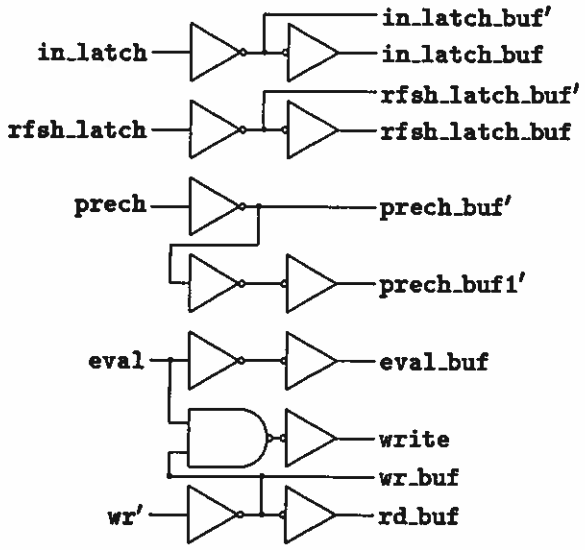
DESCRIPTION

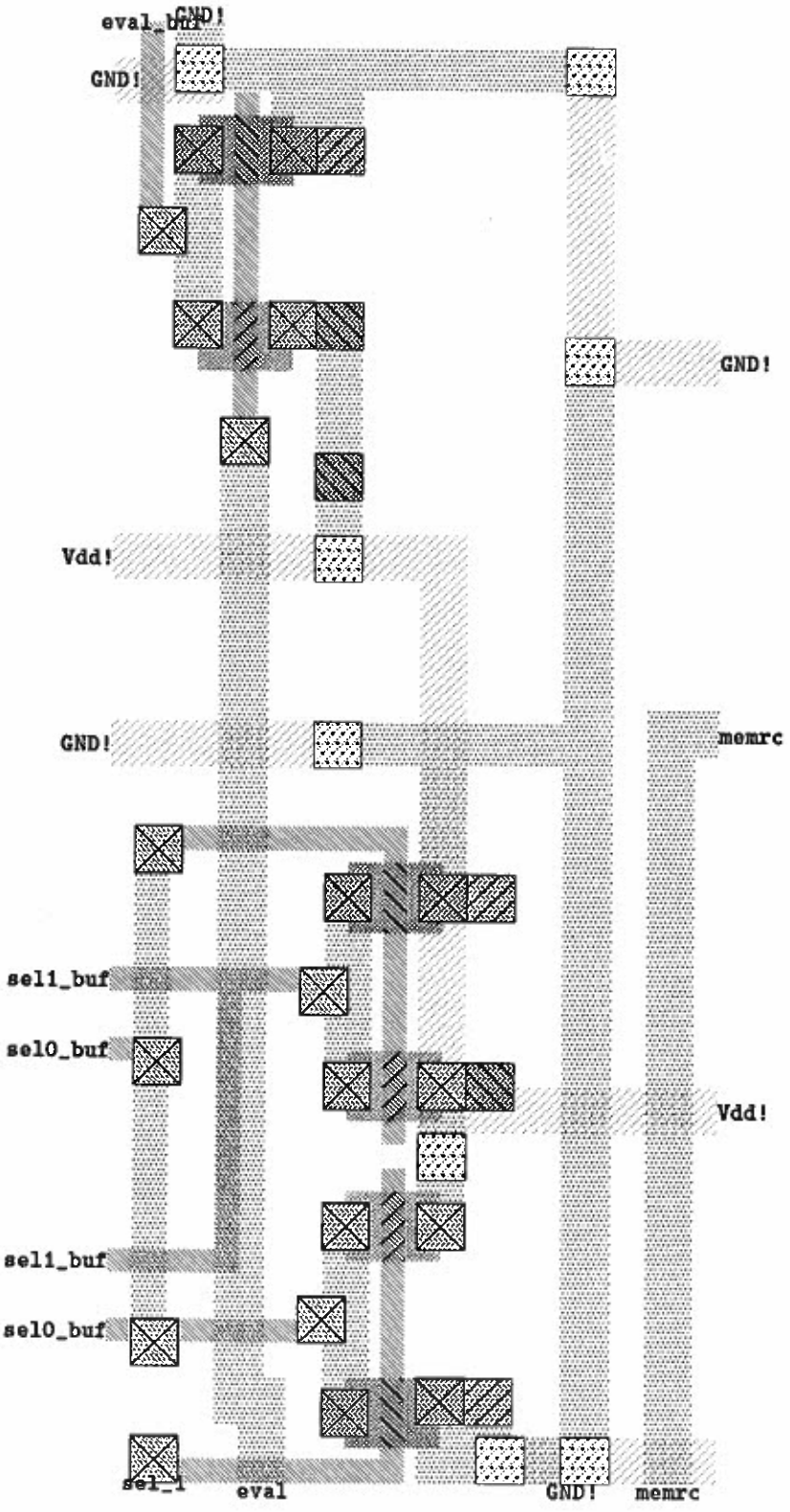
RADRV generates the control lines for the row decoder/drivers and the column drivers.

TESSELATION

Abuts horizontally to COL_DRIVE and ADDR_MUX

LOGIC DIAGRAM




eval_buf
GND!
GND!
GND!
Vdd!
GND!
memrc
sel1_buf
sel0_buf
Vdd!
sel1_buf
sel0_buf
sel_1
eval
GND!
memrc

NAME

MEMORDRV – Driver for memory address multiplexors

SYNOPSIS

memory address multiplexor driver

PROPERTIES

Size

$51 \times 125\lambda$

Interface

	label	name	layer
inputs	eval	evaluate clock	metal1
	sel_1	refesh address select	polycontact
outputs	eval_buf	evaluate clock	polySi
	sel1_buf	refresh address select	polySi
	sel0_buf	normal address select	polySi

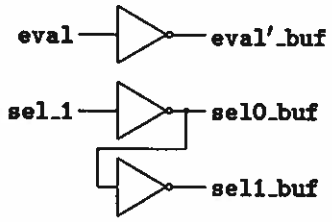
DESCRIPTION

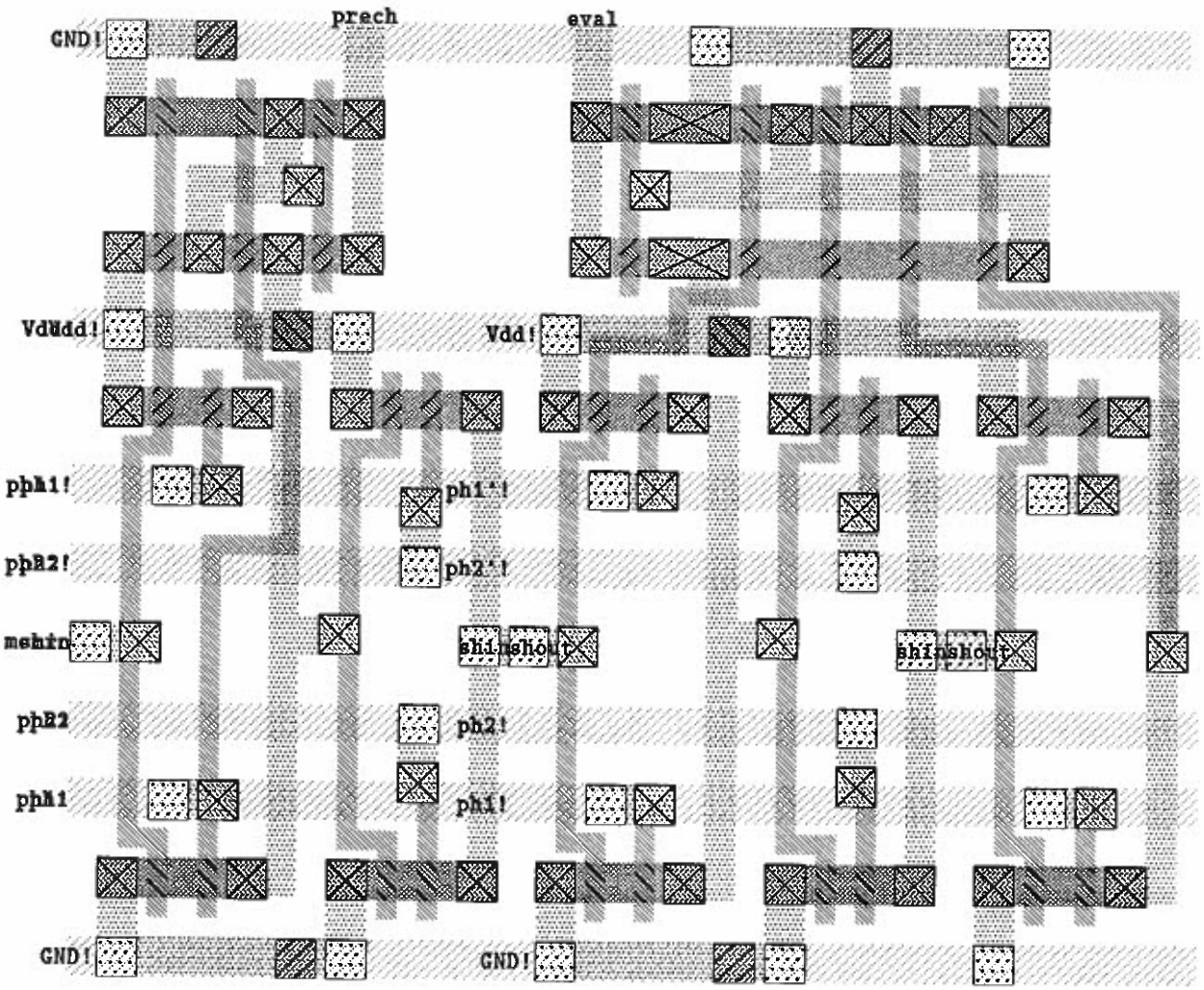
MEMORDRV generates the control lines for the refresh address multiplexor.

