AN1208

Parallel I/O Interface to the NEURON® CHIP

INTRODUCTION

This application note describes the parallel I/O object of the MC143150 and MC143120 NEURON CHIPS, including specifics on the handshaking and token passing process used to establish synchronization and prevent bus contention. Examples are provided for interfacing to both a foreign processor (non–NEURON microprocessor or microcontroller) and other NEURON CHIPS. Timing, interrupts, and memory allocation are also discussed.

Utilizing this application, the NEURON CHIP can act as a communication chip for the foreign processor or can create a bridge, gateway or router. Figure 1 demonstrates typical applications for the NEURON CHIP utilizing the parallel I/O object.

The parallel I/O object employs all eleven I/O pins, eight for information exchange and three for control. No other I/O objects of the NEURON CHIP may be used in conjunction with parallel I/O.

For increased design flexibility, the NEURON CHIP provides three modes of operation for the parallel I/O object: master, slave A, and slave B. The different attributes of each mode can be used to tailor the NEURON CHIP for a specific application.

The NEURON CHIP master mode is the intelligent mode of the parallel I/O object. Refer to Figures 2a and 2c. In this mode, the NEURON CHIP can initiate and establish synchronization with the slave. The slave must be either a NEURON CHIP configured in slave A mode or a foreign processor emulating slave A mode.

A NEURON CHIP in slave A mode implements a hardwired handshake line (HS). The HS line and data are available in the same clock cycles. Although this mode was designed to interface with a NEURON CHIP master, either a foreign processor or another NEURON CHIP can act as the master. See Figures 2a and 2b.

The NEURON CHIP slave B mode is logically similar in operation to the slave A mode; however, the handshake is read from the slave's control register in one cycle and the data is available in a separate cycle. The slave B mode was designed to make the NEURON CHIP act like a peripheral device on a non–NEURON address bus. The master must be a foreign processor as the NEURON CHIP master mode is designed to interface to a slave A configuration. See Figure 2d.

The NEURON C programming language provides several built–in functions that enable the use of the parallel I/O object without the need for a detailed, hardware–level knowledge of the handshaking protocol. These functions are discussed in detail in the NEURON CHIP–to–NEURON CHIP interface section of this document.

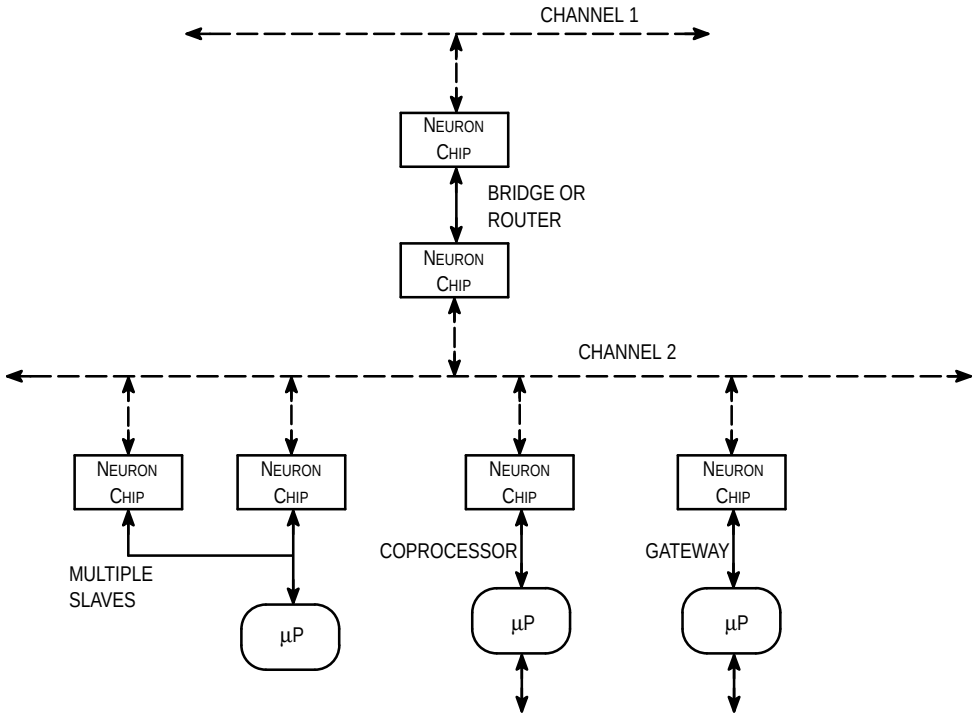
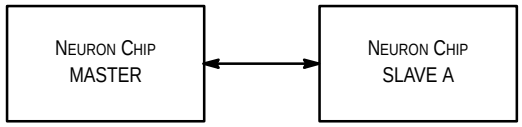
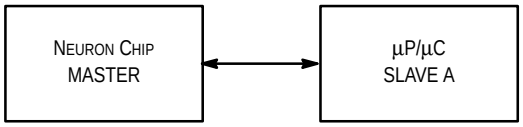


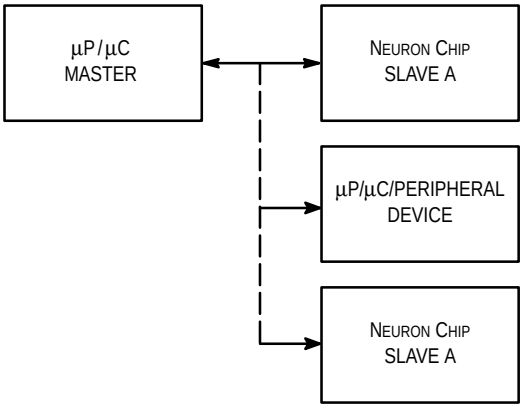
Figure 1. Applications Utilizing the NEURON CHIP Parallel I/O Interface



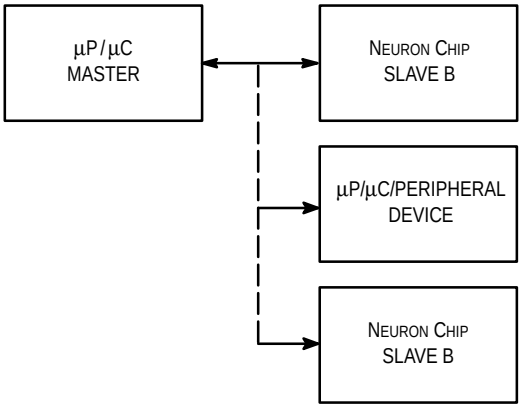
a. NEURON CHIP–to–NEURON CHIP



c. NEURON CHIP Master/Foreign Processor



b. Foreign Processor Master/Slave A



d. Foreign Processor Master/Slave B

Figure 2. Possible Master/Slave Connections for the NEURON CHIP

In a non–NEURON CHIP (foreign processor) interface, it is assumed that the microprocessor or microcontroller involved has the ability to execute the handshaking/token passing algorithm dictated by the attached NEURON CHIP. This usually consists of a hardware interface and a software program that duplicates the actions of a NEURON CHIP.

A foreign processor master can interface to a NEURON CHIP configured in slave A (Figure 2b) or slave B (Figure 2d) mode. In slave B mode, the foreign processor master reads the HS bit on the data bus by accessing the control register. In slave A mode, the HS line can be read using several different approaches. See also the ''Foreign–to–NEURON Processor Interface'' section.

Certain applications, such as a NEURON CHIP–to–NEURON CHIP connection, have only one solution (master to slave A).

Although several possible interfacing scenarios are shown in Figure 2, not all can be considered for every application.

ALTERNATIVES TO THE PARALLEL I/O INTERFACE

Echelon sells a licensed firmware, Microprocessor Interface Program (MIP), which supplies an alternative to parallel I/O interface. As in parallel I/O, MIP also requires a software intensive driver for the host processor. MIP was designed to accommodate systems with complex calculations or I/O, applications needing more than 62 network variables, and large network management applications. The MIP resides on the NEURON CHIP and no other application can be implemented. The MIP is faster than the parallel I/O object discussed in this application note, as no scheduler is used and fewer buffers are needed for data transfers. An Echelon sales representative can provide the license cost, cost per node, and additional information about the MIP.

A common way two microprocessors exchange information is utilizing a dual–port RAM. This concept can also be employed to allow data transfers between the MC143150 NEURON CHIP and a foreign processor. Details will not be discussed in this application note.

The NEURON CHIP also provides an asynchronous serial data format, as in Motorola's SCI, EIA–232 communication, called *serial input/output,* and a synchronous serial data format called *NEUROWIRE input/output,* which interfaces to Motorola's SPI. The serial interfaces are slower than the parallel interface but some applications may require a serial option.

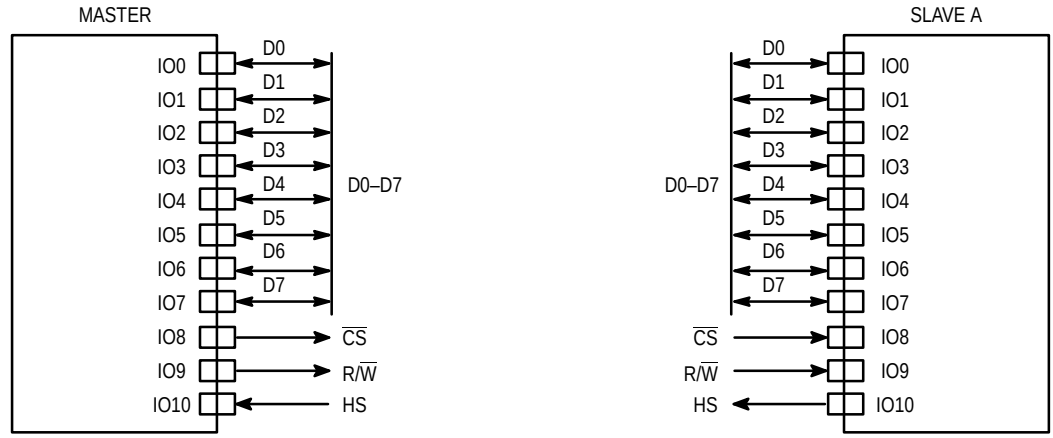
NEURON CHIP PARALLEL I/O INTERFACE

The NEURON CHIP parallel I/O interface consists of eight I/O and three control lines (see Figure 3).

