```
PPPPPP     PPPPPP     U      U
P     P    P     P    U      U
P      P   P      P   U      U
P     P    P     P    U      U
PPPPPP     PPPPPP     U      U     *********
P          P          U      U
P          P          U      U
P          P          U      U
P          P           UUUUUU
```

```
 CCCC   OOOOO   N     N TTTTTTT  RRRR    OOOOO  L      L      EEEEEE  RRRRR
C    C O     O  NN    N    T    R    R  O     O L      L      E       R    R
C    C O     O  NN    N    T    R     R O     O L      L      E       R     R
C      O     O  N N   N    T    R    R  O     O L      L      E       R    R
C      O     O  N  N  N    T    RRRRR   O     O L      L      EEEEE   RRRRR
C      O     O  N   N N    T    R  R    O     O L      L      E       R  R
C    C O     O  N    NN    T    R   R   O     O L      L      E       R   R
C    C O     O  N    NN    T    R    R  O     O L      L      E       R    R
 CCCC   OOOOO   N     N    T    R     R  OOOOO  LLLLLL LLLLLL EEEEEE  R     R
```

```
IIIII  N     N TTTTTTT EEEEEE  RRRR   FFFFFF    A      CCCC  EEEEEE
  I    NN    N    T    E      R    R  F        A A     C    C E
  I    NN    N    T    E      R     R F       A   A    C    C E
  I    N N   N    T    E      R    R  F      A     A   C      E
  I    N  N  N    T    EEEEE  RRRRR   FFFFF  AAAAAAA   C      EEEEE
  I    N   N N    T    E      R  R    F     A       A C      E
  I    N    NN    T    E      R   R   F     A       A C    C E
  I    N    NN    T    E      R    R  F     A       A C    C E
IIIII  N     N    T    EEEEEE R    R  F     A       A  CCCC  EEEEEE
```

```
 SSSSS   PPPPP    EEEEEE   CCCC    SSSSS
S     S  P    P   E       C    C  S     S
 S       P     P  E       C    C   S
  S      P    P   E       C         S
   S     PPPPP    EEEEE   C          S
    S    P        E       C           S
     S   P        E       C    C       S
S     S  P        E       C    C  S     S  **
 SSSSS   P        EEEEEE   CCCC    SSSSS   **
```

PPU - CONTROLLER INTERFACE SPECIFICATION

written by

David W. Milton   7/16/81

PPU - Controller Interface Specification

The PPU resides in the system 8000 mainframe and communicates with the system 8000 bus. With one exception (see SSU specifications), all I/O activities associated with the System 8000 involve an interface over a 50 pin flat cable with standard 3M Co. 3416 (or equivelent) type, 50 pin connectors on either end. One end connects to one of four ports on the PPU. The other end connects to any one of a variety of controllers. Each controller communicates with a specific type of I/O device such as disk drives, terminals, lineprinters, etc.

Within this cable, there are three sub-channels for communication. One is an 8-bit + 1 parity bit wide data path with four control signals to handle bi-directional asynchronous data transfers. This is called the data path. The second is an 8-bit wide path also with 4 control signals to handle bi-directional asynchronous transfers. This is called the command-status path since the PPU will send commands to a controller over this path and a controller will send status to the PPU over this path. The third is two signals named PFW- and RESET-. These signals reflect the integrity of the System 8000 power supply(s), and RESET- also allows the operating system to do a hard reset to a controller. All signals on this 50 pin flat cable are active low.

The protocol on the 8 control wires above (4 for the data path and 4 for the command-status path) is called the handshake (the data path handshake and the command-status handshake respectively). This handshake is used to sort out who is sending data when, and, at what time the receiving unit may know the data is valid and may be latched into his input registers. You will notice in the following table listing pin-outs and signal names on this 50-pin flat cable, that what the PPU asserts as command-out, the controller will see as command-in. Likewise, what the controller asserts as command-out, the PPU will see as command-in. Also, what each unit asserts as flag-out, the other unit sees as flag-in.

PPU - Controller Interface Pin Assignments

PPU Signal Name	Pin Number	Controller Signal Name
Data Path - Parity	1.	Parity - Data Path
- Bit #7	3.	#7 Bit -
- #6	5.	#6 -
- #5	7.	#5 -
- #4	9.	#4 -
- #3	11.	#3 -
- #2	13.	#2 -
- #1	15.	#1 -
- #0	17.	#0 -
Data Command-In	19.	Command-Out Data
Command-Out	21.	Command-In
Flag-In	23.	Flag-Out
Flag-Out	25.	Flag-In
Command-Status - Bit #7	27.	#7 Bit - Command-Status
- #6	29.	#6 -
- #5	31.	#5 -
- #4	33.	#4 -
- #3	35.	#3 -
- #2	37.	#2 -
- #1	39.	#1 -
- #0	41.	#0 -
Command-Status Command-In	43.	Command-Out Command-Status
Command-Out	45.	Command-In
Flag-In	47.	Flag-Out
Flag-Out	49.	Flag-In
RESET-	18.	RESET-
PFW-	42.	PFW-

1) With the exception of #18 and #42 above, all even numbered pins are ground.

2) Bit #7 is the MSB. Bit #0 is the LSB.

3) If the eight data lines and the parity bit all float high, this condition is detected as a parity error.