TESSELATION

Abuts horizontally to ADDR_MUX and MEMCG.

LOGIC DIAGRAM




prech
eval
GND!
Vdd!
Vdd!
ph1!
ph1*!
ph2!
ph2*!
ph2!
ph1!
GND!
GND!

NAME

MEMCG – Clock generator for dynamic memory

SYNOPSIS

memory clock generator

PROPERTIES

Size

$113 \times 98\lambda$

Interface

	label	name	layer
inputs	memrc	start memory cycle	metal2
	ph1	$\phi 1$ clock	metal2
	ph1′	$\overline{\phi 1}$ clock	metal2
	ph2	$\phi 2$ clock	metal2
	ph2′	$\overline{\phi 2}$ clock	metal2
outputs	prech	precharge clock	metal1
	eval	evaluate clock	metal1

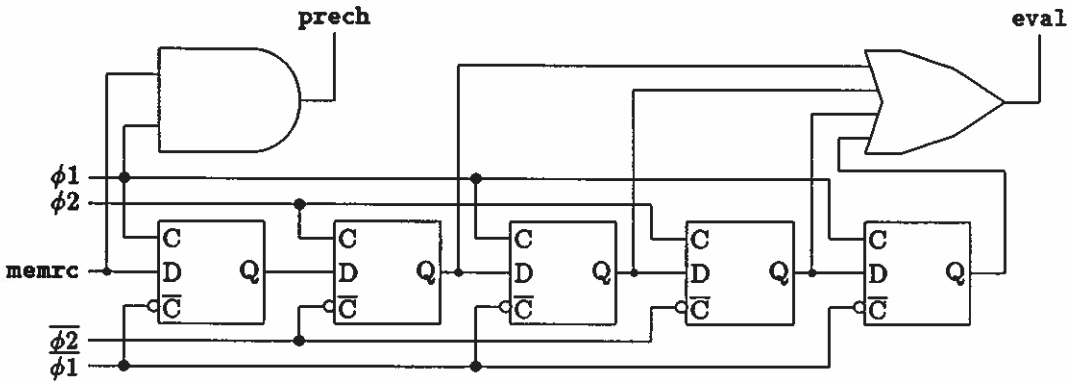
DESCRIPTION

MEMCG[2] generates the two phase asymetric clock for the memory from the system-wide clock. One memory clock cycle takes 4 system clock cycles. It may be stretched by holding memrc high for more that one system clock cycle.

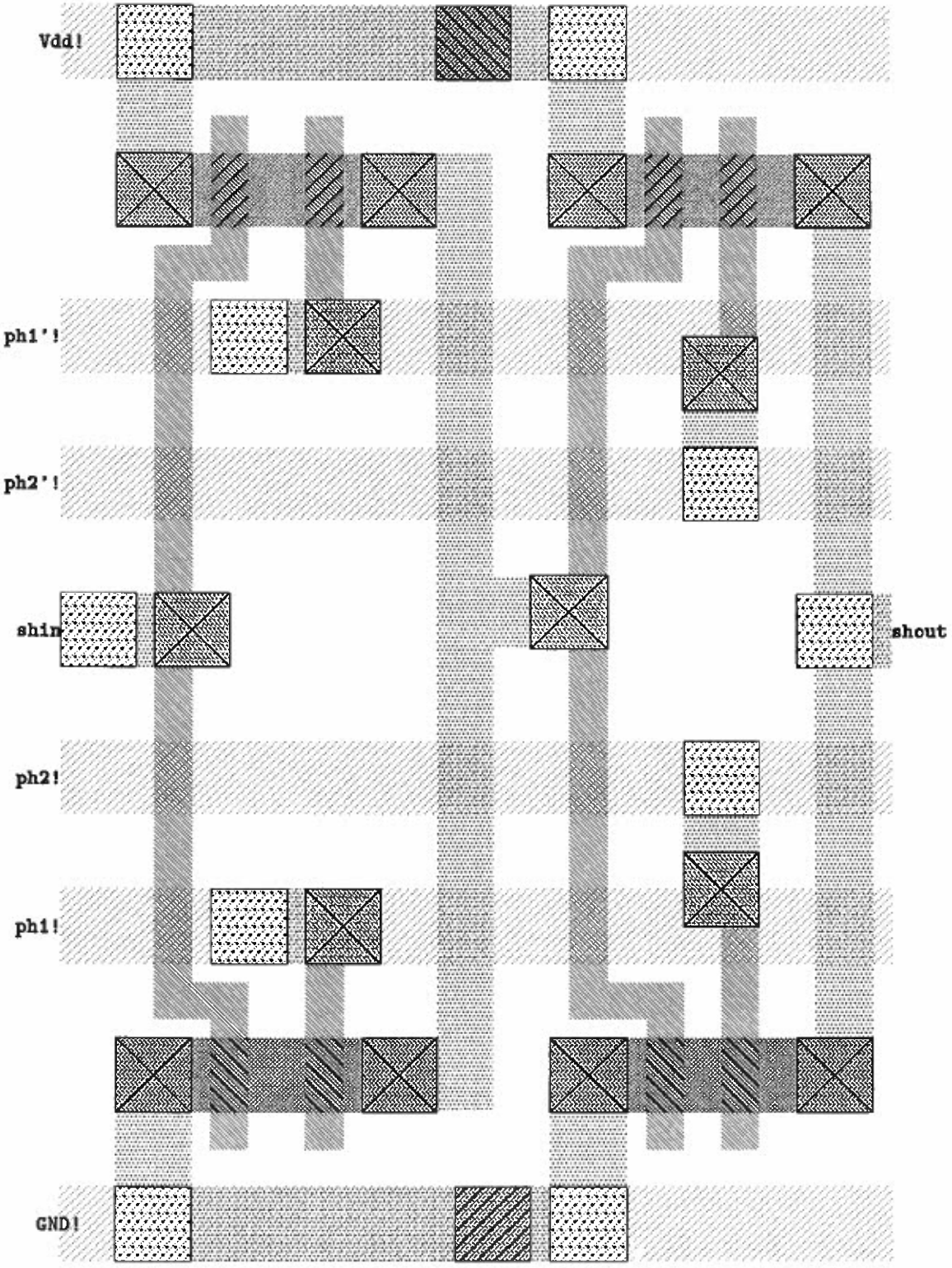
TESSELATION

Abuts horizontally to MEMORDRV

LOGIC DIAGRAM



[2]this cell's layout needs to be redone


Vdd!
ph1'!
ph2'!
shin
shout
ph2!
ph1!
GND!

