Report Number: WUCS-87-5

1987-04-01

Specification of Integrated Circuits for Broadcast Packet Network

Jonathan S. Turner

The broadcast packet network is a form of communications network based on high speed packet switches with a flexible multi-point connection capability, making them suitable for a wide variety of applications. This paper gives preliminary specifications for the integrated circuits being designed for the prototype switching system to be constructed at Washington University.

SPECIFICATION OF INTEGRATED CIRCUITS FOR BROADCAST PACKET NETWORK

Jonathan S. Turner

WUCS-87-5

April 1987

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

Abstract

The broadcast packet network is a form of communications network based on high speed packet switches with a flexible multi-point connection capability, making them suitable for a wide variety of applications. This paper gives preliminary specifications for the integrated circuits being designed for the prototype switching system to be constructed at Washington University.

This work supported by the National Science Foundation (grant DCI 8600947), Bell Communications Research, Italtel SIT and NEC.

SPECIFICATION OF INTEGRATED CIRCUITS FOR BROADCAST PACKET NETWORK

Jonathan S. Turner

These notes give a specification of the integrated circuits used within the Broadcast Packet Switch described in reference 1. The reader is assumed to be familiar with the contents of that report. The first section of this report provides general information covering such things as packet formats, etc. Later sections describe each chip in detail.

1. General Information

The format of packets within the Switch Module is shown in Figure 1. 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 switch elements. A value of 0000 signifies an empty packet slot. A value of 0001 signifies a normal point-to-point data packet. A value of 0011 signifies a normal broadcast packet. A value of 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 *Read* LCXT *Entry.*

2 *Write* LCXT *Entry.*

3 *Read* LCXT *Block.*

4 *Write* LCXT *Block.*

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 Ifield specifies which of four blocks to read. The data is written into nibbles 1–64 of the Ifield.

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

8 *Update* BTT. Directs BTC to update one entry in the BTT. See detailed description 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. Packet Switch Element

The switch element is used to construct the Copy Network, Distribution Network and Routing Network that make up the main part of the system's switch fabric. The way a switch element processes the packets it receives is determined by information in the packet headers and by its operating mode, which is determined by control leads.

When operating in the routing mode, the switch element normally delivers each packet it receives to one of its two output ports based on an address in the packet's header. When operating in the distribution mode, the switch element distributes packets evenly between its outputs. When operating in the copy mode, the switch element replicates certain packets, sending them to both outputs. In addition, all switch elements route test packets (identified by a header field) based on their address field.

2.1. External Interface

The external leads of the switch element are shown in Figure 2 and described briefly below.

- *Upstream data leads* (ud_0, ud_1). Incoming data from upstream neighbors. Four bits wide (one nibble).
- *Upstream parity* (up_0, up_1). Odd parity on incoming data from upstream neighbor.
- *Upstream grants* (ug_0, ug_1). Grant signals to upstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent packet cycle.

- *Downstream data leads* ($\mathtt{dd}_0$, $\mathtt{dd}_1$). Outgoing data to downstream neighbors. Four bits wide (one nibble).
- *Downstream parity* ($\mathtt{dp}_0$, $\mathtt{dp}_1$). Odd parity on outgoing data to downstream neighbor.
- *Downstream grants* ($\mathtt{dg}_0$, $\mathtt{dg}_1$). Grant signals from downstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent packet cycle.
- *Stage number* (sn). Two bit stage number. Each network has four stages (columns). This identifies which stage the switch element is part of.
- *Operating mode* (om). Two bit code identifying which of three possible operating modes the switch element implements. 1 for route, 2 for distribute, 3 for copy.
- *Rotate Routing Field* (rrf). Causes switch element to rotate the second through fourth nibbles of test packets.
- *Reset* (R). Initialize internal state machines.
- *Error* (E). Report parity violation or other error.
- *Clock* (ϕ_1, ϕ_2). Two-phase, non-overlapping clock.
- *Packet Time* (pt). Goes high when first nibble of packet is present on ud leads.
- *Grant Time* (gt). Goes high when valid data is present on downstream grants leads.
- *Power* (P).
- *Ground* (G).