Command-Status Handshake

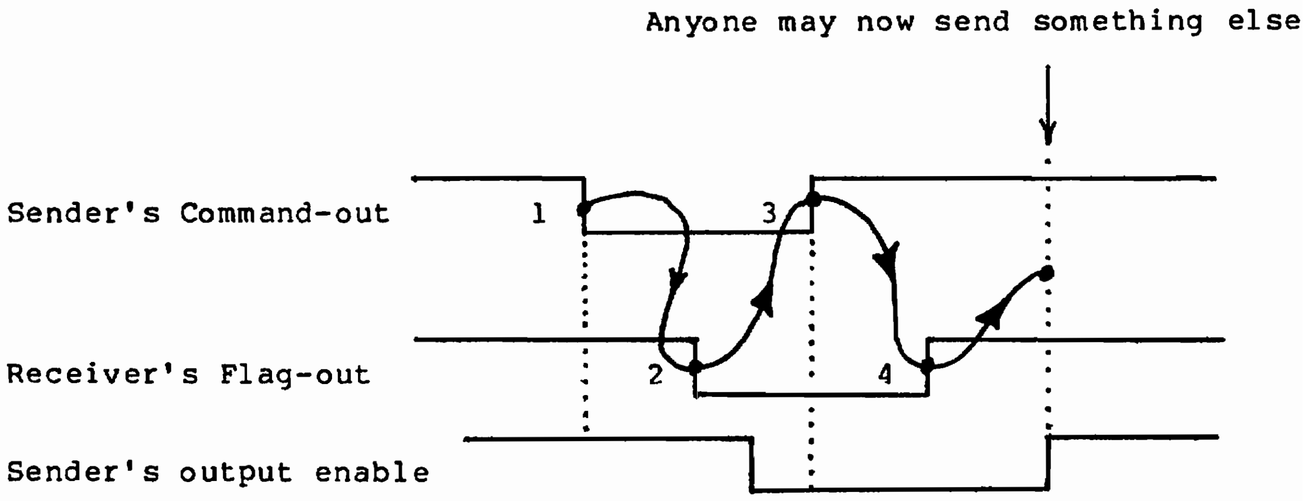
Logically, here is how these signals control a transfer from the PPU to a controller or visa-versa:

1) Unit A (the sender) wants to send something to Unit B (the receiver). Sender asserts command-out to tell the receiver that there is data to be taken.

2) In response, receiver asserts flag-out to tell sender that receiver sees that there is data to be taken.

3) In response, sender releases command-out to tell receiver that he sees that receiver is alive and awake. Note that sometime after asserting command-out and before releasing command-out, sender must enable his output drivers.

4) Receiver will continue to assert flag-out until he has both taken the data sent and has seen that command-out is released. When both of these conditions are satisfied, the receiver releases flag-out, which signifies the end of the transfer. When sender sees that flag-out is released, sender knows that the transfer is complete and may either turn off his output drivers or re-assert command-out with new data to be sent.



All above signals are active low.

Data Path Handshake

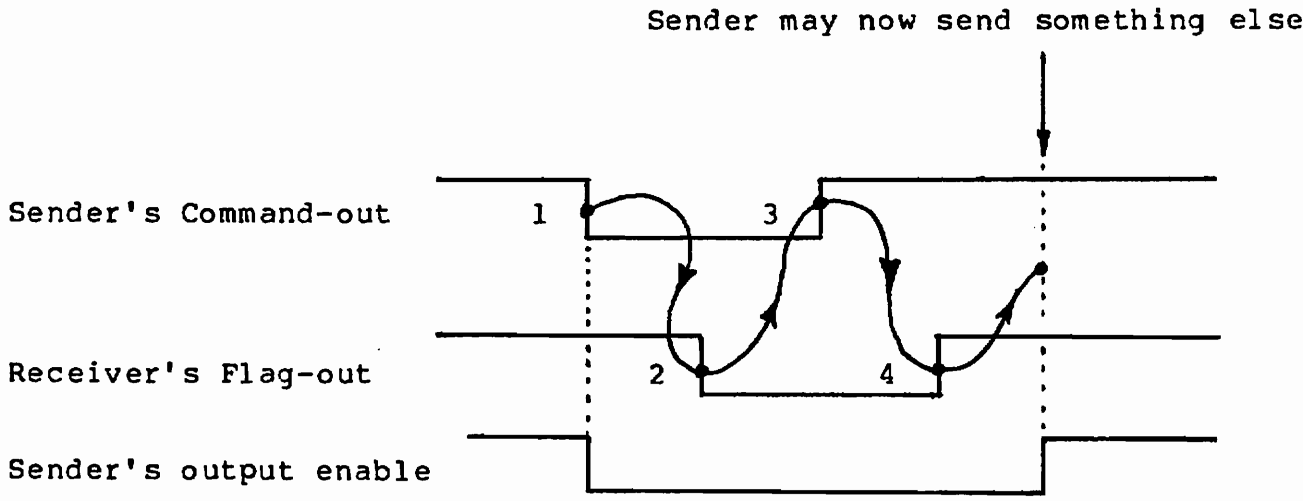
Logically, here is how these signals control a transfer from the PPU to a controller or visa-versa:

1) Unit A (the sender) wants to send something to Unit B (the receiver). Sender loads up his output registers, and then both enables his output drivers and asserts command-out to tell the receiver that there is data to be taken.

2) In response, receiver asserts flag-out to tell sender that receiver sees that there is data to be taken.

3) In response, sender releases command-out to tell receiver that he sees that receiver is alive and awake.

4) Receiver will continue to assert flag-out until he has both taken the data sent and has seen that command-out is released. When both of these conditions are satisfied, the receiver releases flag-out, which signifies the end of the transfer. When sender sees that flag-out is released, sender knows that the transfer is complete and may either turn off his output drivers or re-assert command-out with new data to be sent.



All above signals are active low.

Contentions for the Command-Status Path

The PPU and the Controllers shall function such that contention for use of the data path never occurs. Via communication over the command-status path, both ends will know who is receiving and who is sending well in advance of use of the data path, so that each end will have the appropriate logic enabled.

There may, however, be occasional contention for use of the command-status path. The PPU, the Controllers, and the Operating System shall be programmed to minimize the possibility of contention, but it may still happen. Following is a description of how transfers may happen and how contention will be resolved. Refer to the 2nd page following for a timing diagram of the 3 possible cases.

A. PPU sends data to a controller :
   1. PPU asserts command-out.
   2. Controller acknowledges by asserting flag-out.
   3. PPU releases command-out.
   4. Controller takes data and releases flag-out.

B. Controller sends data to a PPU
   1. Controller asserts command-out.
   2. PPU acknowledges by asserting flag-out.
   3. Controller releases command-out.
   4. PPU takes data and releases flag-out.