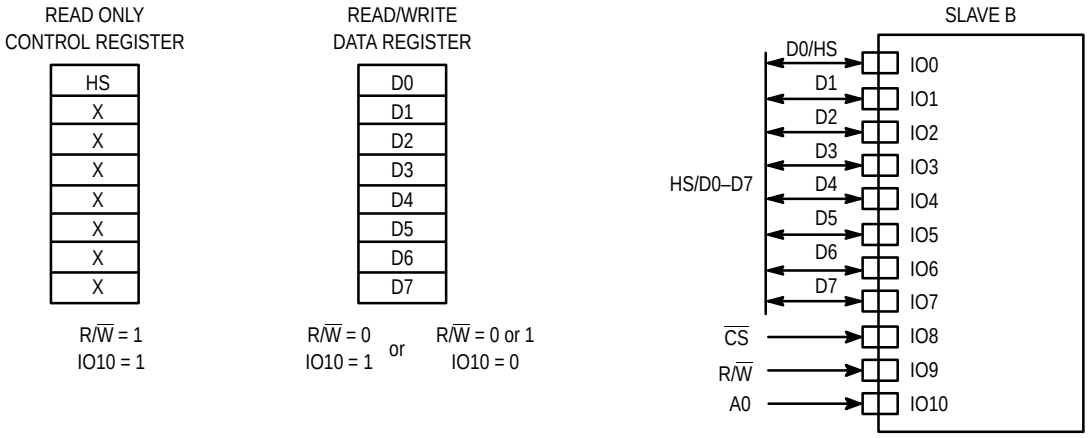
The $\overline{CS}$ line is always driven by the master and, when active, signifies that a byte transfer operation is currently in progress. A low pulse on this line strobes the data into either the master or slave. (Refer to Figures 8, 9, and 10.)

The type of data transfer actually taking place, either a read or a write (with respect to the master), is assessed by the level of the R/$\overline{W}$ line at the time the $\overline{CS}$ line is pulsed low. The R/$\overline{W}$ line is driven by the master.

The HS (handshake) output is always driven by the slave. It informs the master if the slave is busy. In effect the HS output can be treated as a slave–busy signal. When high, the slave is busy performing an action (read or write of a command or data); a low indicates it is ready for the next transaction. In slave A mode, HS is a physical pin and in slave B mode, HS is the least significant bit of the control register.



a. Connections for Master and Slave A Modes of Parallel I/O



b. Connections and Registers for Slave B Mode of Parallel I/O

Figure 3. Parallel Interface

The I/O10 pin is a register select pin, driven by the master for interface to the slave B mode. It can be the least significant address bit which selects between reads of the data register and the control register. An even address typically allows data transfers and reads of an odd address allow HS monitoring. The remaining bits of the control register are unused and indeterminate and therefore should be masked by the software.

It is possible for the master device to come online and poll the HS line before the slave has had a chance to set the proper level on this line. To prevent the master from reading invalid data on the HS line, a pull–up resistor should be used on the HS line of a slave A NEURON CHIP or the HS/D0 line of a slave B NEURON CHIP.

Token–Passing Protocol

A token–passing protocol implemented by the NEURON CHIP firmware permits the coexistence of multiple devices on a common bus. At any given time, only one device is given the option of writing to the bus. A virtual write token is passed alternately between the master and the slave on the bus in an infinite, ping–pong fashion. The owner of the token has the option of writing data, or alternatively, passing the token without any data. The token is not physically passed between the processors but is tracked with software. A token is acquired after a read cycle and relinquished after a write cycle. See also the ''NEURON C Resources'' section of this text.

Figure 4 illustrates the token passing operation between a master and a slave.

Multiple slaves on a common bus, with multiple write tokens, can also be supported by the token–passing protocol. In such a case, the master must keep track of all outstanding write tokens and accordingly direct bus traffic. Slaves may be selected round robin or on a priority basis. Uniquely assigned $\overline{CS}$ lines prevent bus contention.

Once in possession of the write token, a device may perform one of several operations (as shown in Figure 4): write data, pass token, resynchronize (master only), or acknowledge resynchronization (slave only).

The sequence of events for each of the above operations is always the same, for either the master or the slave (A or B). However, the degree to which the user is exposed to the underlying token–passing operations is varied depending on the actual device involved. Built–in tools within the NEURON C language allow for straightforward software coding of the NEURON CHIP. This translates to a transparent token–passing protocol, which in turn results in program simplicity.

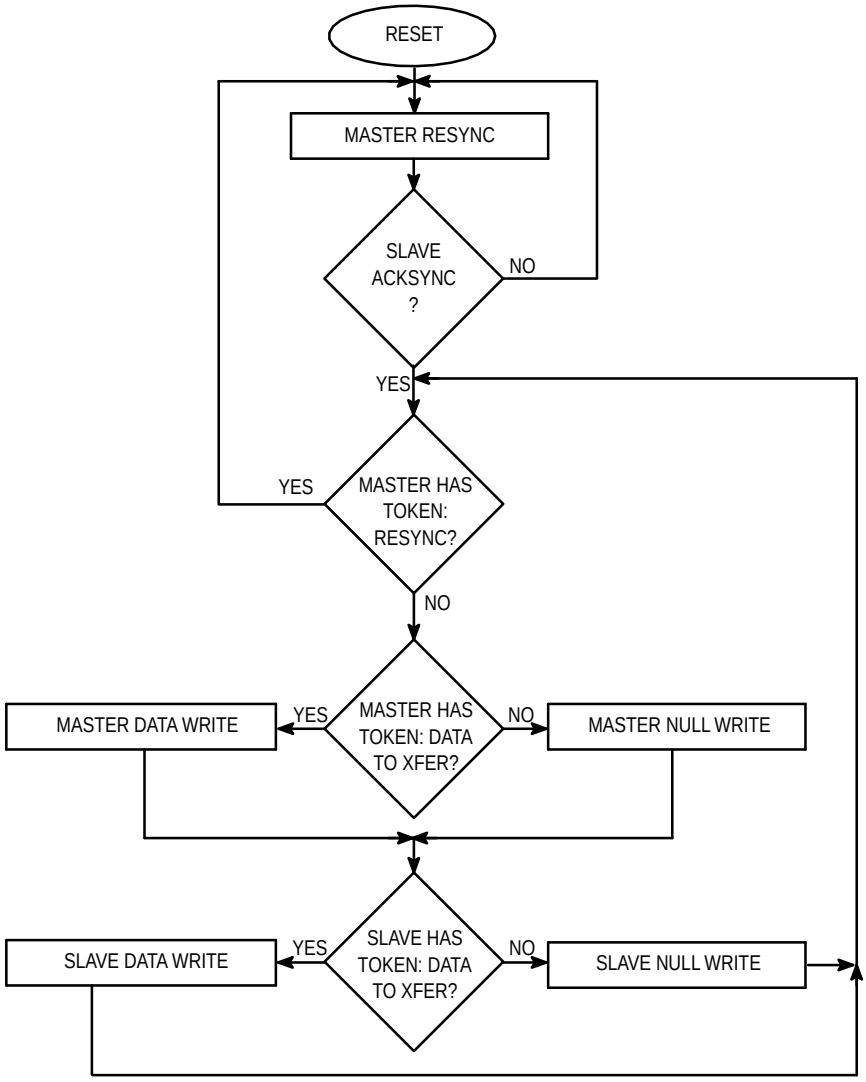


Figure 4. Token–Passing Protocol Sequence Between Master and Slave

On the other hand, if a NEURON CHIP is interfaced to a non–NEURON processor (foreign processor), the user must explicitly implement the handshaking/ token–passing protocol on the foreign processor side. Although the NEURON CHIP software remains straightforward, the data transfer rate may be affected by the additional cycle needed for the foreign processor to read the HS and the code needed to implement the token tracking.

Protocol Commands

The byte format of the command options available to the token holder are shown in Figure 5. Each command is made up of a fixed sequence of read and write operations to the bus by both the master and the slave. The state transition diagram for each command is shown in Figure 6.

These commands are the building blocks on which all communication between a NEURON CHIP parallel I/O and the outside world are based. Only one of the commands can be performed by the token holder at any given time. Upon completion of the command, the token is automatically owned by the other device. The other device now has the opportunity to execute a command. The write token is thus passed back and forth between the master and slave indefinitely.

The owner of the token (either master or slave, NEURON CHIP or foreign processor) can hold the token for an indefinite period of time. (Refer to Figure 4.) However, after passing the token to a NEURON CHIP (master or slave) a check must be implemented periodically to verify if the NEURON CHIP is ready to write to the bus. If the NEURON CHIP is a token–holding slave, the master should monitor the HS line for a low state indicating the slave is ready to output to the bus. If the NEURON CHIP is a token–holding master, the slave should toggle HS low to see if the master is ready to output to the bus. A NEURON CHIP watchdog time–out may occur if communication is not completed within approximately 840 milliseconds (10 MHz) after the NEURON CHIP I/O output is ready. The watch dog scales proportionally to the external clock.

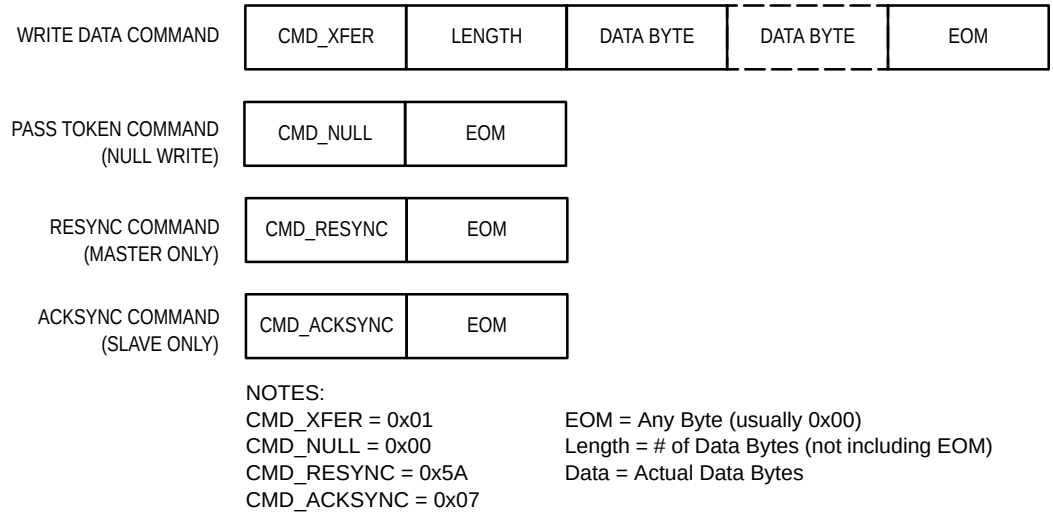


Figure 5. Commands Available to the Token Holder

If the NEURON CHIP slave receives any command byte other than CMD_XFER, CMD_NULL, or CMD_RESYNC it will go into a wait clause until a CMD_RESYNC is received or a watch dog time out occurs.

Read and write operations require a negative pulse (high to low to high) on the $\overline{CS}$ line (Figures 8, 9, and 10). For the slave B write operation, a high to low $\overline{CS}$ transition causes the slave B to put the data on the bus so that it can be latched (strobed) in by a master. For a NEURON CHIP slave A write, the NEURON CHIP continues to drive the data until a low pulse is detected on $\overline{CS}$, indicating the master has latched the data.

In the case of the master write operation, both the NEURON CHIP slave A and slave B read (strobe) the data on the rising edge of $\overline{CS}$.

The low to high transition of the $\overline{CS}$ causes the NEURON CHIP slave (A or B) HS signal to go high. The only exception to this is when the master reads the control register of a slave B. HS is unaffected in this case.

As shown in Figure 6, the EOM byte always terminates a command and is never read by the device it is sent to. The EOM transaction is just a write cycle and is used by the slave to toggle the state of HS at the end of a command in order to pass the write token.