2.2. Detailed Description

The switch element 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. Each switch element is able to buffer two complete packets (80 nibbles each). When possible, packets are propagated through to the downstream data leads during the same packet cycle that they arrive. If an incoming packet cannot be immediately propagated, it is stored in a buffer until a later packet cycle. If a switch element transmits a packet (or packets) during a particular packet cycle, it must transmit the first nibble on the downstream data leads exactly four clock cycles after the st lead goes high. Successive nibbles of the packet must follow on successive clock cycles.

Sometime before a packet cycle begins, the gt lead goes high. This indicates that the signals on the downstream grant leads are valid. When $\mathtt{dg}_0$ is asserted, it means that the switch element has permission to transmit a packet on $\mathtt{dd}_0$ during the next packet cycle. Similarly, when $\mathtt{dg}_1$ is asserted, it means that the switch element has permission to transmit a packet on $\mathtt{dd}_1$ during the next packet cycle. Two clock cycles after gt goes high, the upstream grant leads must be valid. During that time, the switch element must determine if it can accomodate new packets during the next cycle and indicate its decision using the upstream grants.

The chip is clocked by a two phase non-overlapping clock. The desired clock rate is 10 MHz giving a clock period of 100 ηs. When ϕ_1 is high, the signals on the ud and up leads of the chip are valid. The st 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 st are high. Successive up-going transitions of st are at least 80 clock cycles apart. Similarly, gt goes high while ϕ_1 is low and stays high for one clock cycle. The data on the downstream grant leads are valid when both ϕ_1 and gt are high.

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.

A routing switch element always routes the packets it receives based on information in the ADR field of the packet. In particular, it selects one bit of the ADR field, based on the stage number given on the sn leads. If sn = 00, the high order bit of the ADR field is selected, if sn = 01, the second bit is selected, and so forth. The selected address bit specifies one of the switch element's two output ports. The switch element is required to route the packet to the selected port.

When a distribution switch element receives a packet with RC = 00x1, it may route it to either of its two output ports. In general, such packets are routed so as to distribute packets from each input, evenly across the output ports. How this is done will be described more fully below.

When a copy switch element receives a packet with RC = 0011, it examines the fanout field. If FAN > 2^{sn} the packet must be replicated and transmitted on both of the switch element's outputs; otherwise the packet may be routed to either of the two output ports. Packets with RC = 0001 may routed to either output port. Packets with RC = 0111 are routed to the output port specified by the ADR field as described earlier.

The overall structure of a switch element is shown in Figure 3. Its main components are the two *Input Circuits* ($\mathrm{IC}_0, \mathrm{IC}_1$) and the *Node Control Circuit* (NCC). The ICs each contain a buffer large enough to hold one complete packet plus some data path and control circuitry. The NCC allocates output ports on the basis of requests made by the ICs and the downstream grant signals. It also computes the upstream grant signals.

If each IC has a packet, the NCC attempts to accomodate both. If this is not possible, it gives priority to a broadcast packet over a non-broadcast packet and to a packet being routed over a packet that can use either output. If the conflict is between two packets with the same priority, the input port which most recently transmitted a packet while the other was idle, is forced to wait. The NCC also attempts to distribute packets evenly across the output ports. When both ports are available but only one is needed, the NCC determines which output port most recently transmitted a packet while the other was idle and sends the packet to the opposite port. In order to make its decisions, the NCC needs several pieces of information.

- *Port Requests* ($\mathrm{r}_0, \mathrm{r}_1$). These take the form of three bit codes. 100 is a request for either port, 101 is a request for port 0, 110 is a request for port 1 and 111 is a request for both ports. 0xx means that no ports are needed.
- *Port Availability* ($\mathrm{a}_0, \mathrm{a}_1$). A 1 means that the output port is available for use.
- *Last favored input* (in). Gives the number of the input port that most recently sent a packet while the other was idle.
- *Last favored output* (out). Gives the number of the output port that was most recently busy while the other was idle.