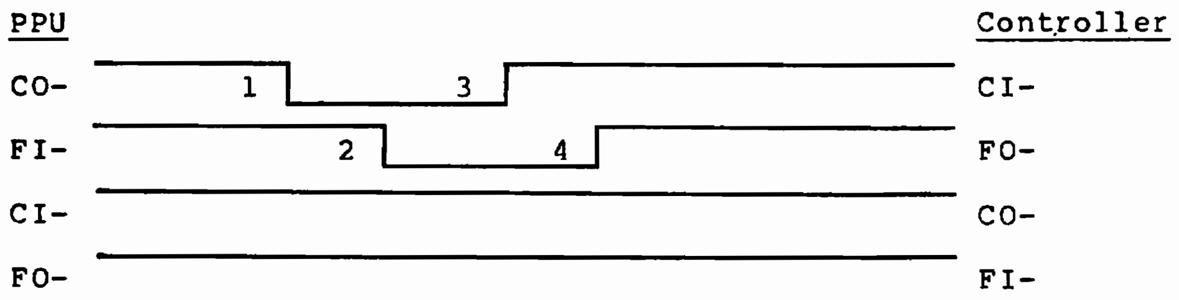
C. Controller tries to send data, PPU ignores it and sends data itself. Controller must take PPU's data :
   1. Controller asserts command-out.
   2. PPU ignores controllers command-in but instead asserts command-out.
   3. Controller releases command-out and acknowledges PPU's command-in by asserting flag-out.
   4. PPU releases command-out.
   5. Controller takes the data and releases flag-out.

The PPU is in command of this handshake. The PPU may respond to command-in and take data the controller is sending or it may choose to ignore it. If the PPU tries to send data by asserting command-out, the data will be sent, and the controller must take it. Anytime a controller sees command-in, it must release command-out (if set) and take the PPU's data by asserting flag-out. Thus, if a controller sets command-out, after an arbitrarily long or short length of time, the controller will see either flag-in which indicates the PPU will take the data, or the controller will see command-in which indicates the PPU will not take the controllers data but instead the controller must take the PPU's data.

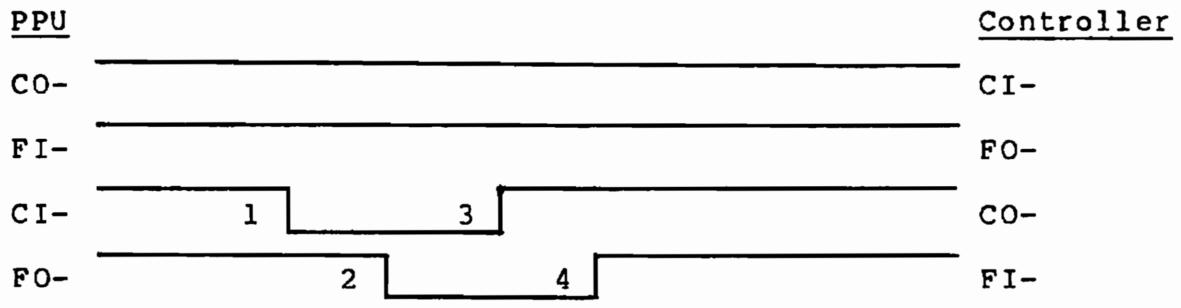
Under no circumstances may a controller or the PPU assert both command-out and flag-out at the same time. This would indicate the unit is both sending and receiving data simultaneously which is not allowed.

PPU - Controller Command-Status Handshake

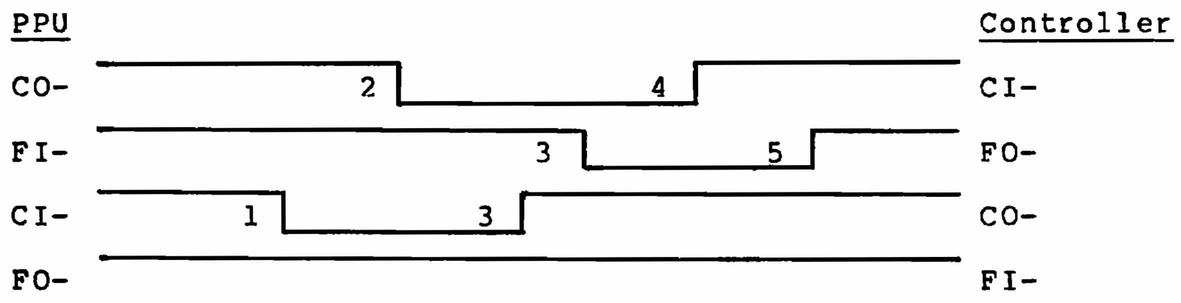
CASE A. PPU sends a command to the controller