NAME

SHREG1 – basic shift register cell

SYNOPSIS

Dynamic shift register cell. (One bit)

PROPERTIES

Size

$44 \times 68\lambda$

Interface

	label	name	layer	cap
inputs	shin	data in	metal2	
	shout	data out	metal2	
	ph1	$\phi 1$ clock	metal2	
	ph1′	$\overline{\phi 1}$ clock	metal2	
	ph2	$\phi 2$ clock	metal2	
	ph2′	$\overline{\phi 2}$ clock	metal2	
output	shout	data out	metal2	

DESCRIPTION

SHREG1 is a dynamic shift register cell. On $\phi 1$ the data (shin) is latched in. On $\phi 2$ the input value is transfered to the output (shout).

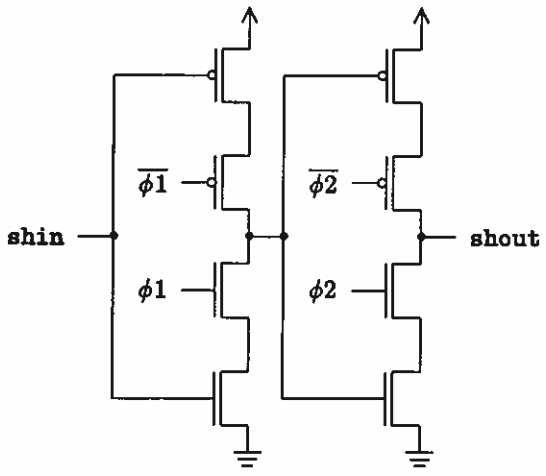
TESSELATION

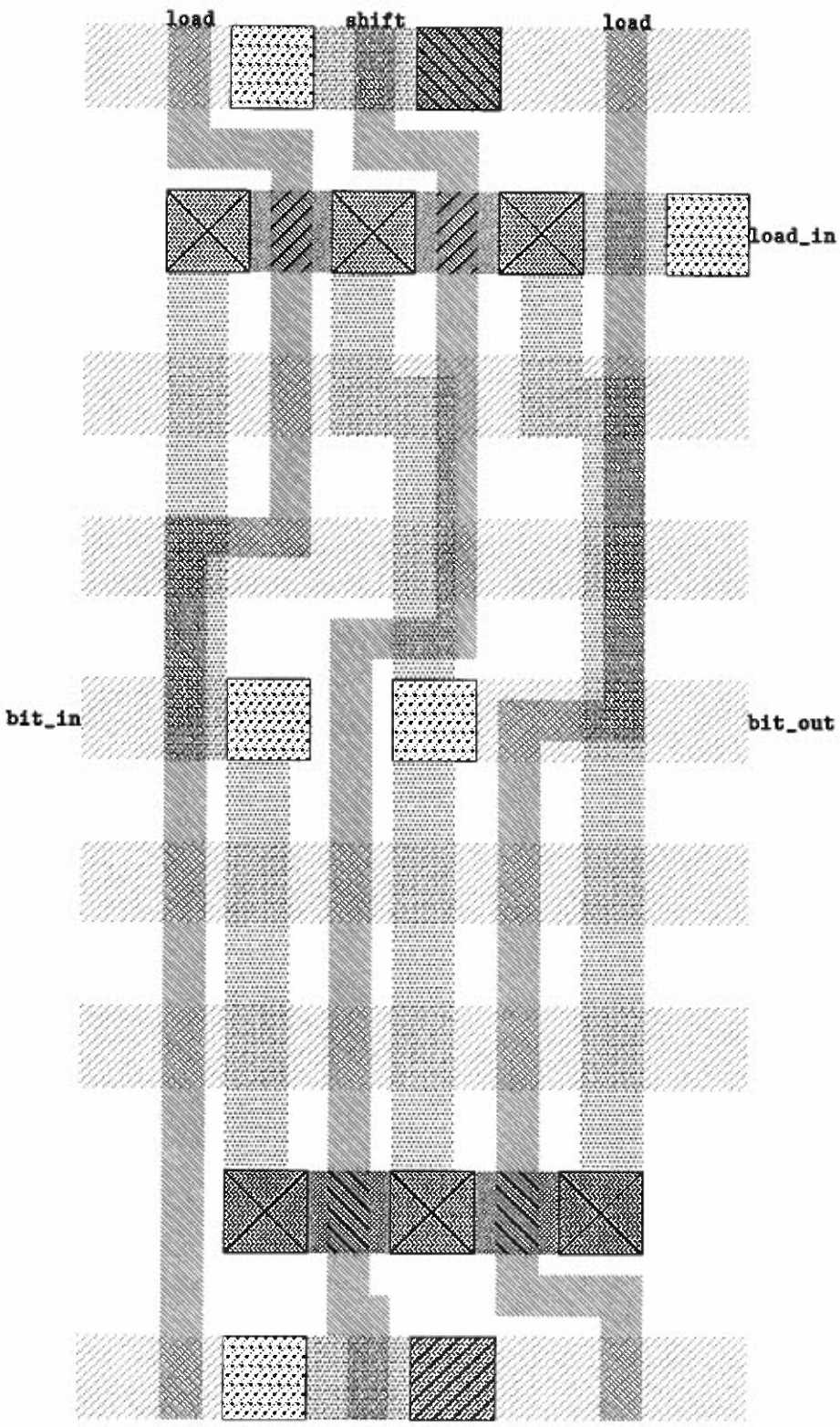
Abuts horizontally. Forms mirrored pairs vertically, overlapping 4λ to share power and ground.

SEE ALSO

SHREG2, SHREG3

LOGIC DIAGRAM




load
shift
load
load_in
bit_in
bit_out

NAME

SHREG2 – Shift register input multiplexor

SYNOPSIS

Input multiplexor for dynamic shift register

PROPERTIES

Size

$32 \times 68\lambda$

Interface

	label	name	layer	cap
inputs	bit_in	shift data in	metal2	
	load_in	load data in	via	
	load	load enable	polySi	
	shift	shift enable	polySi	
outputs	bit_out	data output	metal2	

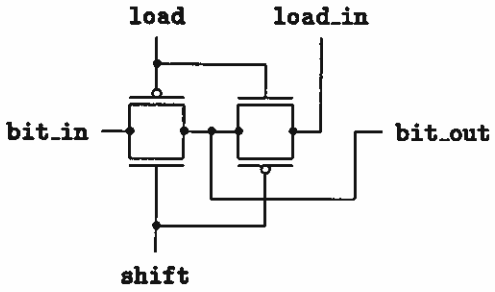
DESCRIPTION

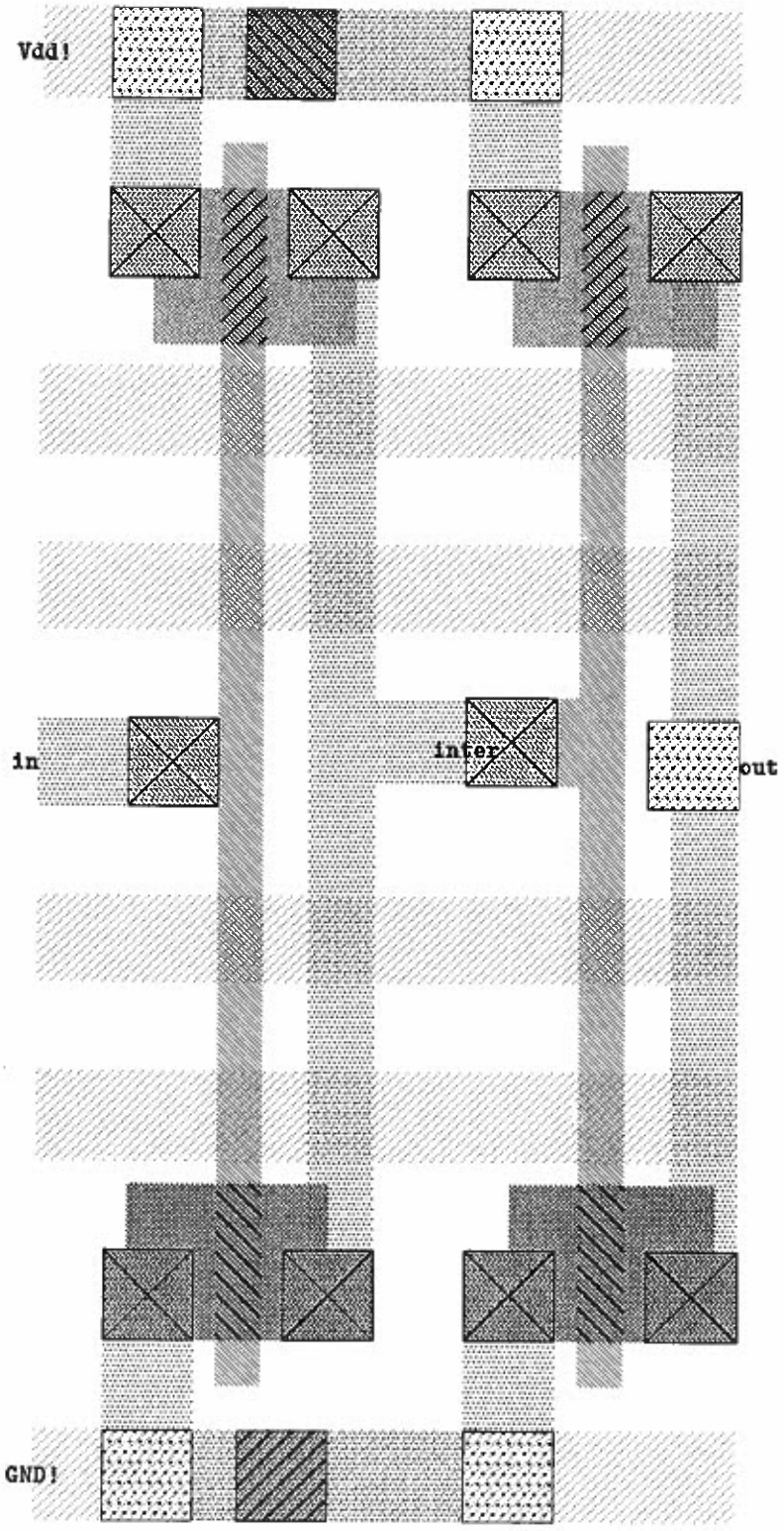
SHREG2 is an input multiplexor for the dynamic shift register cell SHREG1. The shift and load inputs should be complementary.