Using this information, the NCC determines the *enable signals* ($\mathrm{en}_0, \mathrm{en}_1$). These are two bit codes that tell the ICs which output ports to use; 00 specifies no ports, 01 specifies port 0, 10 specifies port 1, 11 specifies both ports.

The NCC performs its computation in two parts. During the propagation of grant signals, it accepts requests from the two ICs and allocates output ports for the next cycle based on these requests and the downstream grant signals. It also computes the upstream grant signals based on this information. When the next packet cycle starts, the ICs may request ports for their newly arriving packets. The NCC attempts to satisfy these requests using whatever ports are still available. Note that if a port is allocated to an IC during grant propagation the IC is guaranteed that it can use that port during the next cycle. Also note that an IC cannot transmit two packets during the same packet cycle. Consequently, if an IC requests a port during grant propagation it will not make a request at the start of the packet cycle.

The computation performed during grant propagation is described below.

```
$a_0 := dg_0$; $a_1 := dg_1$;
$en_0[0] := [r_0 = 111 \wedge a_1 \wedge a_0 \wedge (r_1 \neq 111 \vee in = 1)] \vee$
            $[r_0 = 101 \wedge a_0 \wedge r_1 \neq 111 \wedge (r_1 \neq 101 \vee in = 1)] \vee$
            $[r_0 = 100 \wedge a_0 \wedge r_1 \neq 1x1 \wedge (r_1 \neq 100 \vee in = 1)]$
$en_0[1] := [r_0 = 111 \wedge a_1 \wedge a_0 \wedge (r_1 \neq 111 \vee in = 1)] \vee$
            $[r_0 = 101 \wedge a_1 \wedge r_1 \neq 111 \wedge (r_1 \neq 110 \vee in = 1)] \vee$
            $[r_0 = 100 \wedge a_1 \wedge r_1 \neq 11x \wedge (r_1 \neq 100 \vee in = 0)]$
$en_1[0] := [r_1 = 111 \wedge a_1 \wedge a_0 \wedge (r_0 \neq 111 \vee in = 0)] \vee$
            $[r_1 = 101 \wedge a_0 \wedge r_0 \neq 111 \wedge (r_0 \neq 101 \vee in = 0)] \vee$
            $[r_1 = 100 \wedge a_0 \wedge r_0 \neq 1x1 \wedge (r_0 \neq 100 \vee in = 0)]$
$en_1[1] := [r_1 = 111 \wedge a_1 \wedge a_0 \wedge (r_0 \neq 111 \vee in = 0)] \vee$
            $[r_1 = 101 \wedge a_1 \wedge r_0 \neq 111 \wedge (r_0 \neq 110 \vee in = 0)] \vee$
            $[r_1 = 100 \wedge a_1 \wedge r_0 \neq 11x \wedge (r_0 \neq 100 \vee in = 1)]$
$a_0 := a_0 \wedge \neg en_0[0] \wedge \neg en_1[0]$;
$a_1 := a_1 \wedge \neg en_0[1] \wedge \neg en_1[1]$;
$ug_0 := r_0 = 000 \wedge en_0[0] \wedge en_0[1]$;
$ug_1 := r_1 = 000 \wedge en_1[0] \wedge en_1[1]$;
```