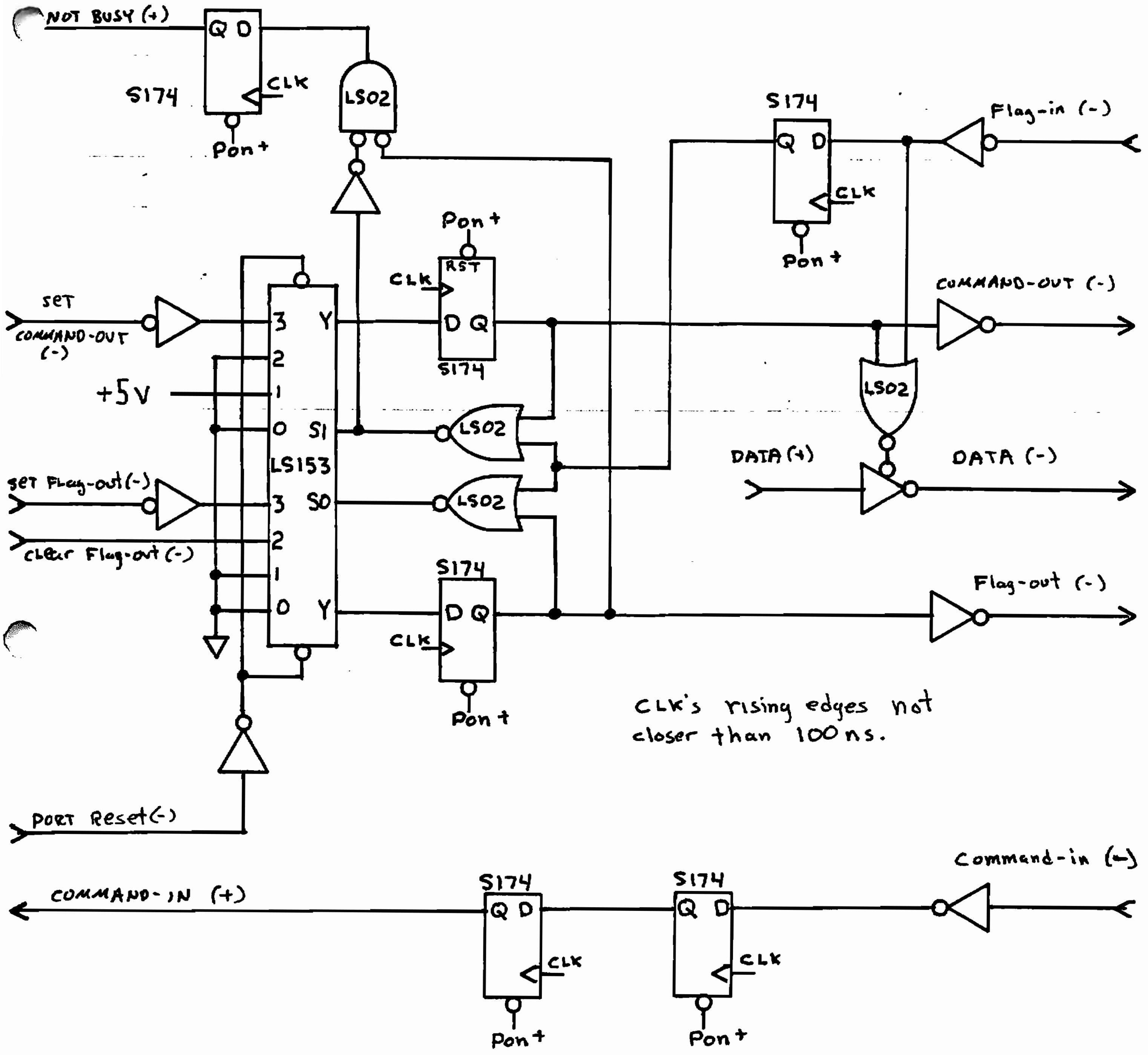
CASE B. Controller sends status to the PPU



CASE C. Controller tries to send status, PPU ignores it and sends a command, Controller takes command.



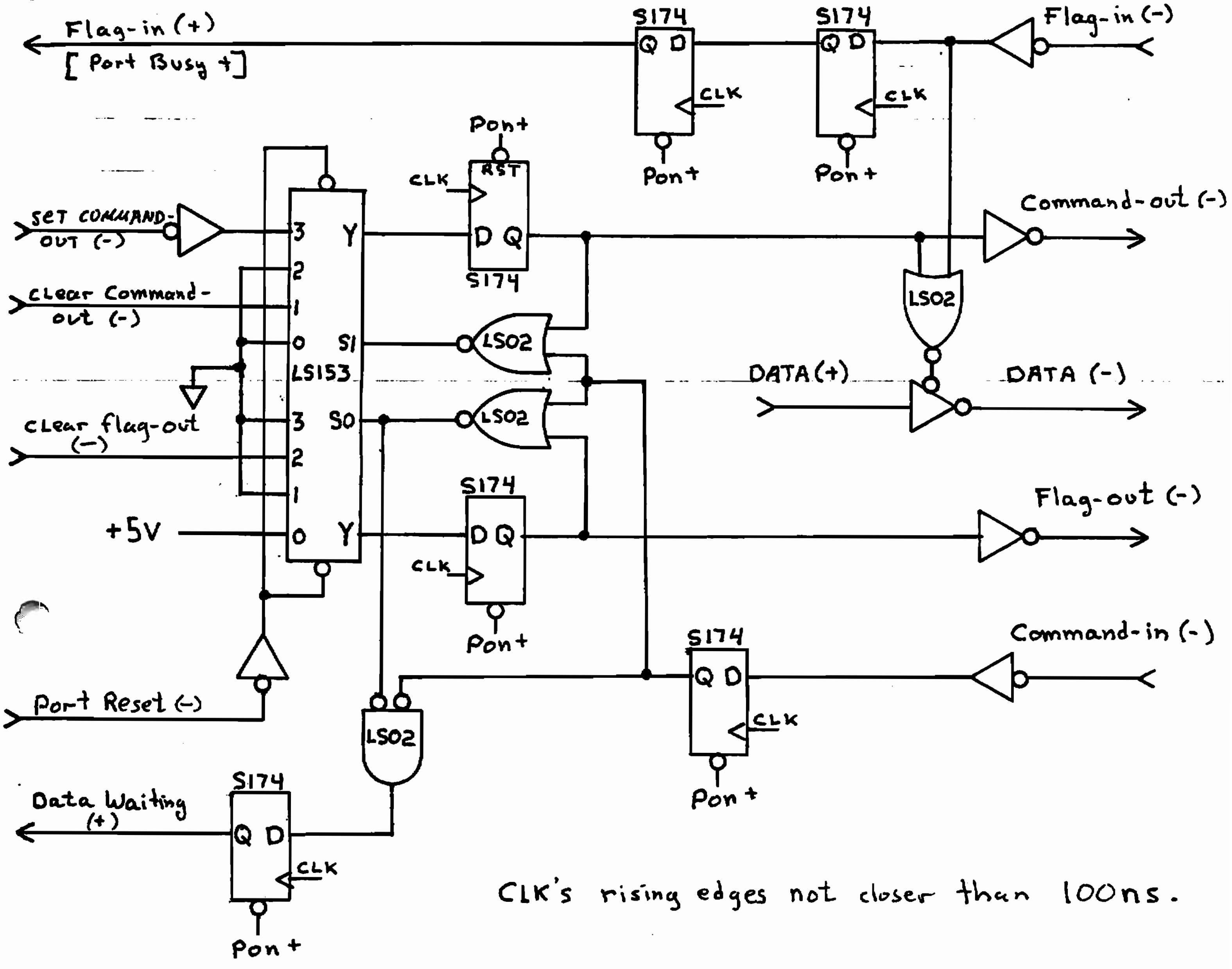
PPU's COMMAND STATUS HANDSHAKE



This circuit implements the System 8ØØØ handshake from the PPU's end, in that PPU may choose to ignore COMMAND IN, and may always set COMMAND OUT provided the last transfer was complete. The PPU must be smart enough to not take the data or clear flag-out until COMMAND IN has gone away.