Handshake

The handshake (HS) signal acts like a slave busy flag to ensure valid data transfers (Figures 8, 9, 10). Slave A has an external HS line and slave B write HS as a control bit in the control register. See Figure 3. The NEURON CHIP HS line is hardware controlled, not firmware controlled.

When the master executes a data transfer, the NEURON CHIP slave toggles the HS signal high. When the slave has completed reading a byte or is ready to write, HS is low. Therefore, HS = 1 indicates the slave is busy and valid data transfers can not be initiated by the master until HS = 0.

When a foreign processor is the master, HS must be explicitly polled by that processor's software routine to ensure HS is low before a read or write operation is initiated (controlled by the $\overline{CS}$ and R/$\overline{W}$ lines).

Synchronization

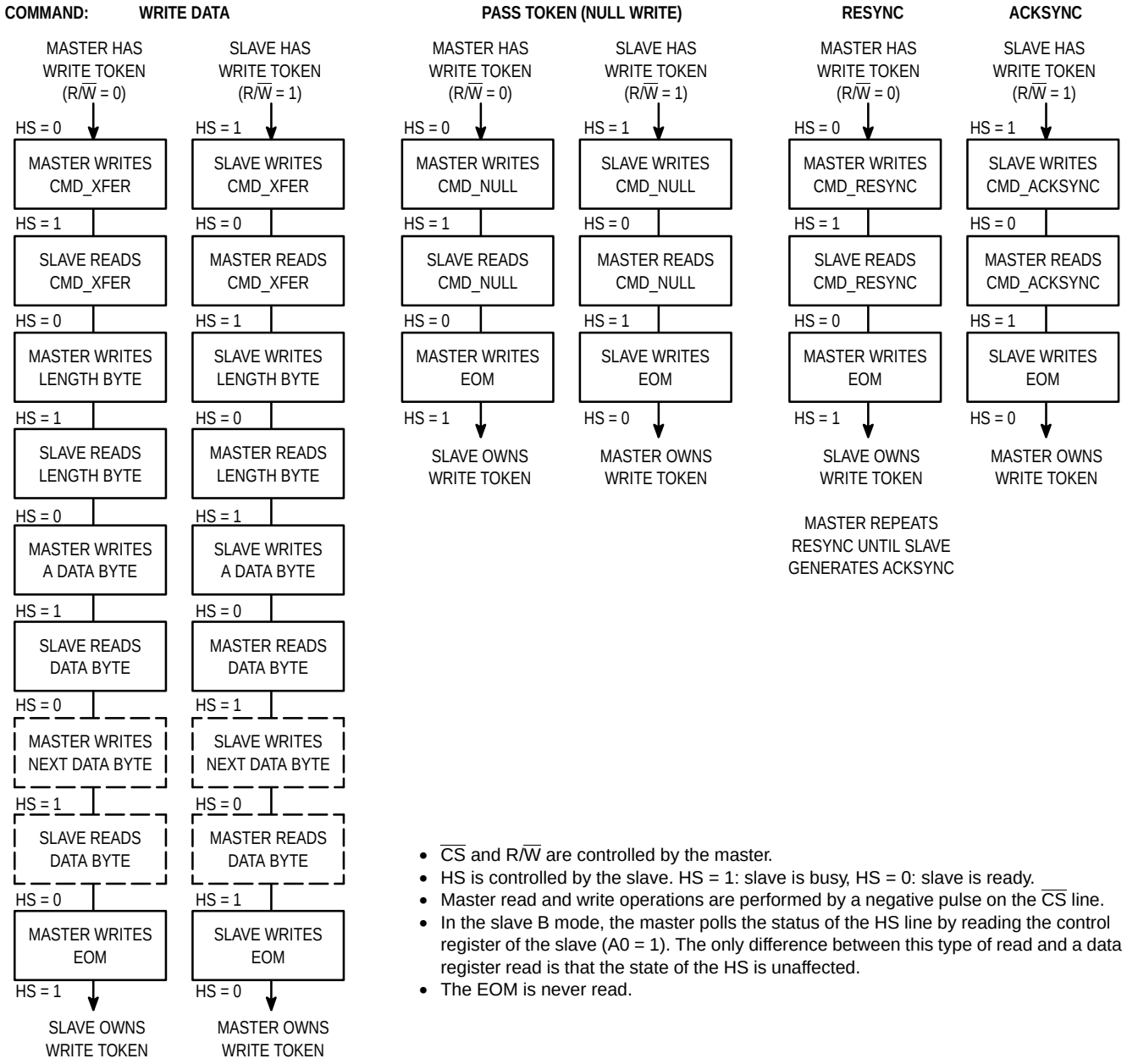
Upon a NEURON CHIP reset, the write token is, by definition, in the possession of the master. Synchronization across the parallel bus is required by the NEURON CHIP following any reset condition. The purpose of synchronization is to ensure both the master and slave are ready for data transaction. Synchronization prevents false starts of data transfers or incorrect data transfers. This is automatically accomplished by the NEURON CHIP through the use of a synchronization sequence.

The NEURON CHIP's automatic synchronization process occurs just before the reset clause of the application program is executed, and just after configuration of the NEURON CHIP's I/O pins. Prior to the synchronization sequence, the I/O pins are configured as inputs.

The automatic synchronization sequence carried out by the NEURON CHIP is dependent on the mode of its parallel I/O object. If the NEURON CHIP is a master, it will initiate a resynchronization command upon reset. If the NEURON CHIP is a slave (A or B), it will await the arrival of a resynchronization command from the master (any other command will be ignored).

The parallel I/O object provides the capability to synchronize the devices at any time when a foreign processor is the master. This enables the foreign processor to ascertain the integrity of the communication medium and reestablish a predetermined state. Aside from the initial synchronization necessary after a reset, a foreign processor is not required to perform this operation at any other time. The capability, however, is provided for the system designer in case a need does arise.

The resynchronization operation can be initiated by the token–holding master at any time by the use of the RESYNC command. The RESYNC command sends a special message (CMD_RESYNC) to the slave, which in turn triggers it to send its own special message (CMD_ACKSYNC) back to the master. Thus, a two–way communication has taken place and the token has been passed from the master to the slave and back to the master again.



- $\overline{CS}$ and R/$\overline{W}$ are controlled by the master.
- HS is controlled by the slave. HS = 1: slave is busy, HS = 0: slave is ready.
- Master read and write operations are performed by a negative pulse on the $\overline{CS}$ line.
- In the slave B mode, the master polls the status of the HS line by reading the control register of the slave (A0 = 1). The only difference between this type of read and a data register read is that the state of the HS is unaffected.
- The EOM is never read.

Figure 6. Micro–Operations of the Handshake Protocol

The operations described by Figure 6, including the synchronization operations, are transparent to the NEURON CHIP application programmer. They are automatically executed by the NEURON CHIP's firmware. When interfacing a foreign processor to the NEURON CHIP, however, the above–mentioned operations must be explicitly carried out by the attached processor.

Reset

Depending on the user application, the reset lines may need to be monitored to ensure the integrity of the transmission. The foreign processor master reset can directly control the NEURON CHIP slave reset. However, the master might handle a slave reset with an interrupt service routine. A reset circuit is shown in Figure 7.

MC68HC11 to NEURON CHIP Interface Reset Circuitry

Reset signals from and to the NEURON CHIP are handled by additional logic as shown in Figure 7. There are two sources of reset for the MC68HC11 and the NEURON CHIP. One source internally generated by the MC68HC11 or NEURON CHIP and the second source externally generated by a Low Voltage Inhibit (LVI), for example, an MC33164 or a push–button reset switch.

The MC68HC11 may reset the NEURON CHIP but not vice–versa.

Additionally, resets may come from the NEURON CHIP by a network management command being received over the LONWORKS network. This network management command causes the reset pin on the NEURON CHIP to become an output and be pulsed low for a short period of time. Due to the

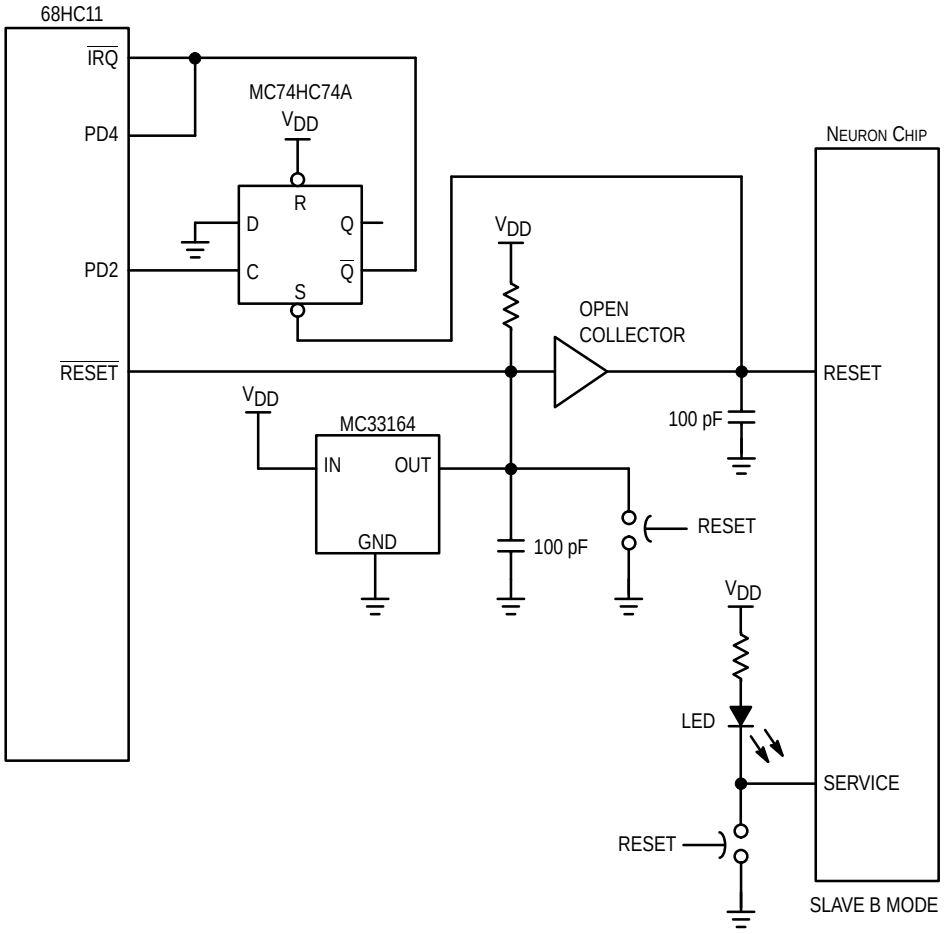


Figure 7. Reset Scheme for the NEURON CHIP Interfacing to a MC68HC11 Processor

short duration of this pulse, this reset condition must be latched (for instance, a 74HC74 D flip–flop). The output of the D flip–flop is then used to interrupt the MC68HC11 to notify the application program of this network management command. Since this signal is an interrupt to the MC68HC11, the IRQ pin must be held low until the interrupt is acknowledged by the interrupt service routine. The interrupt is then cleared by setting PD2 I/O pin low and restoring it back high in the interrupt service routine. Optionally, in case of multiple IRQ interrupts, the output of the flip–flop may also be used as an input to another I/O pin (such as PD4) so that the interrupt service routine may determine the source of the IRQ interrupt.