TESSELATION

Abuts horizontally. Forms mirrored pairs vertically, overlapping 4λ to share power and ground.

LOGIC DIAGRAM




Vdd!
in
inter
out
GND!

NAME

SHREG3 – Shift register output driver

SYNOPSIS

Output driver for dynamic shift register

PROPERTIES

Size

$32 \times 68\lambda$

Interface

	label	name	layer	cap
inputs	in	data in	metal2	
outputs	out	data out	metal2	

DESCRIPTION

SHREG3 is an output buffer for the dynamic shift register cell SHREG1.

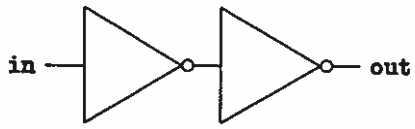
TESSELATION

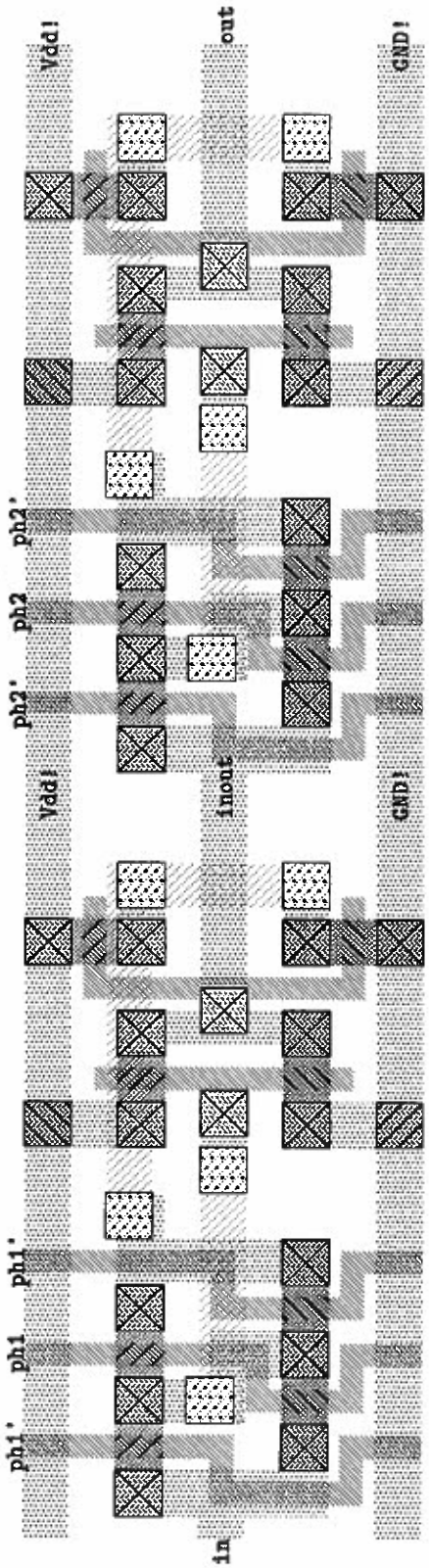
Abuts horizontally. Forms mirrored pairs vertically, overlapping 4λ to share power and ground.

SEE ALSO

SHREG1, SHREG2

LOGIC DIAGRAM




Vdd!
out
GND!
Ph2'
Ph2
Ph2'
Vdd!
inout
GND!
Ph1'
Ph1
Ph1'
in

NAME

Dd_flip – quasi-static D flip-flop

SYNOPSIS

Quasi-static D flip-flop

PROPERTIES

Size

$130 \times 34\lambda$

Interface

	label	name	layer	cap
inputs	in	data in	metal	
	ph1	$\phi 1$ clock	polySi	
	ph1′	$\overline{\phi 1}$ clock	polySi	
	ph2	$\phi 2$ clock	polySi	
	ph2′	$\overline{\phi 2}$ clock	polySi	
outputs	out	data out	metal	