This circuit is not necessarily used, but reflects the logic involved in the handshake.

CONTROLLER'S COMMAND STATUS HANDSHAKE

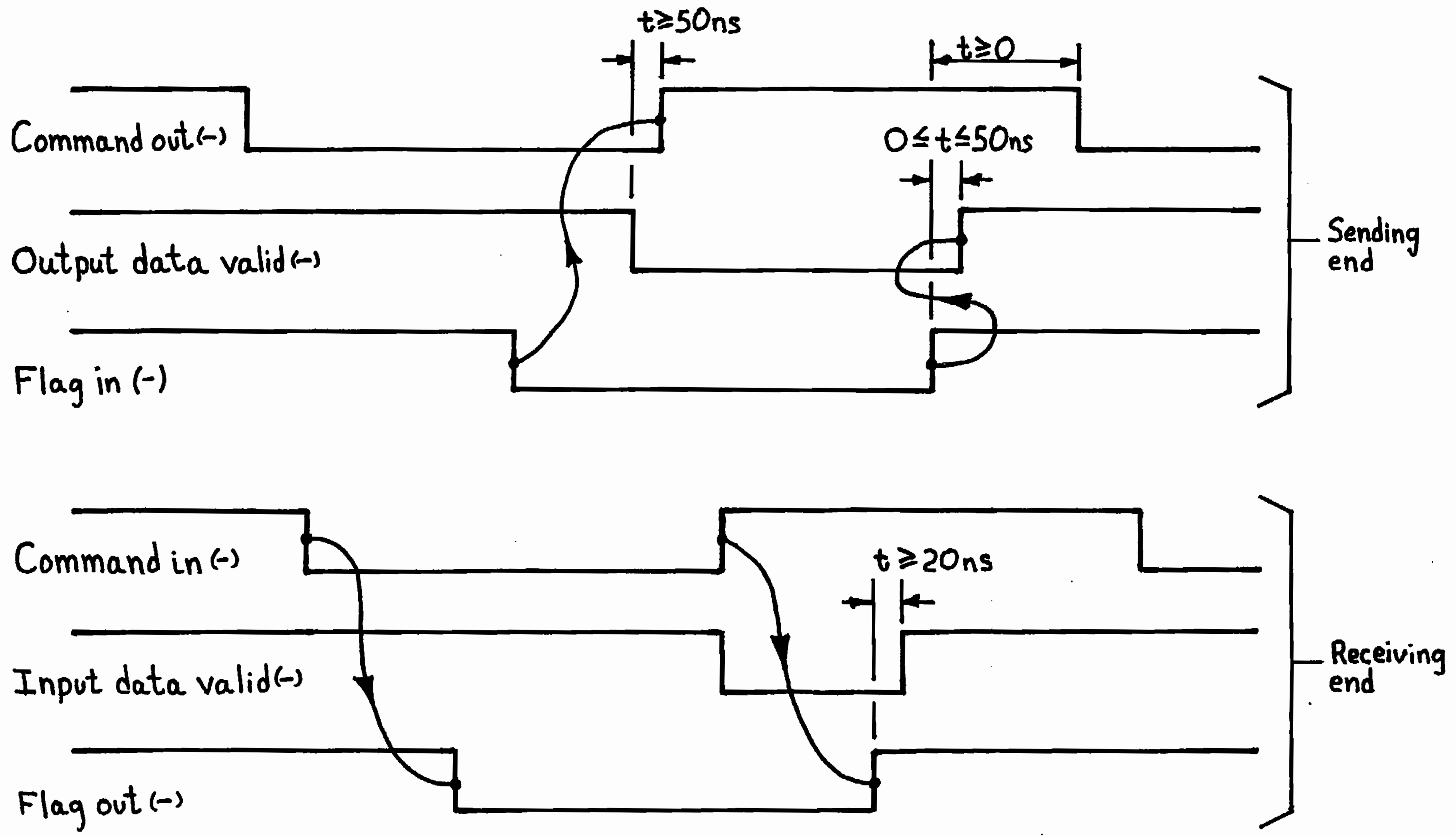


CLK's rising edges not closer than 100ns.

This circuit implements the System 8ØØØ handshake from a controller's end, in that the controller must always answer command-in, but may not get a response to command-out. The controller knows it is sending data when it sees flag-in and must be smart enough not to send more data until it sees flag-in go away.