The open collector device between the MC68HC11 reset pin and the NEURON CHIP reset pin is used to prevent a NEURON CHIP source reset from resetting the MC68HC11. When designing the reset circuit several factors must be taken into consideration, these include:

- How much current the NEURON CHIP can source.
- The saturation voltage of the LVI. This voltage will be current dependent.
- The voltage level the NEURON CHIP will reset.
- The voltage level the NEURON CHIP will output a reset.
- The current level any LEDs will turn on.
- Voltage drops across all components including diodes and resistors.
- Any time constants (ex: RC networks).
- Saturation voltage of the open collector device.

Other reset circuits could be designed to fit specific applications. See also the appropriate technical data sheets for the suggested power_on_reset circuits and other reset issues.

NEURON C RESOURCES

The NEURON C programming language allows access to the parallel I/O object. The following section describes the available resources within the NEURON C programming language.

The parallel I/O object is declared in a NEURON C program using the following syntax (more details are given in Figure 12 of this document and the *LONBUILDER NEURON C Programmer's Guide* [Motorola document order number NEURONCPG/AD].)

```
IO_0 parallel slave/slave_b/master io_object_name;
```

The functions `io_in` and `io_out` are used as parallel reads and writes, respectively. To use the parallel I/O object of the NEURON CHIP, `io_in` and `io_out` require a `parallel_io_interface` structure as defined below:

```
Struct parallel_io_interface {
 unsigned length;          //length of data field
 unsigned data[maxlength]; //data field
 }pio_name;
```

The previous structure must be declared, with an appropriate definition of *maxlength* signifying the largest expected buffer size for any data transfer.

In the case of io_out, *length* is the number of bytes to be transferred out and is set by the user program. In the case of io_in, *length* is the maximum number of bytes to be transferred in. If the incoming length is larger than *length* then the incoming data stream is truncated to length bytes. The length field must be set before calling either io_in or io_out. The maximum value for the *length* and *maxlength* field is 255.

The parallel I/O object of the NEURON CHIP is easily accessed with the use of built–in NEURON C functions and events. The following functions and events are provided specifically for use with the parallel I/O object:

- io_in_ready This event becomes TRUE whenever a message arrives on the parallel bus that must be read. The application must then call io_in to retrieve the data.
- io_out_request This function is used to request an indication for an I/O object. It is up to the application to buffer the data until the io_out_ready event is TRUE.
- io_out_ready This event becomes TRUE whenever the parallel bus is in a state where it can be written to and the io_out_request function was previously invoked. The application must then call the io_out function to write the data to the parallel port.

NEURON C applications may be written that use the parallel bus in a unidirectional manner (i.e., applications may be written without either an io_in_ready or io_out_ready when clause). In the case where no io_in function exists, it is up to the programmer to assure that no read transfers of real data messages will ever be required by the application. This is to protect the device on the other side of the bus from waiting forever on a data transfer.

If there is no data to be transferred, the programmer simply does not generate an in_out_request. No additional code is needed for passing the token (CMD_NULL). The CMD_NULL generation is part of the transparent token passings protocol of the NEURON CHIP.

TIMING

Figures 8, 9, and 10 give the detailed timing specifications for the parallel I/O object. All three modes of the object are included. Note that these are typical *observed* numbers and are not meant to replace actual device characterization.

RAM ALLOCATION

If transferring large packets of data from the communication port (network) through the I/O port, memory issues may be of concern. This section describes how to determine if the on–chip RAM is adequate for a specific application.

There are four types of buffers needed to move data between the application program and the communication port (network). They are: the network input buffers, the application input buffers, the application output buffers and the network output buffers. As shown in Figure 11, the network buffers allow communication between the media access control (MAC) processor and the network processor, and the application buffers allow the network processor and application processor to communicate.

The NEURON CHIP accepts messages from the network into the network input buffers, verifies the CRC, and interprets the destination address. Therefore, the size of the network input buffers must support the largest potential message transmitted over the channel in order to prevent error conditions. The LONBUILDER DEVELOPER'S WORKBENCH default size for each of these buffers is 66 bytes and the default count is two buffers on the MC143150. The default memory allocation for these buffers is on–chip RAMFAR.

In a condition when the count of NEURON CHIP input buffers is not sufficient to support traffic on a network, the NEURON CHIP does not overwrite data in the buffers, but instead ignores the incoming packet. If, for example, the network message is specified as an ''acknowledged service with retries'' the message is not lost immediately. The source NEURON CHIP instead continues to resend the message until either an acknowledgement is received or the maximum retries are sent.

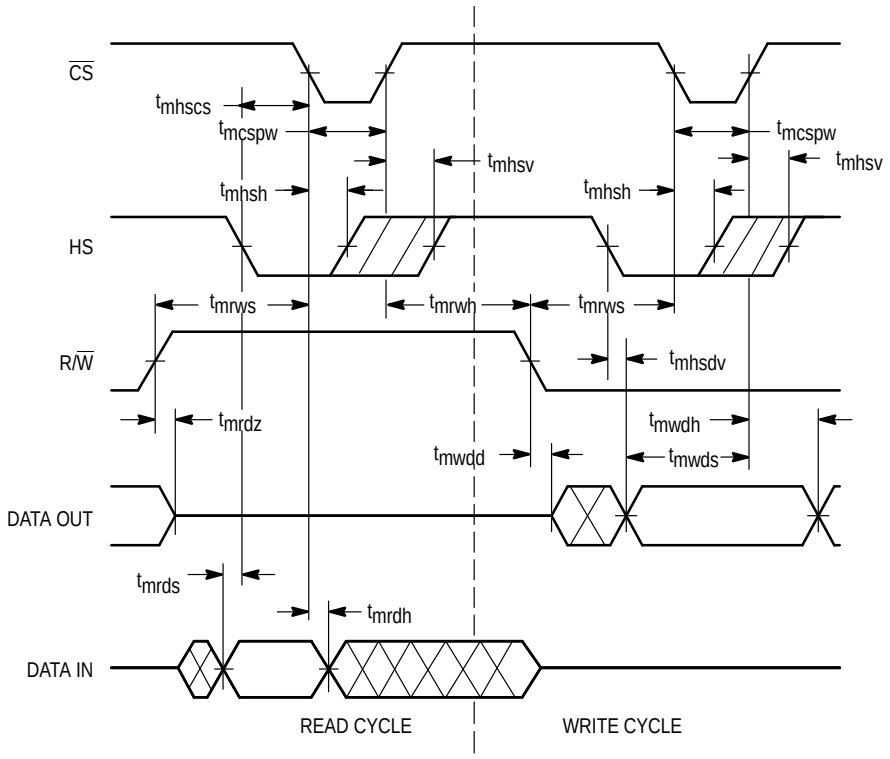
Likewise, the NEURON CHIP does not overwrite data in the output buffers. Refer to the documentation on the "Preemption Mode" in the *NEURON C Programmer's Guide* for details on specifing preemption. If the network is active, an increase in the count of buffers may be needed. In any case, data is never overwritten.

Application variables, including the parallel I/O application structure(s), are also stored in RAM. By default, user RAM is stored in the 256 bytes available in RAMNEAR; however, RAMFAR can also be employed for user RAM.

The MC143150 has 2 K of on–chip RAM. The LONBUILDER DEVELOPER'S WORKBENCH allocates memory for both the system data (i.e., stack) and for the transaction control block. The remaining memory available for the input and output buffers is potentially less than 1K, depending on memory needed for user RAM. The MC143120 has 1K of RAM and may not be suitable for channels with large packet transfers.

For example, if the value of 114 bytes is selected for all the four types of I/O buffers and the counts selected are two, then the total number of I/O buffers is eight. In this case, no external RAM should be needed if the user RAM does not exceed the 256 bytes of RAMNEAR. Note: 114 bytes $\times$ 8 (4 buffers $\times$ count 2) = 912 bytes needed for all I/O buffers. The overhead for the application buffers is maximum 7 bytes and the overhead for the network buffers is 25, therefore, maximum data length in this scenario is 89 (114–25) bytes. Refer to the memory management section of the *NEURON C Programmer's Guide* for allowed buffer values.

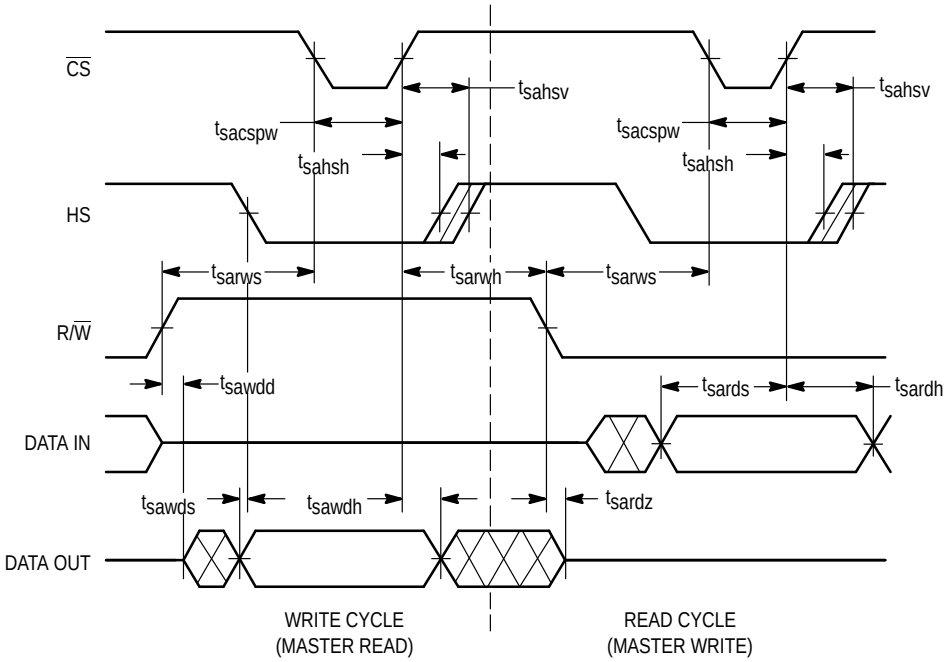
In some applications, large packets of data are transfered in only one direction. If the value assigned to the output buffers is 210 bytes, then the average of the input buffers could potentially be given the value of up to 42 bytes without external memory (210 bytes $\times$ 4 output buffers + 42 bytes $\times$ 4 input buffers = 1008 bytes total RAM needed). Again, the count is left to its default of 2. The maximum potential actual data output size is 185 bytes (210 – 25 overhead = 185). Note, the size of the network input buffers is determined by the largest potential packet transmitted on the channel. The application input buffers need only accomodate the largest packet destined for a particular NEURON CHIP.