At the start of the packet cycle, the following computation is performed.

```
$en_0[0] := en_0[0] \vee [r_0 = 111 \wedge a_1 \wedge a_0 \wedge (r_1 \neq 111 \vee in = 1)] \vee$
                       $[r_0 = 101 \wedge a_0 \wedge r_1 \neq 111 \wedge (r_1 \neq 101 \vee in = 1)] \vee$
                       $[r_0 = 100 \wedge a_0 \wedge r_1 \neq 1x1 \wedge (r_1 \neq 100 \vee in = 1)]$
$en_0[1] := en_0[1] \vee [r_0 = 111 \wedge a_1 \wedge a_0 \wedge (r_1 \neq 111 \vee in = 1)] \vee$
                       $[r_0 = 101 \wedge a_1 \wedge r_1 \neq 111 \wedge (r_1 \neq 110 \vee in = 1)] \vee$
                       $[r_0 = 100 \wedge a_1 \wedge r_1 \neq 11x \wedge (r_1 \neq 100 \vee in = 0)]$
$en_1[0] := en_1[0] \vee [r_1 = 111 \wedge a_1 \wedge a_0 \wedge (r_0 \neq 111 \vee in = 0)] \vee$
                       $[r_1 = 101 \wedge a_0 \wedge r_0 \neq 111 \wedge (r_0 \neq 101 \vee in = 0)] \vee$
                       $[r_1 = 100 \wedge a_0 \wedge r_0 \neq 1x1 \wedge (r_0 \neq 100 \vee in = 0)]$
$en_1[1] := en_1[1] \vee [r_1 = 111 \wedge a_1 \wedge a_0 \wedge (r_0 \neq 111 \vee in = 0)] \vee$
                       $[r_1 = 101 \wedge a_1 \wedge r_0 \neq 111 \wedge (r_0 \neq 110 \vee in = 0)] \vee$
                       $[r_1 = 100 \wedge a_1 \wedge r_0 \neq 11x \wedge (r_0 \neq 100 \vee in = 1)]$
$b_0 := en_0[0] \vee en_0[1]$; $b_1 := en_1[0] \vee en_1[1]$;
$in := (\neg b_0 \wedge b_1) \vee (in \wedge \neg b_0 \wedge b_1) \vee (b_0 \wedge b_1 \wedge en_1[0])$;
$b_0 := en_0[0] \vee en_1[0]$; $b_1 := en_0[1] \vee en_1[1]$;
$out := (\neg b_0 \wedge b_1) \vee (out \wedge b_0 = b_1)$;
```

The header modification circuit changes headers in two ways. When a broadcast packet is replicated, it produces two copies, in which the FAN fields have been halved. If the original value of FAN is odd, the low order bit of the BCN field determines which of the copies gets the "larger half." In particular, if BCN is even, the copy that goes to output port 0 has FAN changed to $\lfloor(\text{FAN}+1)/2\rfloor$, while the copy that goes to output port 1 has FAN changed to $\lfloor \text{FAN}/2 \rfloor$. If BCN is odd, the copy that goes to output port 1 has FAN changed to $\lfloor (\text{FAN}+1)/2 \rfloor$, while the copy that goes to output port 0 has FAN changed to $\lfloor \text{FAN}/2 \rfloor$. In the case of a test packet, a switch element in which the rrf lead is asserted rotates the second through fourth nibbles of the packet. That is, it moves the third and fourth nibbles to the second and third positions respectively while moving the second nibble to the fourth position.

3. Broadcast Translation Chip

This section give a specification of the Broadcast Translation Chip (BTC). The BTC receives packets from the Copy Network and usually forwards them on to the Distribution Network. In the process it may modify certain fields of the packet or use information within the packet to modify its internal tables.

3.1. External Interface

The external leads of the BTC are shown in Figure 4 and described briefly below.

- *Upstream data leads* (ud) 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) Outgoing data to downstream neighbors. Four bits wide (one nibble).
- *Downstream parity* (dp) Odd parity on outgoing data to downstream neighbor.
- *Reset* (R). Initialize internal state machines.
- *Error* (E). Report parity violation or other error.
- *Clock* (ϕ_1, ϕ_2). Two-phase, non-overlapping clock.
- *Packet Time* (pt). Goes high when first nibble of packet is present on ud leads.
- *Power and Ground* (P,G).

3.2. Detailed Description

The BTC operates on the basis of a *packet cycle* which starts when the st 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 100 ηs. When ϕ_1 is high, the signals on the ud and up leads of the chip are valid. The st 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 st are high. Successive up-going transitions of st 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.