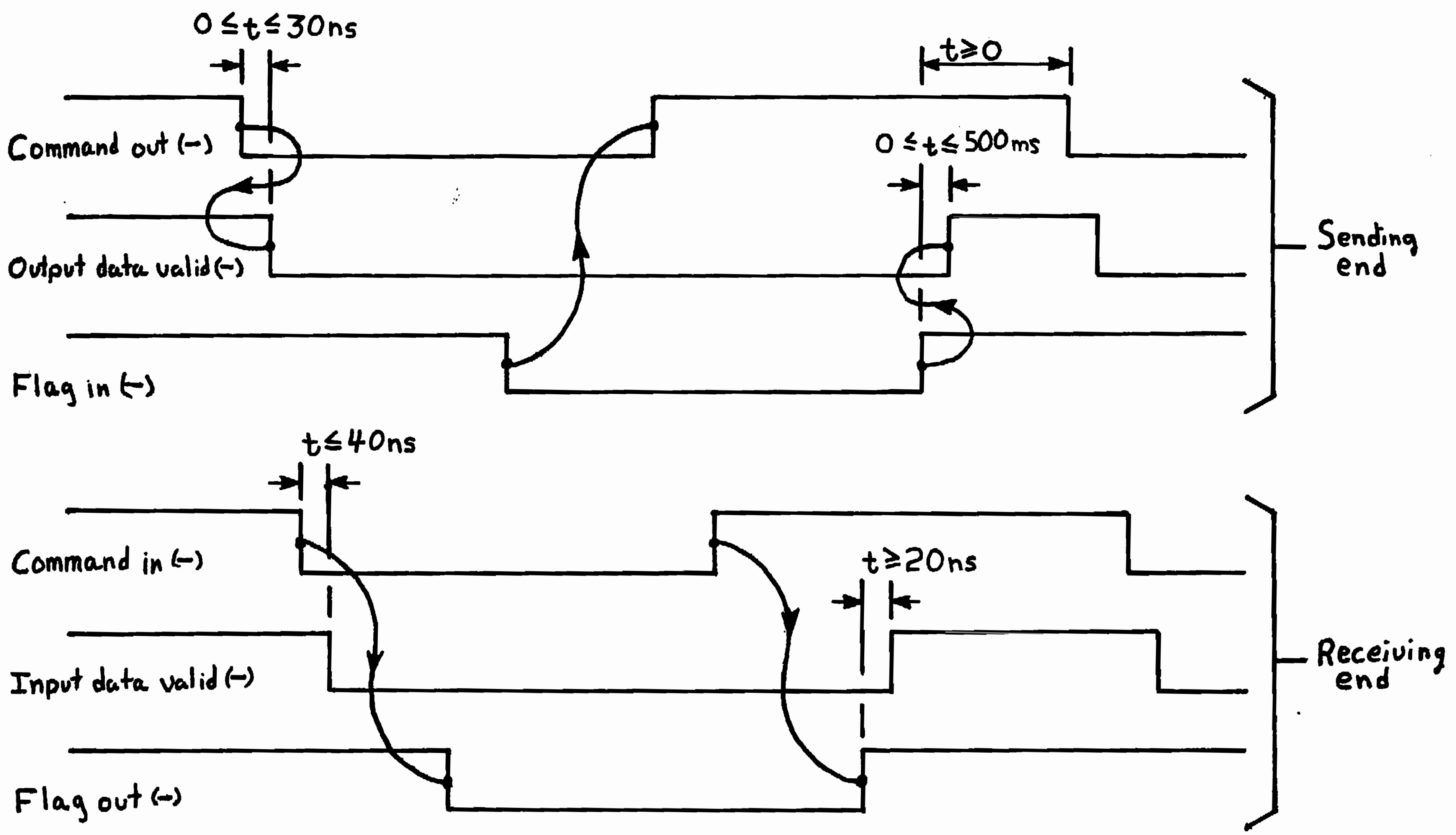
This circuit is not necessarily used, but reflects the logic involved in the handshake.

Timing Specification for the Command-Status Path



Times measured at test point Ⓣ

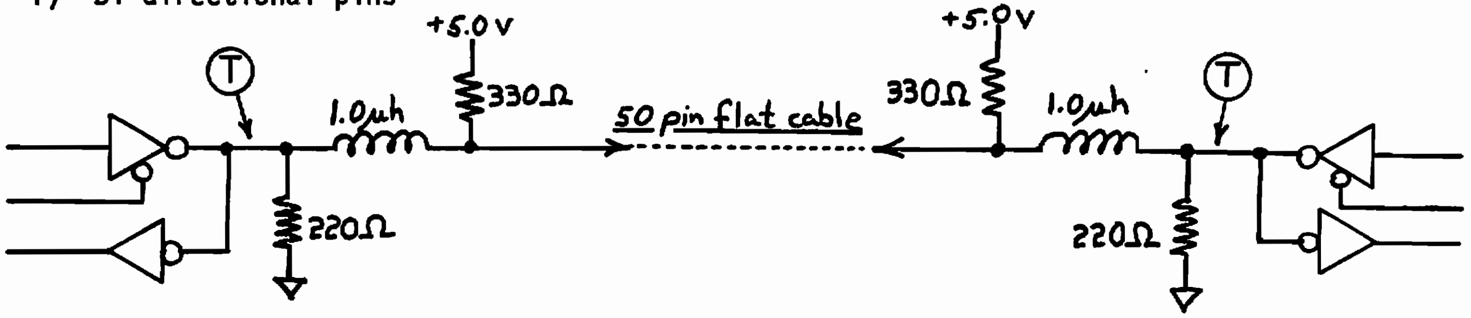
Timing Specification for the Data Path



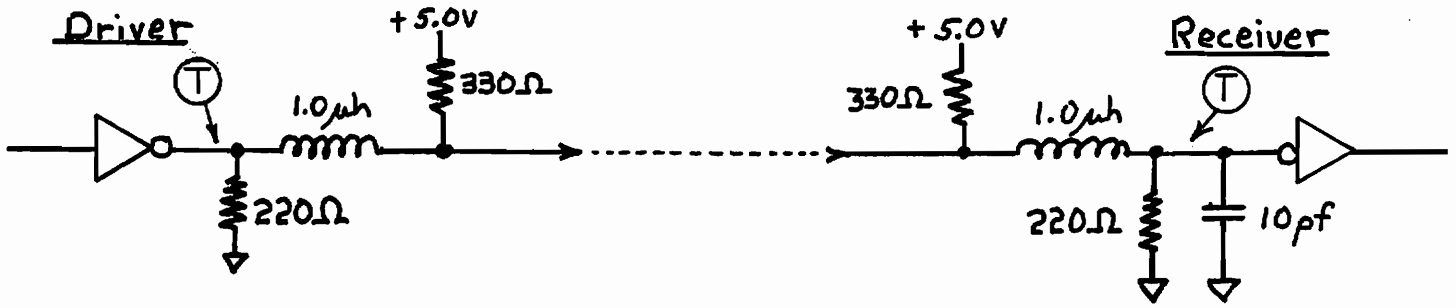
Times measured at test point Ⓣ

Electrical Specification for the Command-Status and Data Sub-Channels

1) Bi-directional pins



2) Uni-directional pins (the handshakes)



Voltage Levels

Vol $\leq$ 0.5 V

Voh $\geq$ 2.25 V

Drivers

Must be able to sink 24mA @ $\leq$ 0.5 V

Must be able to source 7mA @ $\geq$ 2.25 V

(if 74S24X, place only 4 drivers in one package due to max. power dissipation)

Receivers

$I_{il} \leq 500\ \mu A$ @ 0.5 V

Typical Propagation Delays

Through both filters with a diminishingly short cable connecting them:

$$T_{plh} \approx T_{phl} \approx 14\text{ns}$$

Add approximately 1.6ns/foot of cable between the two filters

PPU - Controller Command-Status Bus Firmware Protocal

All control related communication between a PPU and a Controller use the Command-Status bus between the two devices. Any string sent on the Command-Status bus consists of three parts :

1. A first byte that identifies the type and length of the string. This byte is called the Transmission IDentifier, or "TID".