Parameter	Description	Min	Typ	Max
t_{mrws}	R/$\overline{W}$ setup before falling edge of $\overline{CS}$	150 ns	3 CLK1	—
t_{mrwh}	R/$\overline{W}$ hold after rising edge of $\overline{CS}$	100 ns	—	—
t_{mcspw}	$\overline{CS}$ pulse width	150 ns	2 CLK1	—
t_{mhsh}	HS hold after falling edge of $\overline{CS}$	0 ns	—	—
t_{mhsv}	HS checked by firmware after rising edge of $\overline{CS}$	150 ns	10 CLK1	—
t_{mrdz}	Master three–state DATA after rising edge of R/$\overline{W}$	—	—	25 ns
t_{mrds}	Read data setup before falling edge of HS [3]	0 ns	—	—
t_{mhscs}	HS low to falling edge of $\overline{CS}$ [5]	2 CLK1	6 CLK1	—
t_{mrdh}	Read data hold after falling edge of $\overline{CS}$	0 ns	—	—
t_{mwdd}	Master drive of DATA after falling edge of R/$\overline{W}$	150 ns	2 CLK1	—
t_{mhsdv}	HS low to data valid [5]	—	50 ns	—
t_{mwds}	Write data setup before rising edge of $\overline{CS}$	150 ns	2 CLK1	—
t_{mwdh}	Write data hold after rising edge of $\overline{CS}$ [1]	Note 1	—	—

NOTES:
1. Master will hold output data valid during a write until the Slave device pulls HS low.
2. CLK1 represents the period of the NEURON CHIP input clock (100 ns at 10 MHz).
3. HS high is used as a slave busy flag. If HS is held low, the maximum data transfer rate is 24 CLK1s (2.4 μs @ 10 MHz) per byte. If HS is not used for a flag, caution should be taken to ensure the master does not initiate a data transfer before the slave is ready.
4. In a master read, $\overline{CS}$ pulsing low acts like a handshake to flag the slave that data has been latched in.
5. Parameters were added in order to aid interface design with the NEURON CHIP.

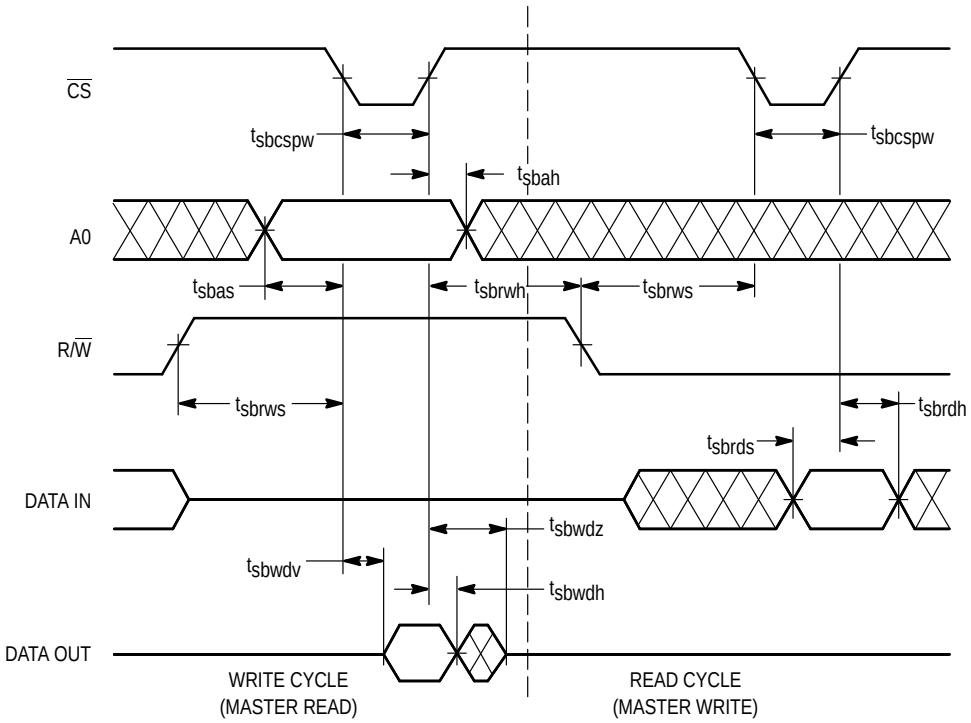
Figure 8. Master Mode Timing



Parameter	Description	Min	Typ	Max
t_{sarws}	R/$\overline{W}$ setup before falling edge of $\overline{CS}$	25 ns	—	—
t_{sarwh}	R/$\overline{W}$ hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sacspw}	$\overline{CS}$ pulse width	45 ns	—	—
t_{sahsh}	HS hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sahsv}	HS valid after rising edge of $\overline{CS}$	—	—	50 ns
t_{sawdd}	Slave A drive of DATA after rising edge of R/$\overline{W}$	25 ns	—	—
t_{sawds}	Write data valid before falling edge of HS	150 ns	2 CLK1	—
t_{sawdh}	Write data valid after rising edge of $\overline{CS}$	150 ns[3]	2 CLK1	—
t_{sardz}	Slave A three–state DATA after falling edge of R/$\overline{W}$	—	—	25 ns
t_{sards}	Read data setup before rising edge of $\overline{CS}$	25 ns	—	—
t_{sardh}	Read data hold after rising edge of $\overline{CS}$	5 ns	—	—

NOTES:
1. CLK1 represents the period of the NEURON CHIP input clock (100 ns at 10 MHz)
2. In slave A mode the HS signal is high, a minimum of 4 CLK1 periods. The typical time HS is high during consecutive data reads or consecutive data writes is also 4 CLK1 periods.
3. If $t_{sarwh} < 150$ ns, then $t_{sawdh} = t_{sarwh}$.

Figure 9. Slave A Mode Timing



Parameter	Description	Min	Typ	Max
t_{sbrws}	R/$\overline{W}$ setup before falling edge of $\overline{CS}$	0 ns	—	—
t_{sbrwh}	R/$\overline{W}$ hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sbcspw}	$\overline{CS}$ pulse width	Note 1	—	—
t_{sbas}	A0 setup to falling edge of $\overline{CS}$	10 ns	—	—
t_{sbah}	A0 hold after rising edge of $\overline{CS}$	0 ns	—	—
t_{sbwdv}	$\overline{CS}$ to write data valid	—	—	50 ns
t_{sbwdh}	Write data hold after rising edge of $\overline{CS}$	0 ns	30 ns	—
t_{sbwdz}	$\overline{CS}$ rising edge to slave B release data bus	—	—	50 ns
t_{sbrds}	Read data setup before rising edge of $\overline{CS}$	25 ns	—	—
t_{sbrdh}	Read data hold after rising edge of $\overline{CS}$	5 ns	—	—

NOTES:

1. The slave B write cycle (master read) $\overline{CS}$ pulse width is directly related to the slave B write data valid parameter and master read set–up parameter. To calculate the write cycle $\overline{CS}$ duration needed for a specific application use:

   $t_{sbcspw} = t_{sbwdv}$ + master's read data setup before rising edge of $\overline{CS}$

   Refer to the Master's Specification Data Book for the master read set–up parameter. The slave read cycle minimum $\overline{CS}$ pulse width = 50 ns.

2. In a slave B write cycle the timing parameters are the same for a control register (HS) write as for a data write.

3. Special Applications: Both the state of $\overline{CS}$ and R/$\overline{W}$ determine a slave B write cycle. If $\overline{CS}$ cannot be used for a data transfer then toggling the R/$\overline{W}$ line can be used with no changes to the hardware. In other words, if $\overline{CS}$ is held low during a slave B write cycle, a positive pulse (low to high to low) on R/$\overline{W}$ can execute a data transfer. The low to high transition on R/$\overline{W}$ causes slave B to drive data with the same timing parameters as t_{sbwdv} (redefined R/$\overline{W}$ to write data valid). Likewise, the falling edge of R/$\overline{W}$ causes slave B to release the data bus with the same timing limits as the $\overline{CS}$ rising edge in t_{sbwdz}. This scenario is only true for a slave B write cycle and is not applicable to a slave B read cycle or any slave A data transitions. This application may be helpful if the master has separate read and write signals but no $\overline{CS}$ signal. Caution must be taken to ensure the bus is free before transfers to avoid bus contention.

Figure 10. Slave B Mode Timing

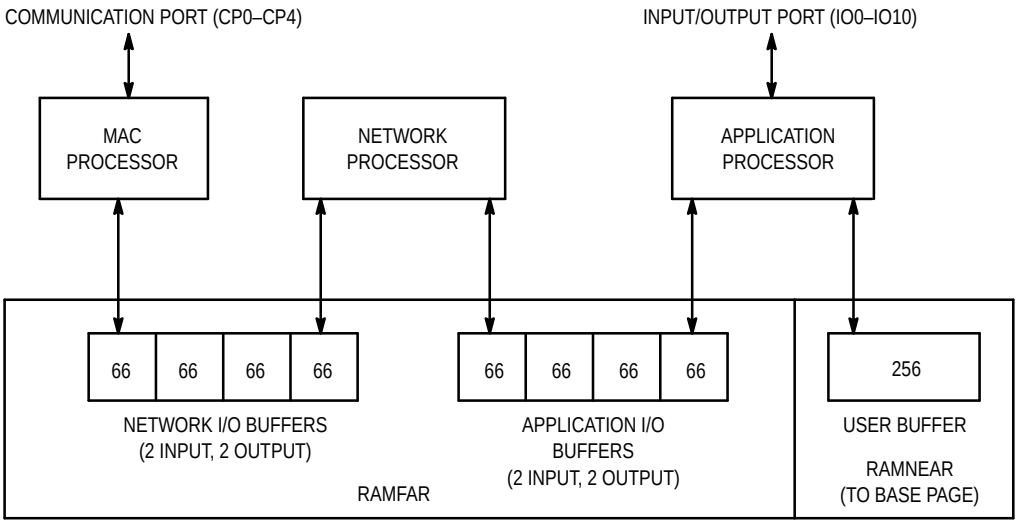


Figure 11. LONBUILDER 2.1 DEVELOPER'S WORKBENCH Default for RAM Allocation (Excluding Priority Buffers)

Following the same calculations as shown above, if the input buffers are assigned the value of 210, then the output buffers could potentially be 42 without external memory.

If the NEURON CHIP requires the size of 210 for the network input buffers to accommodate the largest packet on the channel, the potential size of the output buffers is related to the size of the application input buffers.

In a scenario where the maximum value of 255 bytes is required for output, the source NEURON CHIP could potentially need external memory. Additionally, the destination NEURON CHIP will potentially need external memory, depending on the size of the outgoing messages. Note all the NEURON CHIPS on the channel need to accommodate this packet (all network input buffers must equal 255 on all NEURON CHIPS on the channel).

Pragmas allow the NEURON C programmer to change the size of any of the four buffers, change the count of any of the four buffers, or change the count of the priority buffers. Some examples follow; refer also to the memory management section of the *NEURON C Programmer's Guide* for additional examples:

```
#pragma net_buf_in_size             210
#pragma app_buf_in_size             114
#pragma net_buf_out_size            42
#pragma app_buf_out_size            42
#pragma net_buf_in_count            3
#pragma net_buf_out_priority_count  0
#pragma app_buf_out_priority_count  0
```

The LONBUILDER DEVELOPER'S WORKBENCH provides memory allocation information for a specific application by choosing the *Output Link Summary* build option in the *Options* / *project* pull down menus. The *Build All* option, under the *Project* pull down menu, stores the memory allocation information in a build.log file which can be viewed through the LONBUILDER DEVELOPER'S WORKBENCH editor.