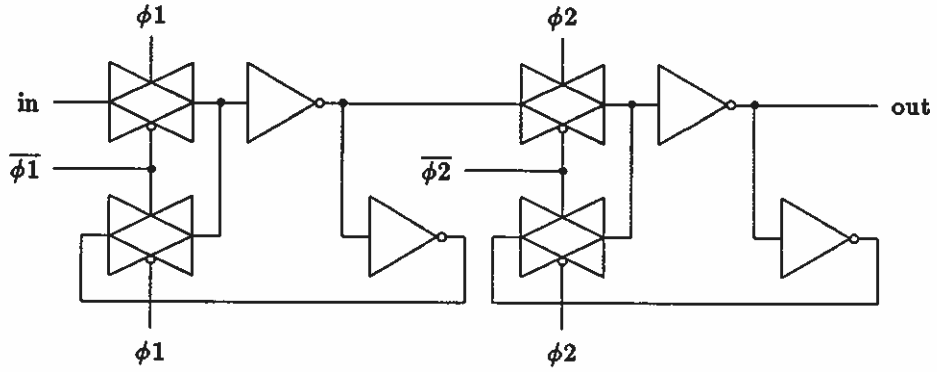
DESCRIPTION

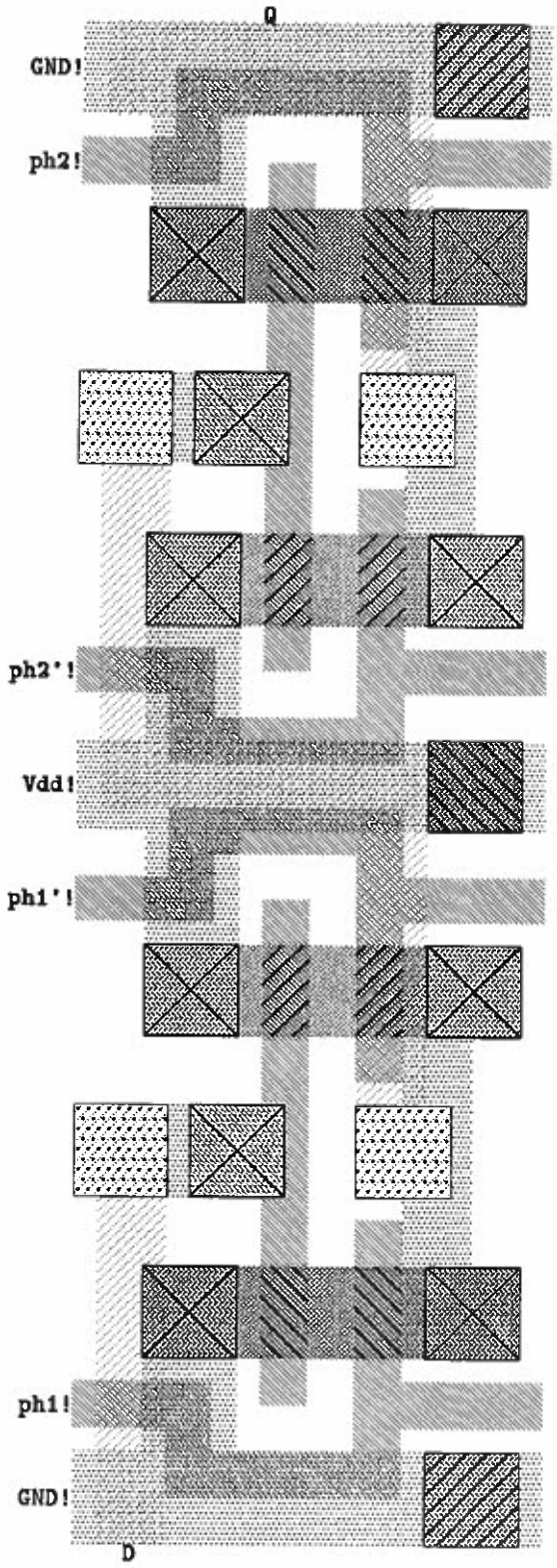
DD_FLIP is a quasi-static D flip-flop. On $\phi 1$ the data (in) is latched in. On $\phi 2$ the input value is transfered to the output (out).

TESSELATION

Abuts horizontally. Forms mirrored pairs vertically, overlapping 4λ to share power and ground.

LOGIC DIAGRAM




Q
GND!
ph2!
ph2'!
Vdd!
ph1'!
ph1!
GND!
D

NAME

LATCH – Dynamic 2 phase latch

SYNOPSIS

Dynamic storage element. (One bit)

PROPERTIES

Size

$20 \times 66\lambda$

Interface

	label	name	layer	cap
inputs	D	data in	metal2	
	ph1	$\phi 1$ clock	polySi	
	ph1′	$\overline{\phi 1}$ clock	polySi	
	ph2	$\phi 2$ clock	polySi	
	ph2′	$\overline{\phi 2}$ clock	polySi	
output	Q	data out	metal2	

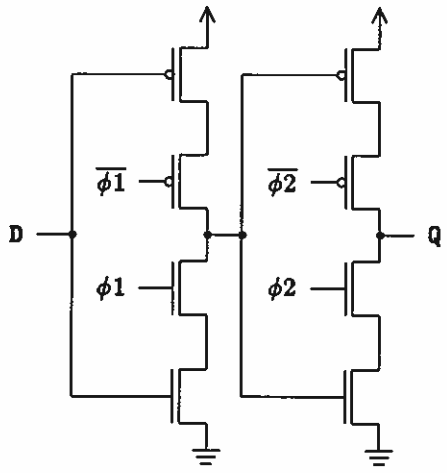
DESCRIPTION

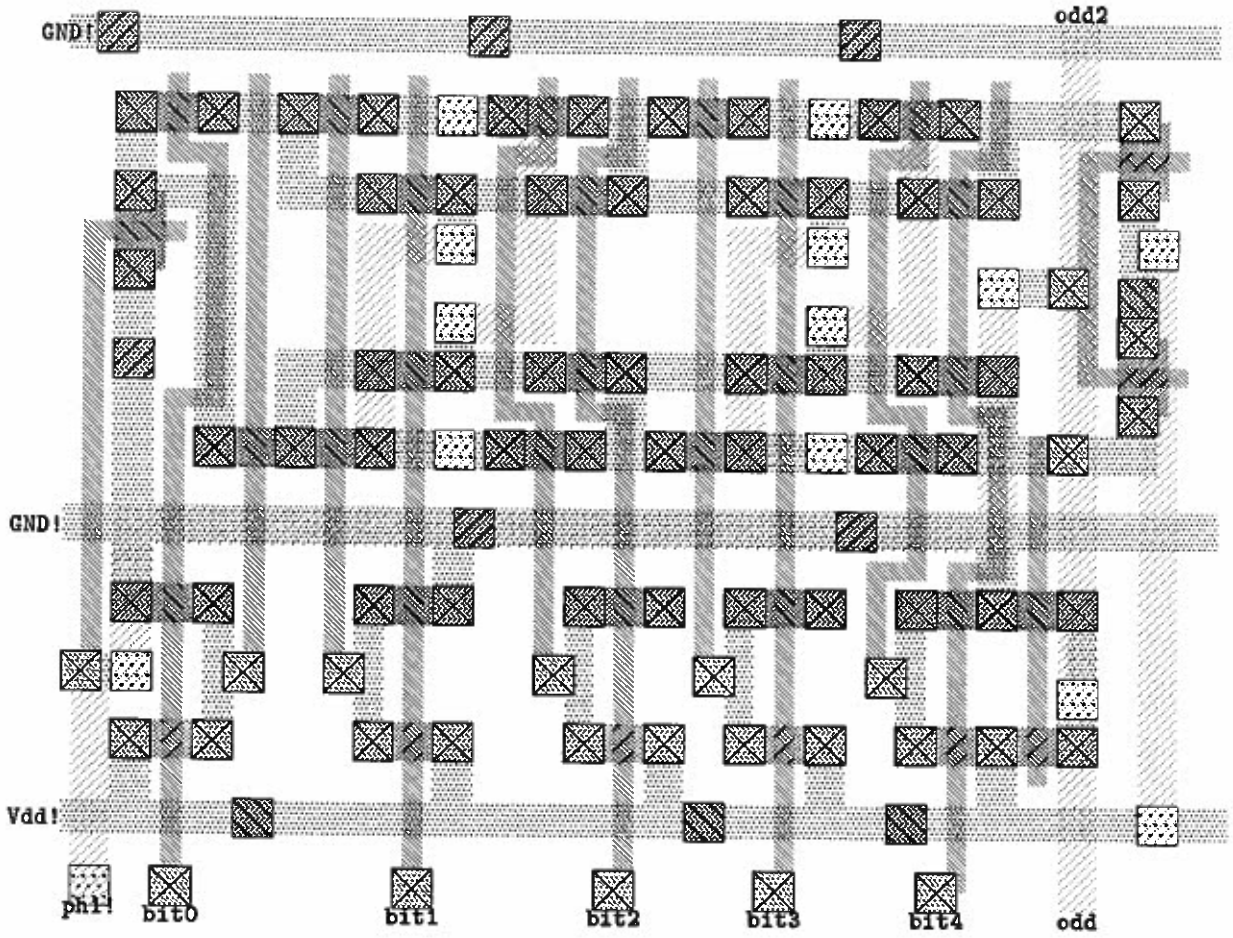
LATCH is a 2 phase dynamic register cell. On $\phi 1$ the data (D) is latched in. On $\phi 2$ the input value is transfered to the output (D).

TESSELATION

Abuts horizontally and vertically.

LOGIC DIAGRAM




GND!
odd2
GND!
Vdd!
ph1!
bit0
bit1
bit2
bit3
bit4
odd

NAME

PARITY – 5 bit parity checker.

SYNOPSIS

Five bit dynamic odd parity checker.