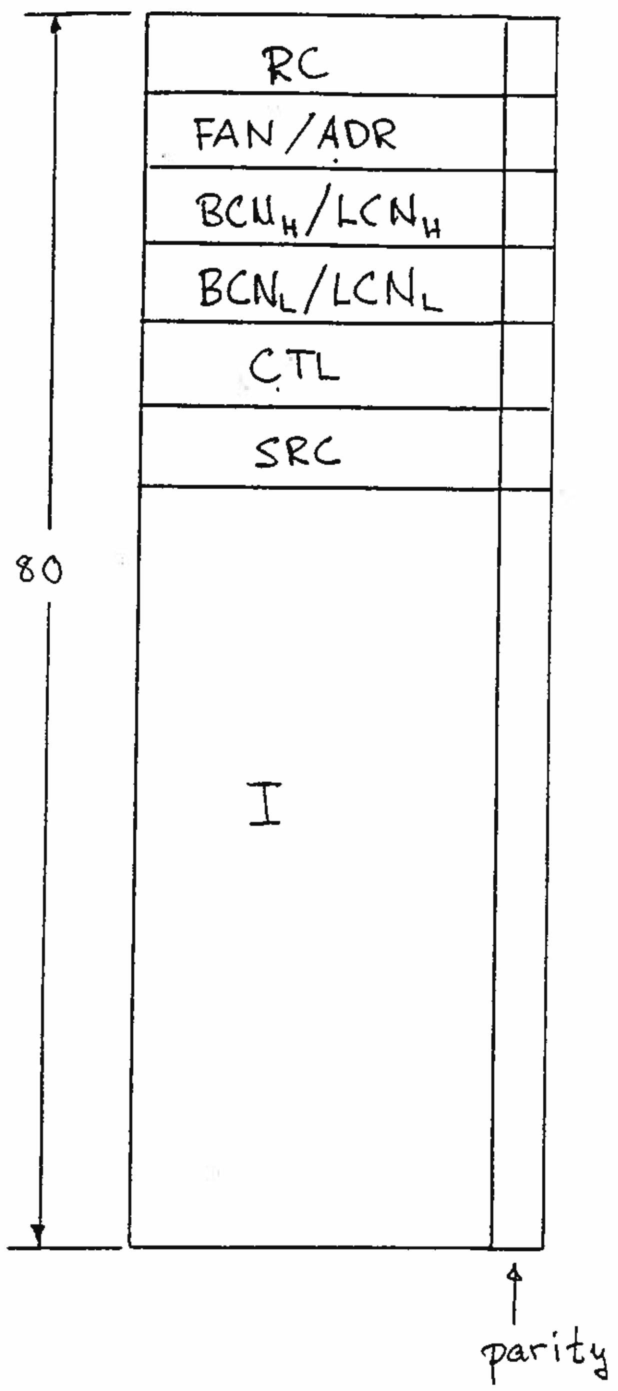
The BTC contains two internal tables. The *Broadcast Translation Table* (BTT) is used to modify the headers of broadcast packets as they pass through the BTT. It has 64 entries, each of which is four nibbles wide. The i^{th} entry is denoted BTT[i]. Parity is also stored for each nibble and checked when 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 below. Note that in some cases, 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.

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

References

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



RC : Routing Control
FAN : Fanout
ADR : Address
BCN : Broadcast Channel Number
LCN : Logical Channel Number
CTL : Control Field
SRC : Source
I : Information field