Priority buffers and authentications require additional buffer allocation and are not considered in the previous examples. The LONBUILDER DEVELOPER'S WORKBENCH by default assigns priority buffers; therefore, pragmas are required to release this memory space.

NEURON CHIP–TO–NEURON CHIP INTERFACE

The parallel connection of one NEURON CHIP to another is accomplished by assigning one as the master device and the other as a slave A device. The hardware requirements in this case reduce to a direct, one–to–one, connection of all eleven I/O pins on both sides.

Figure 12 illustrates a typical parallel I/O processing interface routine which would reside on the slave (A) NEURON CHIP. Information is transferred through the I/O port and is never transmitted over the network in this example.

FOREIGN–TO–NEURON PROCESSOR INTERFACE (SLAVE B MODE)

Slave B mode was designed to interface the NEURON CHIP to a foreign processor master.

This section illustrates an M68HC11 (HC11) acting as a master to a NEURON CHIP configured in slave B mode. The NEURON CHIP looks like a peripheral device residing on the HC11's address bus. The hardware interface is shown in Figure 13. ECLK is gated twice for $\overline{CS}$ in order to insure timing compatibility for the HC11's read data hold parameter.

External address decoding logic allows the HC11 to access the NEURON CHIP by using specific addresses (one address for the data register (A0=0) and one for the control register (A0=1)). The decoding circuitry in Figure 13 specifically memory maps the NEURON CHIP to addresses between hex 4000 and hex 7FFF. This section of memory was chosen because it had the simplest decoding circuitry for this specific example's available memory map. In particular, this example software specifies address 4000 for the data register and 4001 for the control register.

```
*************************************************************************
* Example 1
* This NEURON C example program configures the NEURON CHIP in slave A
* mode.
*************************************************************************
IO_0 parallel slave s_bus;
#define DATA_SIZE 255                 //maximum data field allowed
struct parallel_io_interface
{
      unsigned int length;            // length of data field
      unsigned int data[DATA_SIZE];
}piofc
when (io_in_ready(s_bus))             //ready to input data
{
      piofc.length = DATA_SIZE;       //maximum number of bytes
                                        to read
      io_in(s_bus,&piofc);            //get 10 bytes of
                                        incoming data
}
when (io_out_ready(s_bus))            //ready to output data
{
      piofc.length = 10;              //number of bytes to write
      io_out(s_bus,&piofc);           //output 10 bytes from buffer
}
when (...)                            //user defined event
                                        indicating buffer has been filled
 {
      io_out_request(s_bus);          //post the write transfer
 }                                      request
```

Figure 12. Example 1 — NEURON C Program for the NEURON CHIP

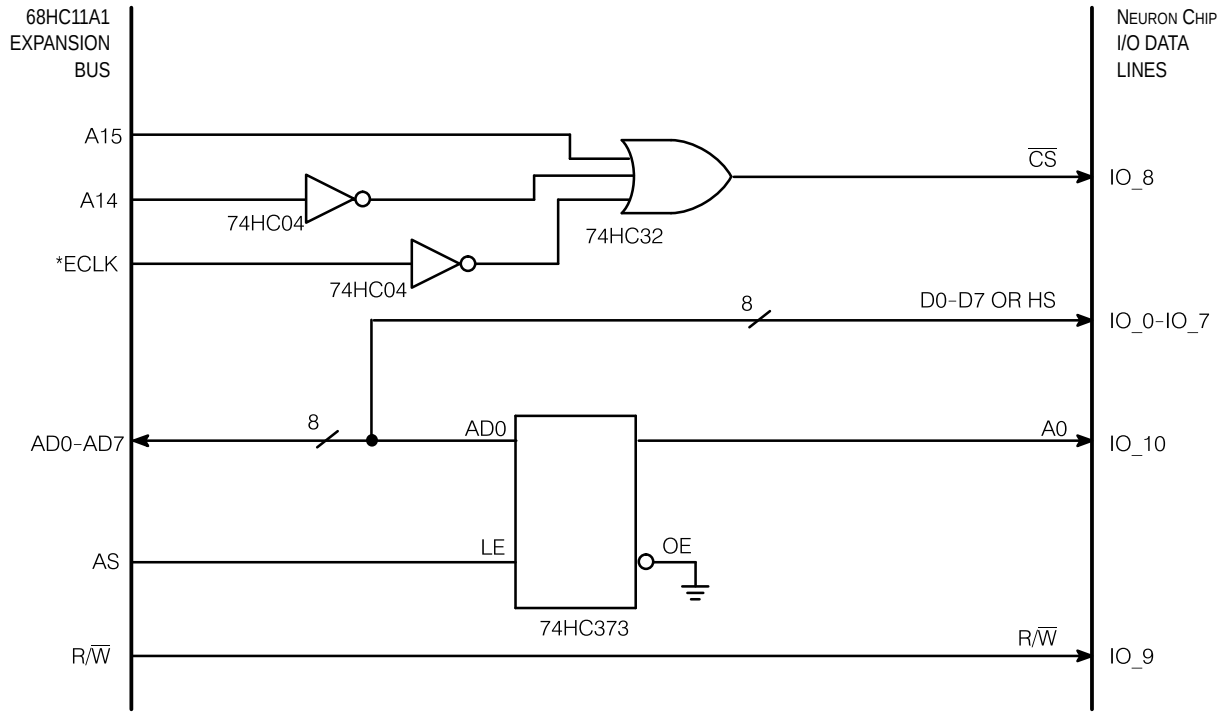


Figure 13. Interfacing the NEURON CHIP (Slave B) to the M68HC11A1
(* ECLK Must be Gated Twice for Interface Timing Requirements)

The HC11's least significant byte of the address bus is multiplexed with its data bus. As A0–A7 is valid the first half of the HC11's E–clock (ECLK) cycle and the data, D0–D7, is valid the last half of the HC11's E–clock cycle, a 74HC373 was used to latch A0. Therefore, A0 controls access to the data register or the control (HS) register.

The M68HC11EVM (EVM) evaluation board was used for this example, as the address and data lines are easily accessible through the existing expanded mode connector. An MC68HC11A1 is supplied with the EVM board. The EVM board provides an 8–MHz external crystal and supports 2–MHz bus operations. The specified bus cable length is 6 inches.

The M68HC11EVBU also brings the address and data lines to the external world through an expanded mode connector and could be used with adjustments to this example's software memory mapping and external decoding circuitry. The M68HC11EVB evaluation board is not suggested as the address and data lines appear as general purpose I/O lines on the connectors and cannot easily support peripheral memory mapped devices.

Another option includes Motorola's M68HC11EVS.

The LONBUILDER DEVELOPER'S WORKBENCH I/O cards (Echelon part number 27800 or 27810; contact an Echelon salesperson) may be used to access the I/O pins from the NEURON emulator unit. The *LONBUILDER DEVELOPER'S WORKBENCH Startup and Hardware Guide* describes how to configure the jumpers for parallel I/O and suggests I/O_9 (R/$\overline{W}$) be tied to the master's I/O buffers circuitry to configure (on the fly) the I/O lines for either input or output mode. As per this example, the slave will need the R/$\overline{W}$ line inverted before connecting to the I/O buffers or alternately the pull up on the I/O buffer schematic can be configured as a pull down. The eight bidirectional I/O data lines are configured on jumpers JP15–JP8. Motorola also provides a direct connect transceiver board (M143204EVK).

The NEURON C code listing (Figure 14) which interfaces the NEURON CHIP with an HC11 can also be used for NEURON CHIP–to–NEURON CHIP interface by declaring the I/O object a slave instead of a slave_b. Due to the transparent nature of the communication protocol at the NEURON C programming level, the NEURON CHIP programmer need not be aware that the interface is to an M68HC11 (or any other foreign processor for that matter) instead of to another NEURON CHIP.

For the M68HC11, an assembly program listing (Figure 15) and also a C program listing (Figure 16) is provided in this application note. Either program may be chosen for the HC11. An IASM11 assembler is provided with the EVM. Additional software tools, including C compilers for the HC11, are available (contact a Motorola salesperson). It is ***not*** recommended to use the bulletin board C compilers designed for the HC11.

The HC11 watchdog time–out option was not implemented in this code but can be included for further reliability purposes.

In the code presented in Figures 14–16, both the NEURON CHIP and the HC11 always have data to send. Therefore, the NULL command which simply passes the token without data transfer is not implemented.

As seen in Table 1, the processing time required by the NEURON CHIP for the HC11 *CMD_XFER* and *length* bytes, which preceeds the actual data transfer, greatly affects the overall performance of the NEURON CHIP.

However, all the HC11 byte reads, per the assembly code provided in Figure 15, are dependent on the HC11 speed.

Therefore, the overall performance is dependent both on the processing time of the NEURON CHIP and the speed of the HC11.

Typical timing using the master HC11 assembly code (from Figure 15) and the slave B NEURON C code (from Figure 14) is given in Table 1.

Table 2 shows the overall performance of the NEURON CHIP slave B interfacing to a foreign processor master. The HC11 code from Figure 15 was utilized. The size of the data buffer varied with the data length.

The NEURON C code, from Figure 14, was also used to acquire the data in Table 2. The buffer size assigned in the NEURON C parallel I/O structure, as specificed by *Max_* (see Figure 14), does not affect the performance of the NEURON CHIP unless the buffer size designated is smaller than the actual length of data being transferred. Therefore, buffer size was adjusted accordingly.

A write/read cycle is defined as starting at the CMD_XFER master write and ending at the following CMD_XFER master write. Therefore, the timing in Table 2 not only considers the actual byte transfer as discussed in Table 1, but also includes time required by the scheduler and processing of the NEURON C event (when) statements.

Table 1. Typical Byte Transfer Timing for NEURON CHIP Slave B–to–HC11 Master Parallel I/O Interface

HC11 Cycle	Byte	Data	Typical Process Time (μs) (per byte)	Typical Time to HS Low (μs) (per byte)
Write	CMD_XFER TO LENGTH	01	730	719
Write	LENGTH TO FIRST DATA	05	207	197
Write	FIRST DATA TO NEXT BYTE	51	37	27.5
Write	ADTL DATA	52–55	24	*
Read	CMD_XFER TO LENGTH	01	18	*
Read	LENGTH TO FIRST BYTE	06	21	*
Read	DATA	0–5	24	*

*HS was low the first loop it was tested, therefore, a faster foreign processor would improve the typical byte transfer time. Typically, HS is low within 3.0 μs during these data transfers.

Table 2 does not reflect network activity; increase in network activity may increase processing time.

An HC11 NULL write occurs when the HC11 passes the token without a data transfer. Altering the HC11 assembly program in Figure 15, a NULL write was performed to pass the token to the NEURON CHIP and the NEURON CHIP transfered 4 bytes of data to the HC11. The time for this write/read cycle was 2.2 ms.