2. The information to be sent, anywhere from 0 to 5 bytes long. (see appropriate controller documentation for what information may be sent in these bytes).

3. The Check byte used to insure odd vertical parity. This byte is called a Block Check Character, abrieviated "BCC".

A device that is receiving a string will check the first byte to see what type and how long the string is. After the transfer is complete, the receiving device will also check the vertical parity to be sure it received the string correctly. When a string is properly received the receiving device issues an "ACK" byte back to the sending device, showing acceptance of the string. When a string is received with an undefined first byte or bad parity, the receiving device sends a "NACK" back to the sending device, showing rejection of the string.

The low order 3 bits in every first byte contain the number of bytes <u>between</u> the first byte and final check byte.

Strings are sent one byte at a time using the two handshake lines : Command-out and Flag-in.

Firmware Constraints for the Command-Status Path

1. When a PPU is sending a string to a controller, it must not acknowledge a CI (i.e.; send a F0 to that controller) until all the bytes in that string have been accepted by that controller.

2. A controller will not accept input until it is able to act on it.

3. Unless executing a previous command, a controller will be fast enough to accept a string of 8 bytes from the PPU and ACK or NACK it within 10ms of the first CI.

4. A controller will not issue a NACK or an ACK upon receiving a 1 byte transfer that is a NACK or an ACK.

5. All strings except ACK & NACK will be an even number of bytes long, including first byte and check byte. i.e. A 5 byte message (7 byte string with first byte and check byte) will have an extra byte before the check byte. This byte is undefined but must be included in all check byte calculations. ACK and NACK are single byte transfers with no check bytes following them.

6. If a device is outputting or has just finished outputting a string or is expecting an ACK or NACK and receives something different it should assume it received a NACK. If the device is a controller and the received byte was a valid first byte the controller should also accept the PPU's string. That is, a first byte from the PPU may be both a NACK and a first byte as described on the next page.

7. When a device is receiving a string it should
   a) send a NACK if the first byte isn't legal (see #4)
   b) XOR all the bytes together
   c) test the result for "FF"
   d) if the result is not "FF", send a NACK and ignore the string
   e) if the result is "FF", send an ACK and take the appropriate action.

8. When a device is sending a string and receives a NACK it should try to send the string again, unless it has exceeded its retry count. The retry count is device dependent.

First Byte Definitions

A. From PPU to Controller

1.	96	Operating System sending a word to the Controller.
2.	62	Operating System requesting a status from a Controller.

B. From Controller to PPU

1.	34	Controller updating its secondary status word.
2.	60	Controller notifying PPU of its readiness to accept data over the data path (commonly known as a Go byte).
3.	50	Controller requests PPU to interrupt Operating System. (commonly known as an interrupt byte).
4.	54	Controller updates Primary Status and requests PPU to interrupt Operating System.
5.	90	Controller notifies PPU that it believes the data transmission over the data path to be completed. (commonly known as an EOR or End of Record byte).
6.	A4	Controller responding to a Status Request.
7.	C4	Controller updating its Primary Status word.

C. Bi-directional

1.	F8	NACK : did not accept incoming string.
2.	0E	ACK : did accept incoming string.

PFW and RESET

PFW-

The POWER FAIL WARNING signal is availiable to all units associated with the System 8000 computer. It is active low to warn of an impending power failure or system boot or system reset. Only those controllers which write onto mass storage media need examine this signal. Just before initiating a lengthy write procedure, a controller should check PFW- and not begin the write procedure if a power loss / system reset is imminent since the write probably would not have time to complete anyway.

RESET-

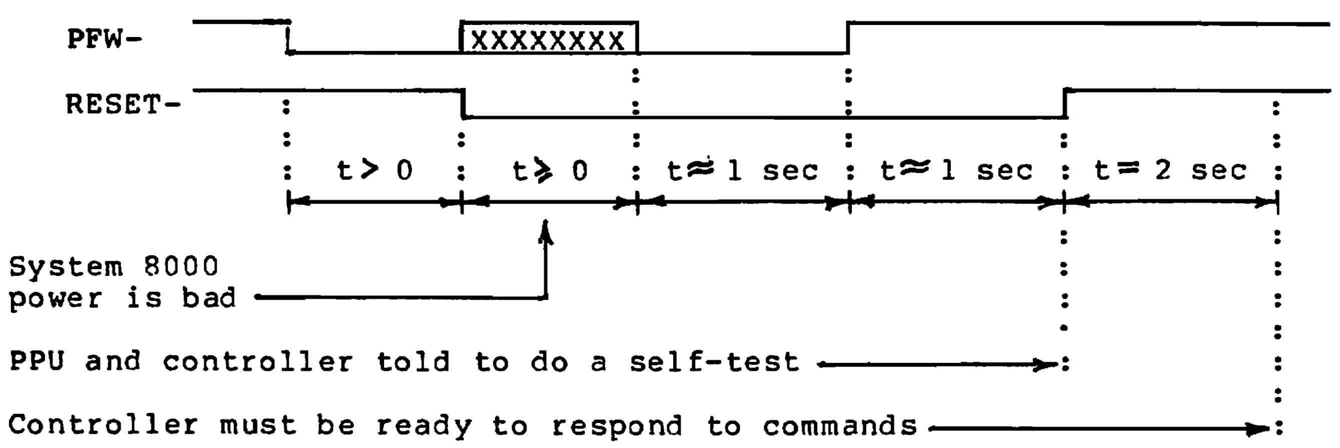
This signal is a command to a controller to do a hard reset to itself. In particular, all the handshake lines in both the controller and the PPU must unconditionally be held inactive. In addition, the controller should be reset in a manner similar to how it wakes up when power is first turned on.

RESET- is used to perform two functions. Its primary function is to hold the handshake inactive and the controller stopped while the power is bad in the System 8000 mainframe. Secondly, the operating system may instruct the PPU to do a reset to a particular controller as a last chance effort to straighten out a controller that has accidentally gotten itself into a very confused state.