Figure 1. Packet Format

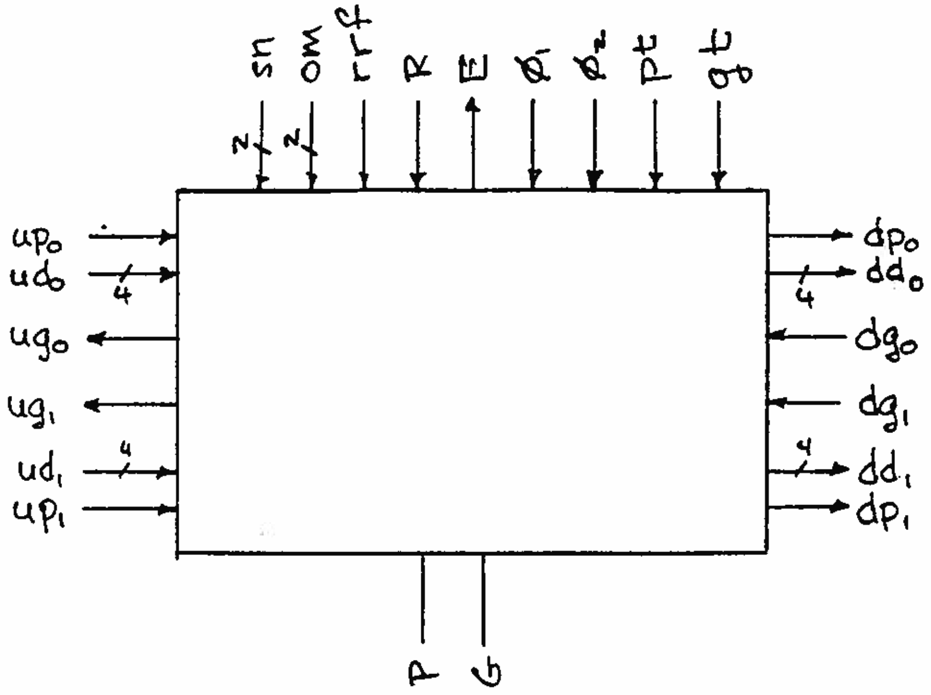


Figure 2. Node Connections