If the application is such that the HC11 would never transfer data to the slave, the `io_in_ready` event statement can be removed and the total master read cycle time reduces to 1.5 ms.

Resynchronizing on the fly can be implemented as a safeguard to verify data integrity. The master initiates the resynchronization (CMD_RESYNC) and the NEURON CHIP replies with an acknowledgement (CMD_ACKSYNC). The resynchronization does not affect data waiting to be transferred.

Utilizing a program similar to the HC11 assembly program shown in Figure 15, the typical time for resynchronization is less than 940 µs. The NEURON C program, from Figure 14, was employed with no modifications (as NEURON CHIP synchronization is not handled by the application).

Table 2. Typical Write/Read Cycle Timing Interfacing the NEURON CHIP in Parallel I/O Slave B Mode to the HC11 Master

HC11 Write (# of Data Bytes)	HC11 Read (# of Data Bytes)	Typical Time for One Write/Read Cycle (ms)
1	4	2.4
5	4	2.5
20	4	2.9
60	4	3.8
200	4	7.2

```
//************************************************************************
// Example 2. Neuron C code slave B mode
// Program for a Neuron Chip in parallel I/O interface with
// an M68HC11. The Neuron Chip is in slave B mode and the HC11 is acting
// as a master. The program enters in an infinite loop of read and
// write cycles.
//************************************************************************
#define MAX_ 10                         //buffer size is ten
IO_0 parallel slave_b p_bus;
unsigned char i=0;                      //counter to fill buffer
unsigned char maxin=10, len_out=4;      //# of bytes for input and output
struct parallel_io
{
      unsigned char len;                //actual number of bytes in buffer
      unsigned char buf[MAX_];          //array to store data
}pio;                                   //name of structure
when (io_in_ready(p_bus))
{
      pio.len = maxin;                  //maximum input length
      io_in(p_bus,&pio);                //read in data
      io_out_request(p_bus);
}
when (io_out_ready(p_bus))
{
      pio.len=len_out;                  //number of bytes to be output
      for (i=0; i<len_out; i++)         //fill buffer with data
            pio.buf[i] = i;
      io_out(p_bus,&pio);               //output data
}
```

Figure 14. Example 2 — NEURON C Code Slave B Bode

```
*****************************************************************
** Example 3. Assembly code for the HC11 master
** Assembly listing of a test program for master/slave B mode where
** master is resident on 68HC11 and slave is resident on the
** NEURON CHIP.
**
** The code below implements the 68HC11 portion,
** receiving any data and sending pre-defined data messages.
** This code is implemented more as a test of the interface
** rather than a test of the protocol.
*****************************************************************

NEURON_ADDR   equ          $4000
DEBUG_ADDR    equ          $0030
HS_MASK       equ          $01
MAXMSGLEN     equ          $20
EOM           equ          $0
TRUE          equ          $1
FALSE         equ          $0
CMD_RESYNC    equ          $5A
CMD_ACKSYNC   equ          $07

** The NEURON CHIP is sitting on the HC11's data bus with a chip
** select address decode set to the following addresses:

data          equ          $4000
control       equ          $4001

     ORG      $0000

     XDEF     token        *boolean representing which side has the token
token         RMB  1       * token

     XDEF     counter      *general purpose counter
counter       RMB  l

     XDEF     msgi         *message in structure
msgi          RMB  34

mi_command    equ  0       *location of command in the msg structure
mi_length     equ  1       *location of data length in the msg structure
mi_data       equ  2       *location of start of data in the msg structure

**     Program Section

     ORG      $E000

*********************************************************
** start of parallel master code
*********************************************************

     XDEF     start_pio
start_pio
     JSR      master_init  *initialize

     XDEF     main_loop
main_loop
     LDAB     token        *load token
     BEQ      no_token     *if token==O, can't write
     JSR      pio_write    *send code message

** Receives any messages

no_token
     JSR      pio_read     *try to read
     BRA      main_loop    *repeat
```

Figure 15. Example 3 — Assembly Code for the HC11 Master (Sheet 1 of 3)

```
*********************************************************************
** wait_hs
** When the NEURON CPU reads or writes to the data port, it
** initially drives the HS line high. The master must wait for HS
** to go low again before the next read from or write to the port.
*********************************************************************

       XDEF   wait_hs
wait_hs
       LDAB   control
       ANDB   #HS_MASK
       BNE    wait_hs
       RTS

*****************************************************************
** master_init
** Standard synchronization with the NEURON CHIP.
** Continue to write the CMD_RESYNC value plus EOM until the
** slave returns the CMD_ACKSYNC value. Return owning token.
*****************************************************************

       XDEF   master_init
master_init
       JSR    wait_hs                 *wait for H.S.
       LDAB   #CMD_RESYNC             *
       STAB   data                    *send the resync value
       JSR    write_eom               *and the EOM.

       JSR    wait_hs                 *wait for the CMD_ACKSYNC.
       LDAB   data                    *read data from the port
       CMPB   #CMD_ACKSYNC
       BEQ    read_complete           *repeat is not sync'ed
       BRA    master_init

*****************************************************************
** pio_read
*****************************************************************

       XDEF   pio_read
pio_read
       LDAB   control                 *load control
       ANDB   #HS_MASK
       BEQ    da
       RTS                            *no data available

** We have data available, handshake line is low

da
       LDY    #msgi                   *set up Y index
       LDAB   data                    *read data from the port
       STAB   0,Y                     *store in message.command
       INY
       BNE    have_data               *go get data, if command!=NULL

** This was token passing message (NULL)

       CLR    0,Y                     *msgi.length=0
       BRA    read_complete

** Since the command was non-zero, get the length byte next.

have_data
       JSR    wait_hs                 *wait for indication of data
       LDAB   data                    *read data from port
       STAB   0,Y                     *msgi.length=ACCB
       INY
       STAB   counter                 *set up the counter
```

Figure 15. Example 3 — Assembly Code for the HC11 Master (Sheet 2 of 3)

```
loop_data
       LDAB   counter                        *load the counter, Z=1, if counter==0
       BEQ    read_complete                  *if counter==0, read is complete

** There is more data to be read from port.

       JSR    wait_hs                        *wait for data available
       LDAB   data                           *read byte from data port
       STAB   0,Y                            *store byte at Y[0]
       INY                                   *increment Y
       DEC    counter                        *decrement counter
       BRA    loop_data

read_complete
       LDAB   #TRUE
       STAB   token
       JSR    wait_hs                        *wait for EOM to be sent
       RTS

*********************************************************************
** pio_write
*********************************************************************

       XDEF   pio_write
pio_write
       LDY    #msgo                   *load pointer to message
       LDAB   0,Y                     *store Y[0] in ACCB
       STAB   data                    *X[0]=Y[0]
       BEQ    write_eom               *if command!=0, then there is a message

** There is data (non-zero command) so send it

is_data
       INY                            *increment to length
       JSR    wait_hs                 *wait for handshake
       LDAB   O,Y                     *load length byte
       STAB   counter                 *store in counter
       STAB   data                    *send the length

**     Send the data

send_next
       LDAB   counter                 *load the counter
       BEQ    write_eom               *if counter==0, then done
       DEC    counter                 *counter--
       INY                            *increment message pointer
       JSR    wait_hs                 *wait for receiver
       LDAB   0,Y                     *load the next byte
       STAB   data                    *send the byte
       BRA    send_next

       XDEF   write_eom
write_eom
       JSR    wait_hs                 *wait before sending EOM
       CLR    data                    *send EOM
       CLR    token                   *token=FALSE
       RTS

** coded outgoing message;

       XDEF   msgo
msgo
       FCB    $01,$05,$51,$52,$53,$54,$55
       END
```

Figure 15. Example 3 — Assembly Code for the HC11 Master (Sheet 3 of 3)

```
/*****************************************************************************/
/* Example 4 – C code for the HC11 master                                    */
/* Example in C–programming language for a 68HC11A1 interfacing              */
/* with a NEURON CHIP. The NEURON CHIP is in parallel I/O                    */
/* slave B mode and the HC11 is acting as a master. The program              */
/* synchronizes the HC11 master and NEURON CHIP slave and then               */
/* enters an infinite loop of read and write cycles                          */
/*****************************************************************************/

#define HS_MASK           0x01        /*mask for lsb of control register     */
#define CMD_RESYNC        0x5A        /*initial command to synchronize NEURON CHIP */
#define CMD_ACKSYNC       0x07        /*synchronization acknowledge from slave */
#define CMD_XFER          0x01        /*command to transfer data             */
#define LENGTH_OUT        0x08        /*length of data transfer from master  */
#define EOM               0X00        /*end of message                       */
#define MAX_              0X09        /*maximum size of data buffer          */
#define DATA_REGISTER     0X4000      /*even adrs accesses data register     */
#define CNTRL_REGISTER    0x4001      /*odd address accesses HS register     */
#define MASTER            1           /*token tracking for master write      */
#define SLAVE             0           /*token tracking for master read       */

unsigned char token;                  /*tracks read and write cycles         */
unsigned char *data, *hs;             /*pointers for data and HS registers   */

struct parallel_io                    /*buffer for data transfers            */
{
unsigned char len;                    /*length of data transferred           */
unsigned char data[MAX_];             /*array to store data                  */
}pio;

/*****************************************************************************/
/* Verify the processors are synchronized before any data is                 */
/* transmitted. The master sends the command to resynchronize until          */
/* the slave acknowledges with CMD_ACKSYNC. The master owns the               */
/* token after resynchronization.                                            */
/*****************************************************************************/

sync_loop()
{
    while (*data != CMD_ACKSYNC) { /*loop until acknowledge received        */
       hndshk();
       *data = CMD_RESYNC         /*send command to resync                  */
       hndshk();
       *data = EOM;               /*send end of message                     */
       hndshk();
    }
    token = MASTER;               /*master owns token after reset           */
}

/*****************************************************************************/
/* Verify the slave is ready for the next byte transaction. Read             */
/* the control register of the slave which accesses the handkshake           */
/* signal(least significant bit of the control register). Mask all           */
/* bits but the handshake bit and verify if the handshake signal has         */
/* gone low.                                                                 */
/*****************************************************************************/

hndshk()                  /*infinite loop until the handshake bit goes low  */
{
     while ((*hs) & HS_MASK);
}
```