PROPERTIES

Size

$116 \times 91\lambda$

Interface

	label	name	layer	cap
inputs	bit[43210]	data in	polycontact	
	ph1	$\phi 1$ clock	via	
output	odd	odd parity	metal2	

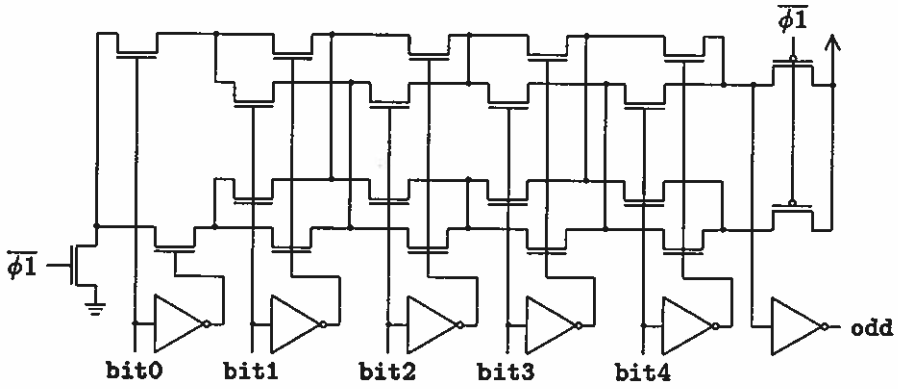
DESCRIPTION

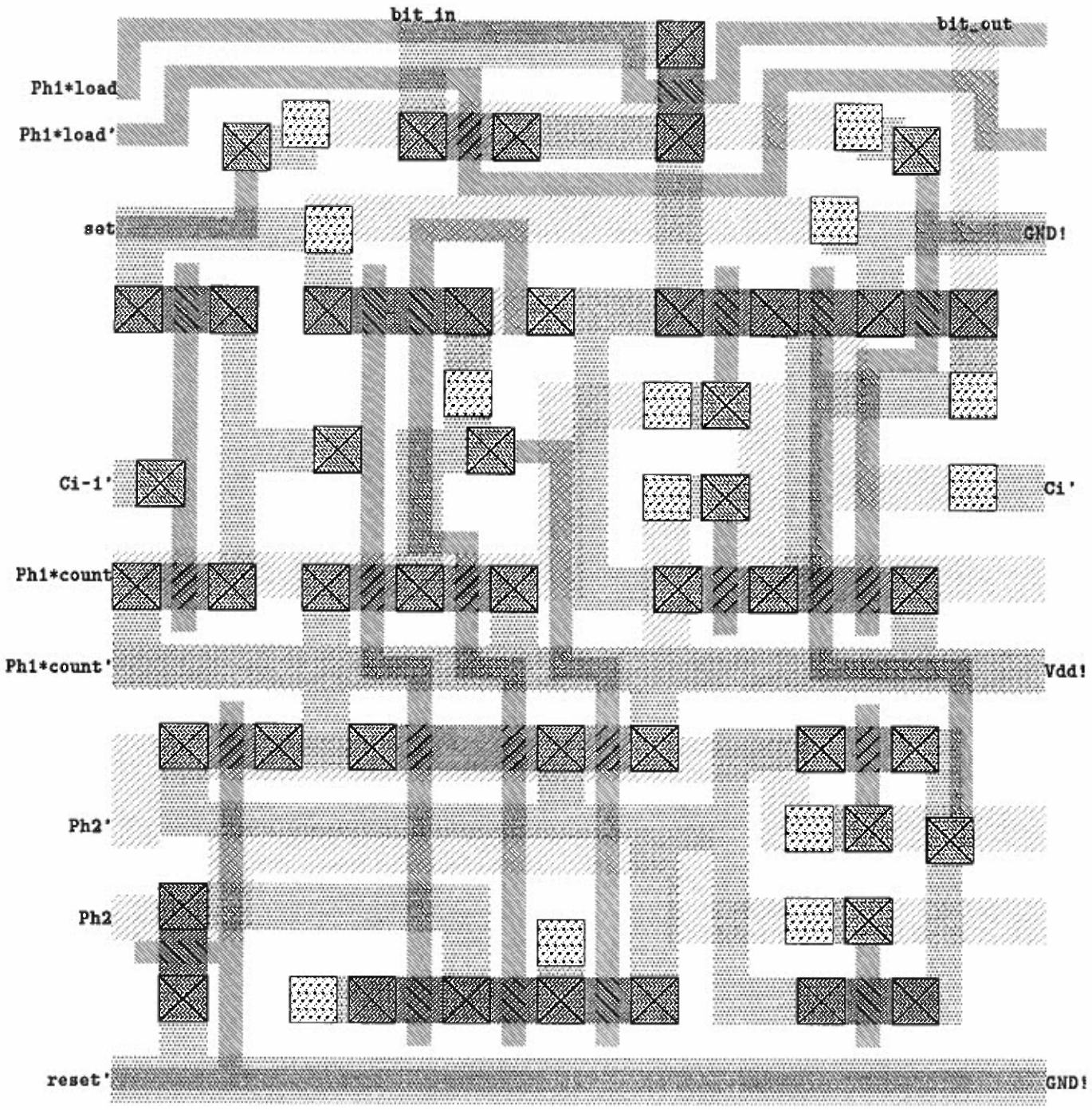
PARITY is a dynamic 5 bit odd parity checker. On $\phi 1$ the array is precharged. On $\overline{\phi 1}$ the output is discharged if an even number of ones are present on bit0 through bit4.

TESSELATION

Abuts horizontally.

LOGIC DIAGRAM




bit_in
bit_out
Phi*load
Phi*load'
set
GND!
Ci-i'
Ci'
Phi*count
Phi*count'
Vdd!
Ph2'
Ph2
reset'
GND!

NAME

COUNT_LOAD – loadable counter cell

SYNOPSIS

One bit of a loadable binary up-counter with set and reset

PROPERTIES

Size

$79 \times 94\lambda$

Interface

	label	name	layer	cap
inputs	Ci-1'	carry input	metal1	
	bit_in	load data in	metal1	
	Ph1*count	$\phi 1$ clock for count	metal2	
	Ph1*count'	$\overline{\phi 1}$ clock for count	metal2	
	Ph1*load	$\phi 1$ clock for load	polySi	
	Ph1*load'	$\overline{\phi 1}$ clock for load	polySi	
	Ph2	$\phi 2$ clock	metal2	
	Ph2'	$\overline{\phi 2}$ clock	metal2	
outputs	Ci'	carry output	metal1	
	bit_out	data out	metal2	