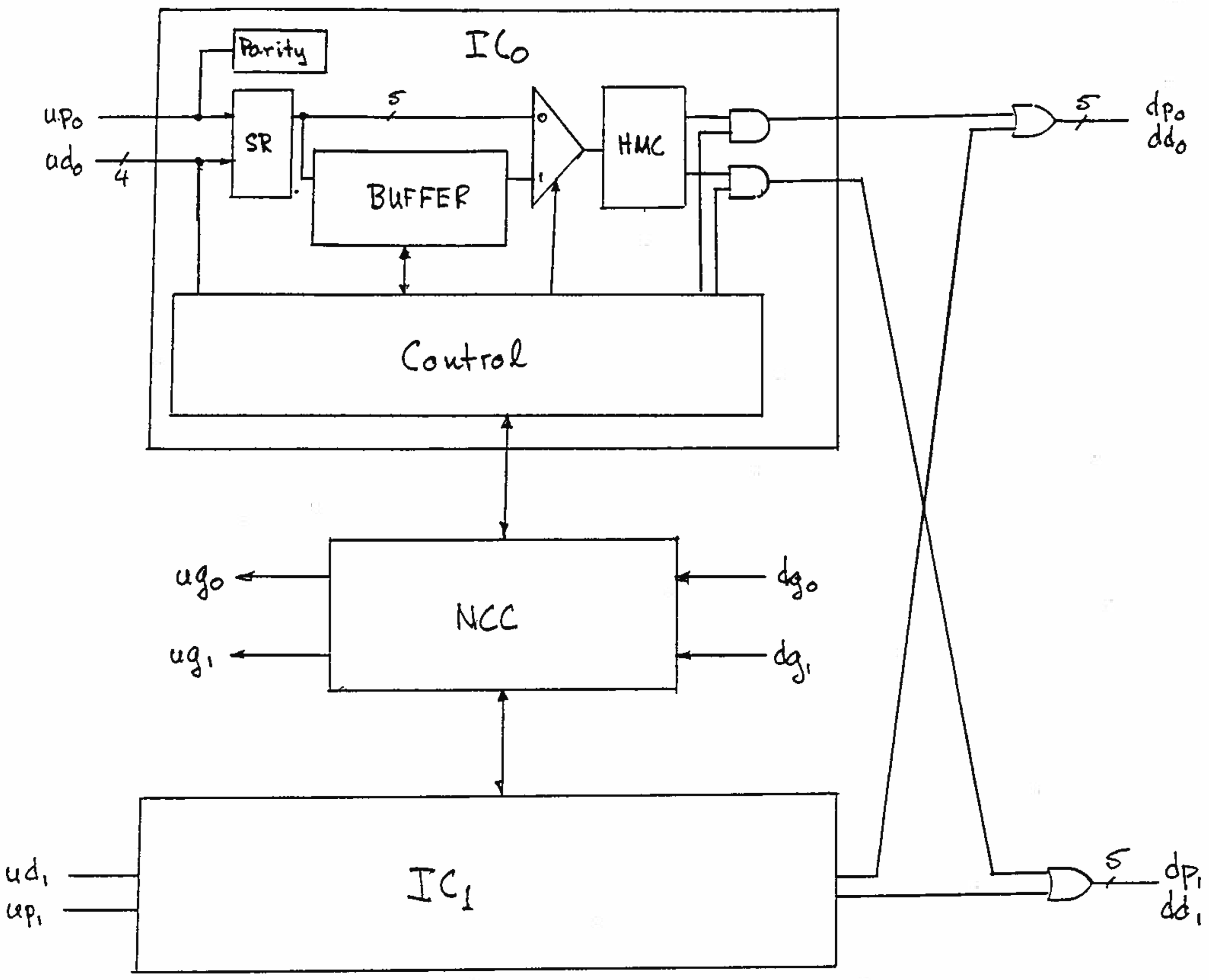


Figure 3. Node Structure

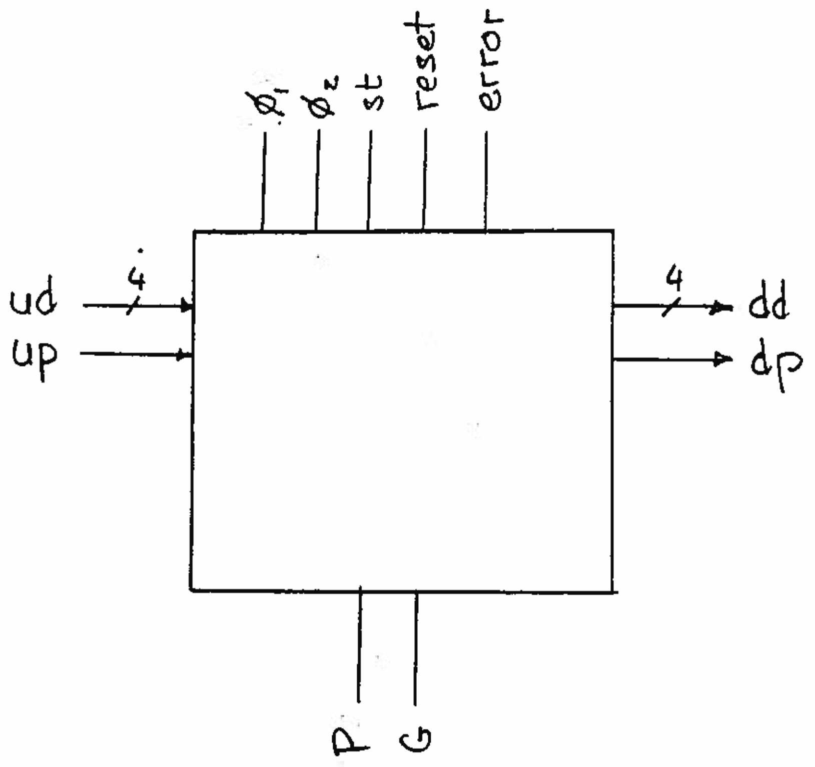


Figure 4. BTC Connections