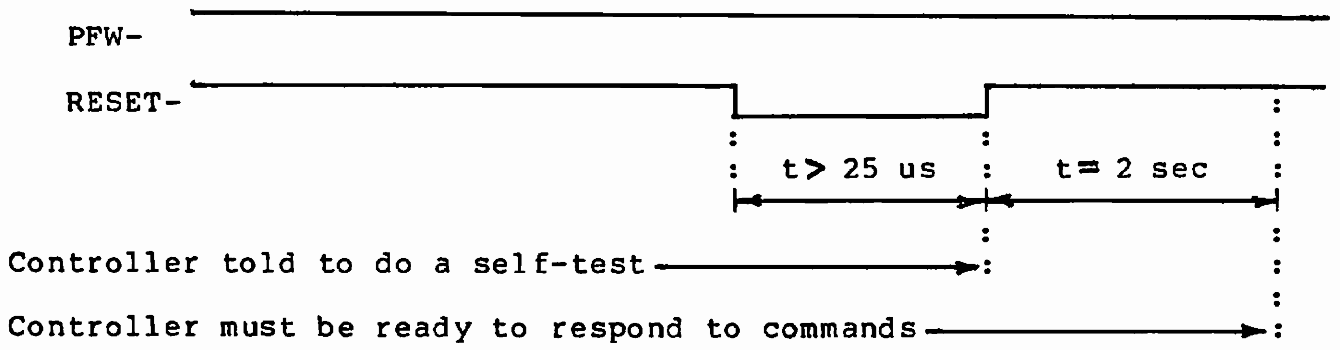
Following the release of RESET-, a controller must perform its self-test(s) and be ready to respond to the PPU within 2.0 seconds of RESET- being released.

Timing for PFW- and RESET-

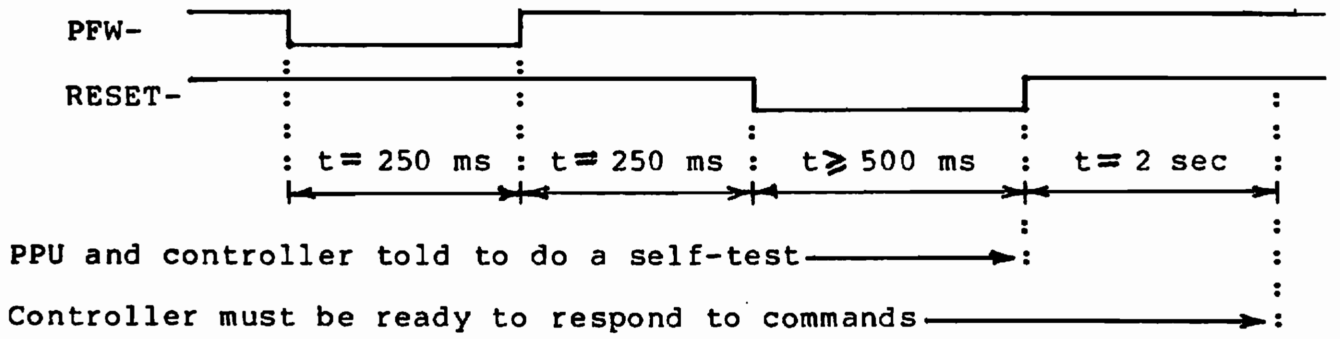
Power failure


PFW-
RESET-
XXXXXXXX
t > 0
t ≥ 0
t ≈ 1 sec
t ≈ 1 sec
t = 2 sec
System 8000 power is bad
PPU and controller told to do a self-test
Controller must be ready to respond to commands

Particular controller reset


PFW-
RESET-
t > 25 us
t = 2 sec
Controller told to do a self-test
Controller must be ready to respond to commands

System boot or reset

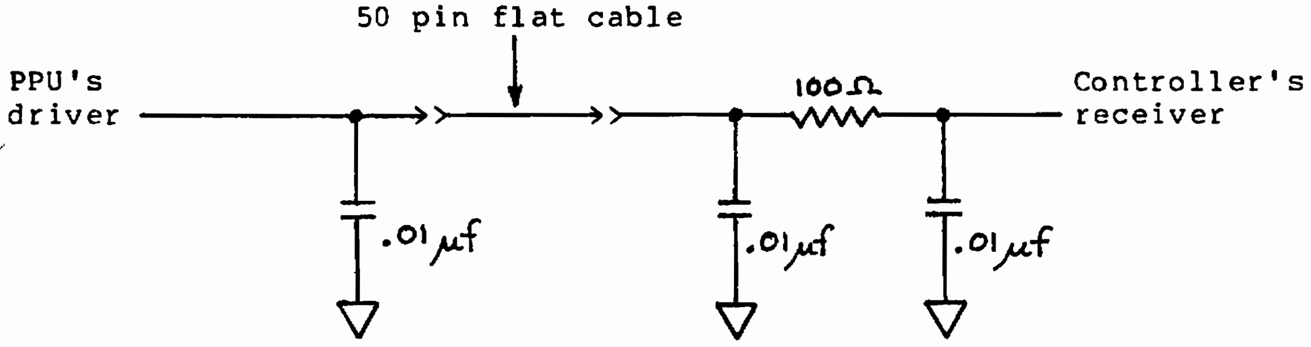

PFW-
RESET-
t = 250 ms
t = 250 ms
t ≥ 500 ms
t = 2 sec
PPU and controller told to do a self-test
Controller must be ready to respond to commands

Electrical specifications of PFW- and RESET-

PFW-	* RESET-
Vl $\leq$ 0.5 V	Vl $\leq$ 0.8 V
Vh $\geq$ 2.4 V	Vh $\geq$ 3.75 V
Ih $\geq$ 200 uA @ 2.4 V	Ih $\geq$ 2.0 mA @ 3.75 V
Il $\geq$ 1.0 mA @ 0.5 V	Il $\geq$ 250 uA @ 0.8 V

* A controller's RESET- must float to a valid low logic level to guarantee that a controller which is not cabled to the System 8000 will be held reset.

Required filtering on both PFW- and RESET- :



The filtering is to maintain the signal at an A.C. ground since adjacent signals in the 50 pin flat cable are high speed signals. By the time PFW- and RESET- get to a controller, the rise / fall time is quite slow and this must be considered in the detect circuitry.

Reset within the controller must be valid regardless of the state of that controller's power supply(s). A controller must hold itself reset if its own power is bad.