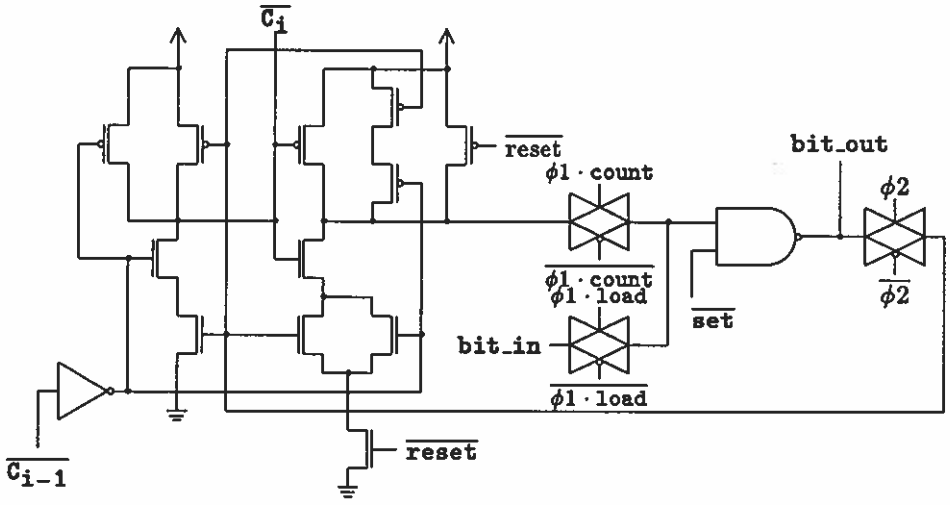
DESCRIPTION

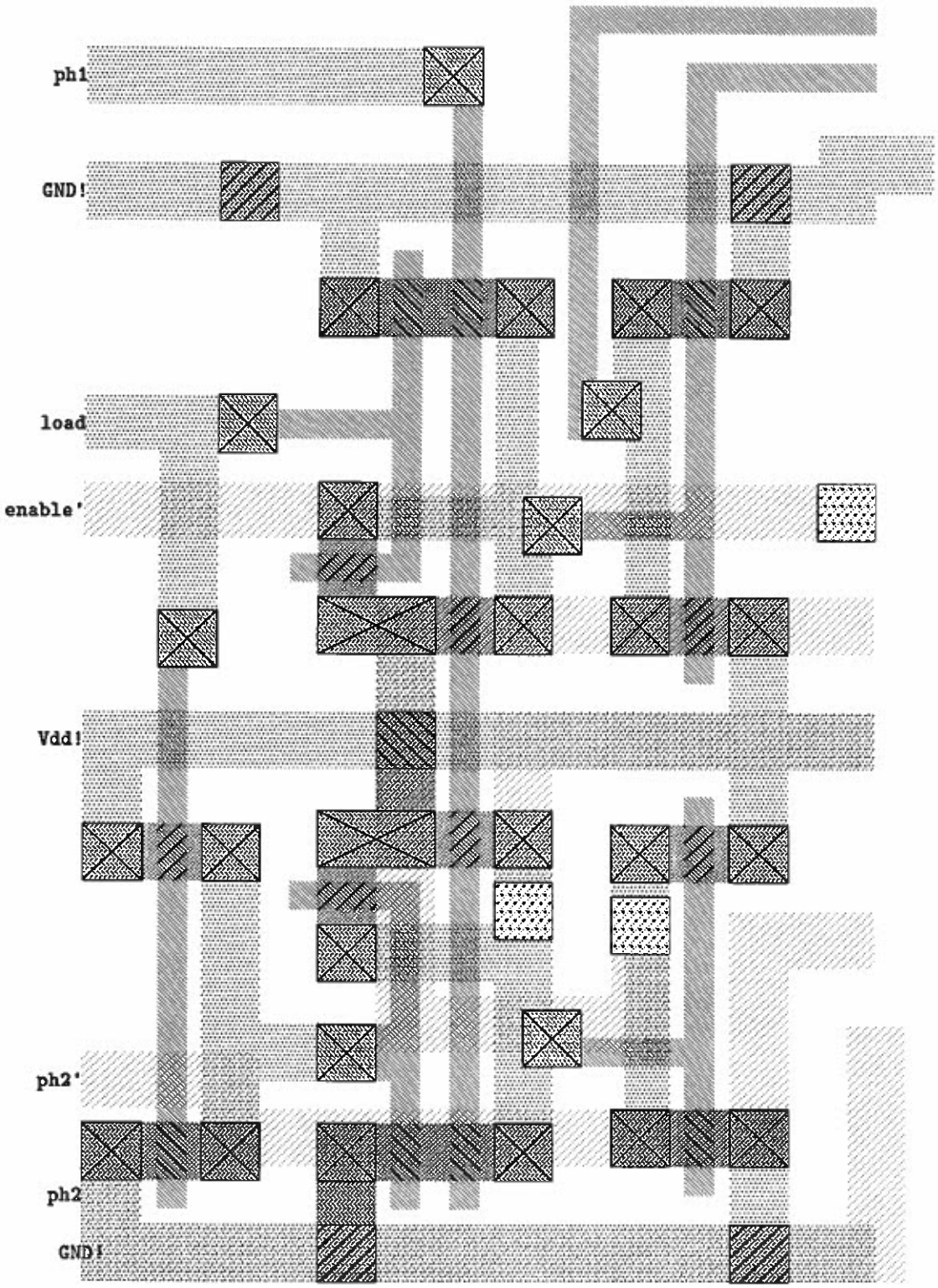
COUNT_LOAD is a single bit of a dynamic binary up-counter with set, reset and synchronous load. In normal operation (counting) $\phi 1 \cdot count$ and $\phi 2$ are used to clock the counter. Ci' of the low order bit is held low to enable counting.

TESSELATION

Abuts horizontally.

LOGIC DIAGRAM




ph1
GND!
load
enable'
Vdd!
ph2'
ph2
GND!

NAME

COUNT_LOADDRV – driver for loadable counter

SYNOPSIS

driver/control decoder for loadable counter

PROPERTIES

Size

$79 \times 94\lambda$

Interface

	label	name	layer	cap
inputs	enable	count enable	metal2	
	load	load/$\overline{\text{count}}$	metal1	
	Ph1	$\phi 1$ clock	metal1	
	Ph2	$\phi 2$ clock	metal2	
	Ph2′	$\overline{\phi 2}$ clock	metal2	

DESCRIPTION

COUNT_LOADDRV generates the various control signals needed by the loadable counter (COUNT_LOAD).

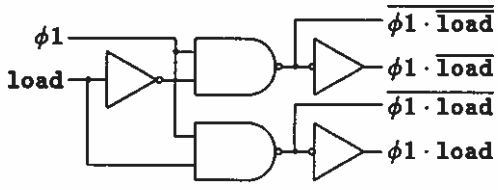
TESSELATION

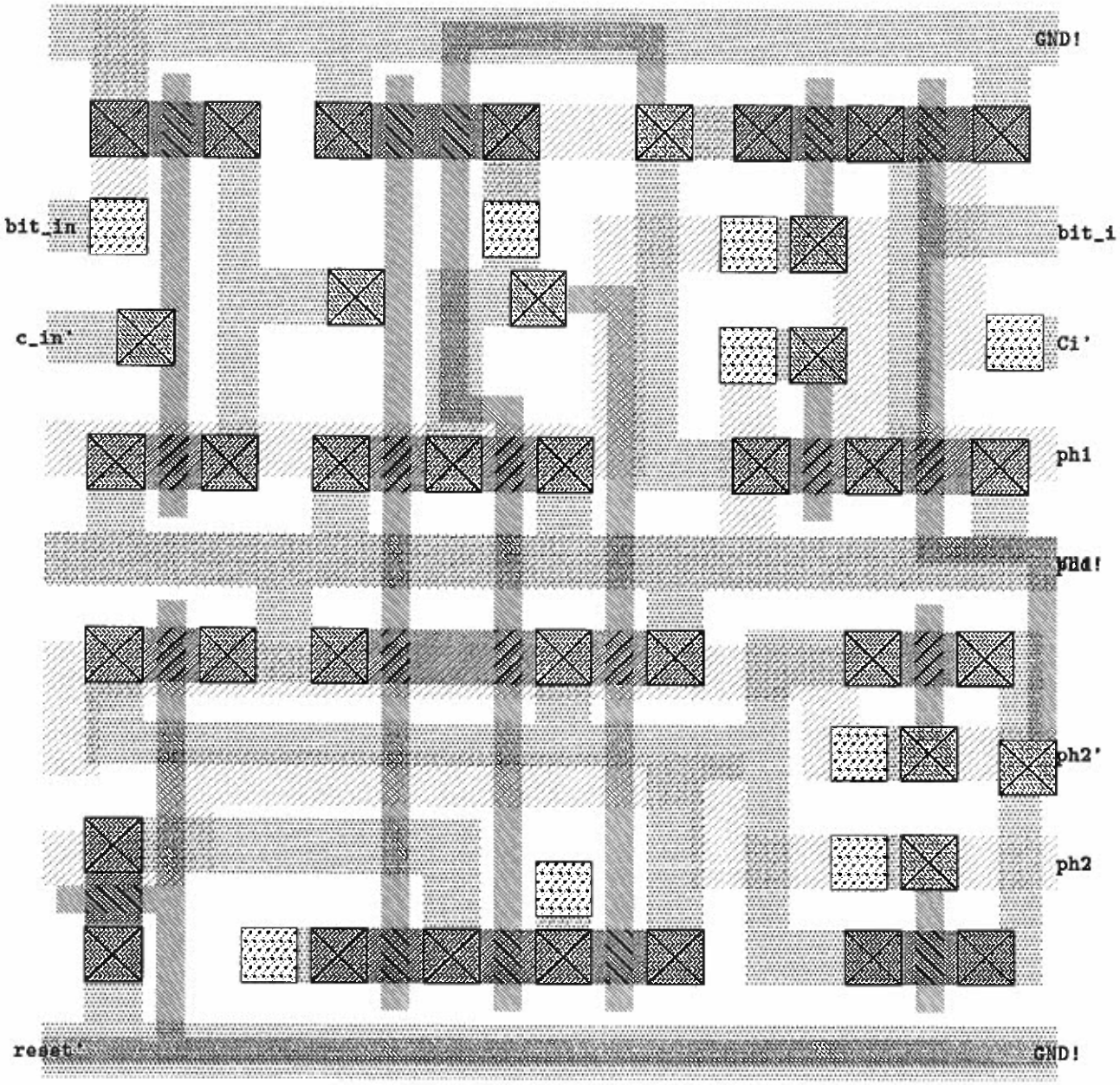
Abuts the left edge of COUNT_LOAD overlapping by 4λ.

SEE ALSO

COUNT_LOAD

LOGIC DIAGRAM




GND!
bit_in
bit_i
c_in'
Ci'
ph1
VDd!
ph2'
ph2
reset'
GND!

NAME

COUNT_CELL – counter cell

SYNOPSIS

One bit of a binary up-counter with reset

PROPERTIES

Size

$72 \times 78\lambda$

Interface

	label	name	layer	cap
inputs	Cin′	carry input	metal1	
	Ph1	$\phi 1$ clock	metal2	
	Ph1′	$\overline{\phi 1}$ clock	metal2	
	Ph2	$\phi 2$ clock	metal2	
	Ph2′	$\overline{\phi 2}$ clock	metal2	
outputs	Ci′	carry out	metal1	
	bit_i	data out	metal1	

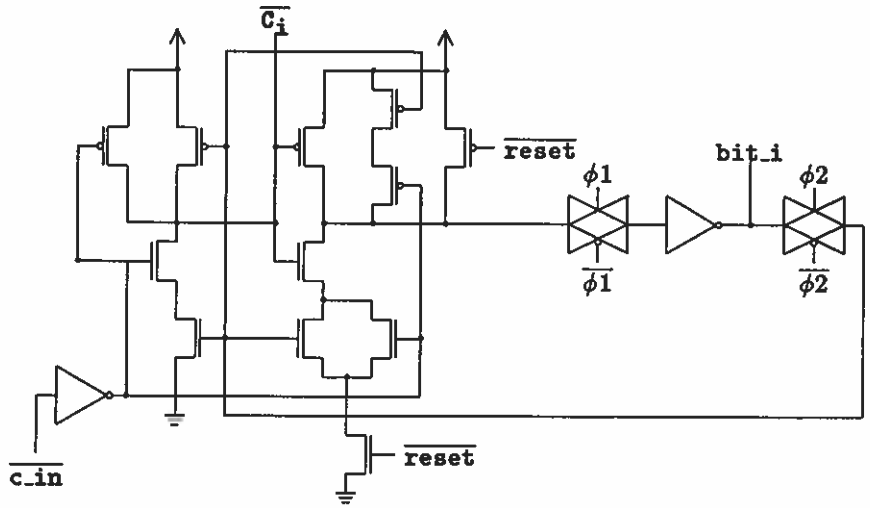
DESCRIPTION

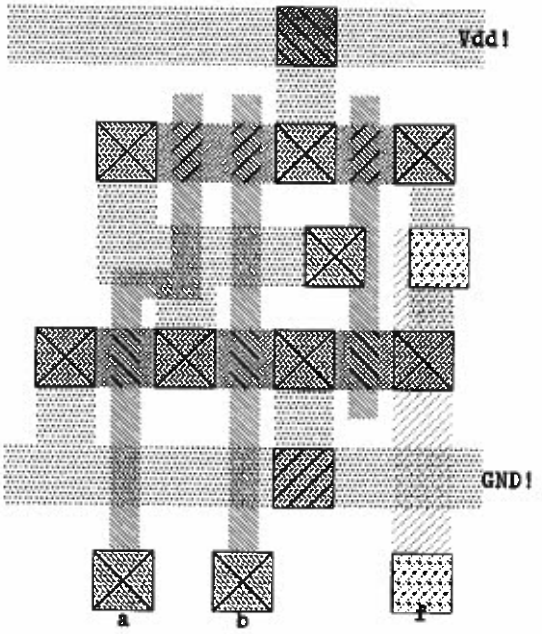
COUNT_CELL is a single bit of a dynamic binary up-counter with reset. In normal operation $\phi 1$ and $\phi 2$ are used to clock the counter. Ci′ of the low order bit is held low to enable counting.

TESSELATION

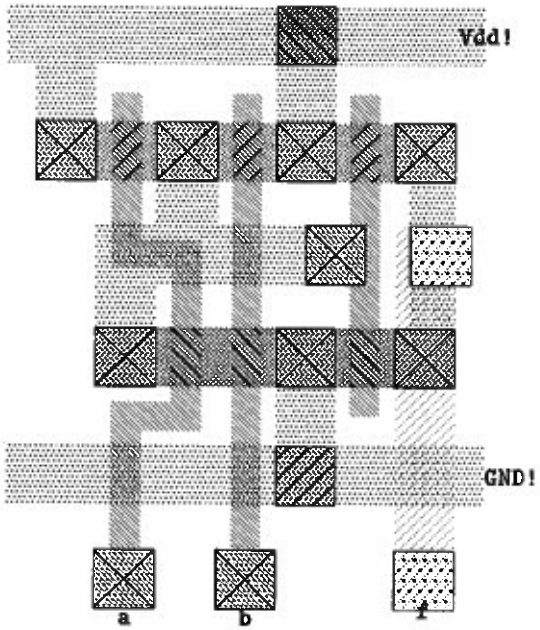
Abuts horizontally.

LOGIC DIAGRAM

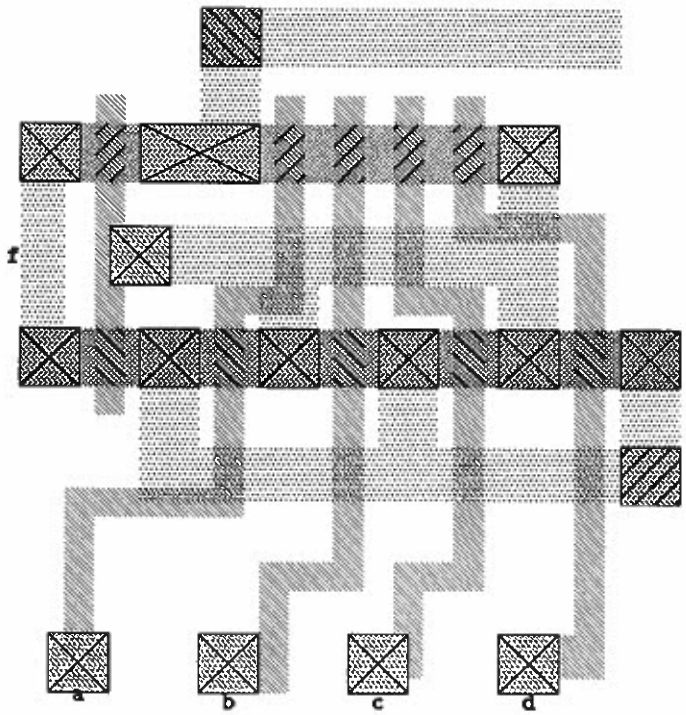




or



and



or4

NAME

OR – 2 input static OR gate
OR4 – 4 input static OR gate
AND – 2 input static AND gate

SYNOPSIS

Basic gates

PROPERTIES

Size

$32 \times 68\lambda$

Interface

	label	name	layer	cap
inputs	a		polycontact	
	b		polycontact	
	c		polycontact	
	d		polycontact	
outputs	f		metal1	

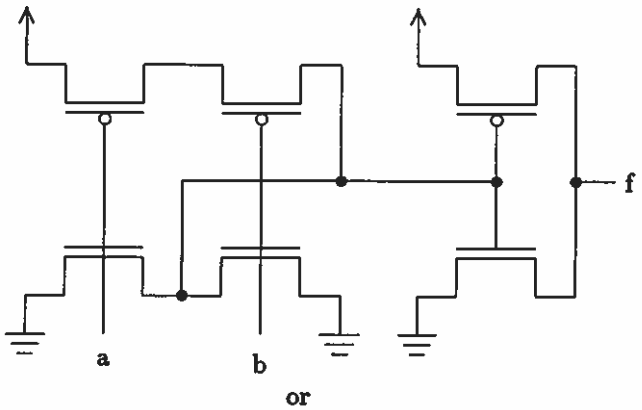
DESCRIPTION

These cells provide three basic gates to be used in "gluing" the the other pieces of the chip together.

TESSELATION

Abuts horizontally.

LOGIC DIAGRAM



or

or4 and and similar

References

[Tu85] Turner, Jonathan S. "Design of a Broadcast Packet Switching Network," Washington University Computer Science Department, WUCS-85-4, 3/85.

[WeEs85] Weste, Neil H. E. and Eshraghian, Kamran *Principles of CMOS VLSI Design*, Addison-Wesley, 1985.

[Bu85] Bubenik, Richard. "Performance Evaluation of a Broadcast Packet Switch." M.S. thesis, 8/85, Washington University, Computer Science Department

[IE] Mann, F. A. "Explanation of New Logic Symbols," in *The TTL Data Book*, Vol 1, Texas Instruments, 1984.