Figure 16. Example 4 — C Code for the HC11 Master (Sheet 1 of 3)

```
/*****************************************************************************/
/* Identify the owner of the token to determine if a read or write           */
/* is appropriate. If the master owns the token a write cycle is             */
/* performed. If the slave owns the token a read cycle is initiated.         */
/* Token passing prevents bus contention, as only the owner of the           */
/* token can write to the bus.                                               */
/*****************************************************************************/

main_loop ()
{
   while(1) {                          /*infinite loop of read/write cycles  */
      if (token == MASTER)             /*master owns the token               */
        write();                       /*master writes to the slave          */
      else                             /*slave owns the token                */
         read();                       /*master reads from the slave         */
   }
}

/*****************************************************************************/
/* The master owns the token at the start of this function,                  */
/* therefore, the master can write to the bus. The buffer is filled,         */
/* the command to send data (CMD_XFER) is transmitted, the length            */
/* (number of bytes of data) is transmitted and the data is                  */
/* transmitted one byte at a time. The handshake signal is                   */
/* monitored for low transition before each byte transfer. After             */
/* the data is transmitted, the token is processed.                          */
/*****************************************************************************/

write()
{
    unsigned char send_data;
    make_buffer();                     /*assign length and create data       */
    hndshk();
    *data = CMD_XFER;                  /*command to send data                */
    hndshk();
    *data = pio.len;                   /*send length of data to be transmitted */
    for (send_data=0; send_data<pio.len; send_data++){
       hndshk();
       *data = pio.data[send_data];           /*send one byte of data        */
    }
    pass_token();                              /*process the token            */
}

/*****************************************************************************/
/* Assign the data length. Fill the buffer with data before                  */
/* transmitting. The data is ascii: P,Q,R,S,T,U,V,W.                         */
/*****************************************************************************/

make_buffer()
{
    unsigned char data_out;             /*counter for creating data          */
    pio.len = LENGTH_OUT;               /*length of bytes of data            */
    for(data_out=0; data_out<LENGTH_OUT; data_out++)
    pio.data[data_out]=(data_out+(0x50));
}        /*ascii output*/
```

Figure 16. Example 4 — C Code for the HC11 Master (Sheet 2 of 3)

```
/*****************************************************************************/
/* The slave has the token at the beginning of this function,                */
/* therefore, the master reads from the slave. If the first byte is          */
/* the command to transfer, read the length of data bytes to be              */
/* received, read each byte of data, then transfer the token to the          */
/* master. If the slave has no data to send, assume the command is a         */
/* NULL and simply transfer the token to the master. Always wait             */
/* for the handshake signal to be low before each transaction.               */
/* Note: No error checking is implemented to verify the command              */
/* is a NULL.                                                                */
/*****************************************************************************/

read()

    {
    unsigned char cmd;                /*stores the command from the slave     */
    unsigned char i=0;                /*counter to read in data               */
     hndshk();
     if ((cmd = *data) == CMD_XFER) {  /*slave has data to send               */
         hndshk();
         pio.len = *data;             /*read length of data to be transferred */
         while (pio.len--) {          /*read in each byte of data             */
         hndshk();
         pio.data[i] = *data;         /*put data in a buffer                  */
         ++i;
         }
    }
    pass_token();                     /*pass token to the master              */
}
/*****************************************************************************/
/* Process the token. If the master owns the token, send an end of           */
/* message to the bus and then pass the token to the slave.                  */
/* If the slave owns the token, simply pass the token to the master.         */
/*****************************************************************************/

pass_token()
{
    if (token == MASTER) {            /*master owns the token                 */
       hndshk();
       *data = EOM;                   /*write an end of message               */
       token = SLAVE;                 /*pass the token to the slave           */
    }
    else
       token = MASTER;                /*pass the token to the master          */
}
main ()
{
    data = DATA_REGISTER;             /*data points to the data register      */
    hs = CNTRL_REGISTER;              /*hs points to the control register     */
    sync_loop();                      /*synchronize the processors            */
    main_loop();                      /*infinite loop of read/write cycles    */
}
```

Figure 16. Example 4 — C Code for the HC11 Master (Sheet 3 of 3)

Debugging the Example Programs

The foreign processor must stabilize in approximately 840 milliseconds (10 MHz) after the NEURON CHIP reset to avoid a NEURON CHIP watchdog timeout. If a watchdog timeout does occur the NEURON CHIP simply resets and waits again for the synchronization command.

JP1 and JP2 on the emulator boards should be disconnected. If you are using Echelon's I/O evaluation board, verify that D7 through D0 are jumpered on JP8–JP15 not JP5–JP12.

Prior to the 2.1, release the documentation on configuring the I/O buffer circuitry by tying to I/O9 (R/$\overline{W}$) was for the master NEURON chip only. The slave must have the R/$\overline{W}$ signal inverted.

FOREIGN–TO–NEURON PROCESSOR INTERFACE (SLAVE A MODE)

Slave A mode was designed for a NEURON CHIP–to–NEURON CHIP interface. However, interface of a foreign processor master to a NEURON CHIP in slave A mode can be accomplished several different ways. One interface using an HC11 is conceptually demonstrated in Figure 17.

The slave A write data hold time after the rising edge of $\overline{CS}$ (t_{sawdh}) is 150 ns. Therefore, the HC11 can not easily support the NEURON CHIP as a peripherial device in expanded chip mode.

The HC11 single chip mode does, however, support a parallel IO mode which allows port C to be a full handshake IO port. This option will conceptually support an interface which uses only 11 IO lines and will not require an additional cycle for HS reads. It can be optioned to use HC11 interrupts. This mode may not be appropriate for multiple NEURONS interfacing to a single HC11.

In this example, STRB is configured as an interlocked active high signal for HC11 reads and an interlocked active low signal for HC11 writes. STRA is always configured as an active falling edge signal. The R/$\overline{W}$ signal is generated by any general purpose HC11 output pin.

During an HC11 read, HS directly drives STRA. Initially, STRB is high indicating the HC11 is ready for a data transfer. When the NEURON CHIP has data and is ready to output, the HS line transitions low. The HC11 then pulls STRB low, beginning a low pulse on $\overline{CS}$. When the HC11 pulls STRB high, this indicates data is latched and the NEURON CHIP releases the data and pulls HS high again until the next byte transfer. The HS line does not transition high until after STRB ($\overline{CS}$) is high, therefore, allowing HS to control STRA directly.

During an HC11 write both the HS signal and the STRB signal must be low before STRA falls. Therefore, both processors have activated their ready signals before the transfer is initiated.

The PIOC register should be reconfigured as follows:

HC11 read: CWOM=1, HNDS=1, OIN=0, PLS=0, EGA=0, INVB=1

HC11 write: CWOM=1, HNDS=1, OIN=1, PLS=0, EGA=0, INVB=0

Figure 17 demonstrates the conceptual hardware interface. Interrupts can be generated for each byte transfer. For more information, including timing parameters, refer to document M68HC11RM/AD, *M68HC11 Reference Manual*.

Synchronization

After reset, the master initiates synchronization by sending the command for resynchronization (CMD_RESYNC). If the slave is powered up and ready to begin communication, it sends an acknowledgment back to the master (CMD_ACKSYNC). If the slave is not ready, the master continues to initiate the resynchronization until the slave acknowledges. After synchronization, the master owns the token.

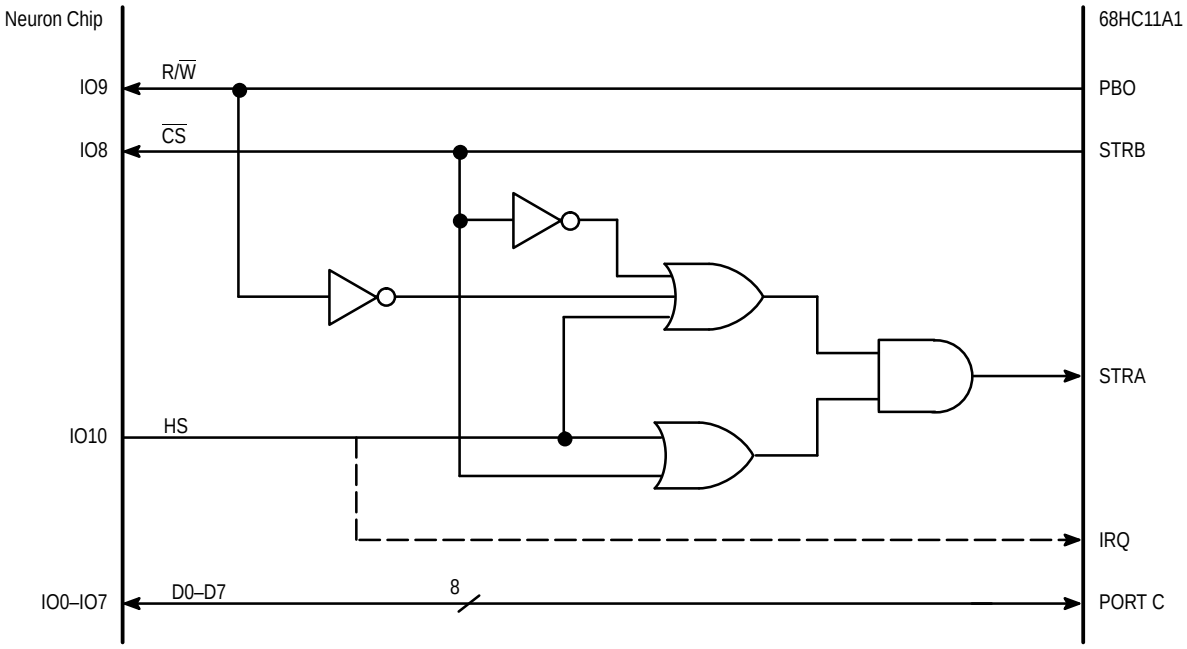


Figure 17. Interfacing the Slave A NEURON CHIP to the M68HC11EVM Master

Master Owns Token

Only the owner of the token can write to the bus. At this point, the master can resynchronize with the slave, write data to the bus, or perform a null write. After the master completes one of these commands, the slave owns the token.

Slave Owns Token

Once the slave owns the token, the slave can acknowledge a resynchronization, or if a resynchronization has not been initiated, the slave can either write data to the bus or perform a null write. After the slave has completed one of these commands, the master owns the token again. The cycle alternates between master writes (master owns the token) and master reads (slave owns the token).

Handshake

The master verifies HS is low before every byte transfer to ensure the slave has finished any internal processing and is ready for the next transaction.

Refer to document NEURONCPG/AD, *NEURON C Programmer's Guide* and document MC143150/D, *MC143150/MC143120 Technical Data* for additional information on NEURON CHIP parallel I/O objects.

GENERATING AN INTERRUPT SIGNAL TO A FOREIGN PROCESSOR MASTER

If the NEURON CHIP holds the token and the foreign processor master does not complete a master read in an appropriate amount of time (approximately 840 milliseconds (10 MHz) after the NEURON CHIP is ready to write to the bus) a NEURON watchdog timeout may occur.

If the master holds the token and does not pass the token to the NEURON CHIP in a timely manner the network could potentially suffer or data could be lost. On the other hand, polling the NEURON CHIP often for data transfers can be inefficient. Therefore, an interrupt strategy is appropriate to release the foreign processor from polling the NEURON CHIP.

Information for configuring and handling interrupts can be found in the foreign processor technical data book or reference manual. In particular, the HC11's I bit in the condition code register (CCR) is the primary enable control for HC11's IRQ, and is set during reset. It should be cleared by the HC11 with a CLI instruction after parallel I/O synchronization is complete. The I bit is automatically set during entry to the HC11's service routine and can be automatically cleared with each return from the service routine using the RTI command. Refer to the HC11's reference manual for more details. The HC11's IRQ interrupt vector is located at address FFF2.

Utilizing the LONTALK Network to Generate an Interrupt Signal to a Foreign Processor for Parallel I/O Transfers

Figure 18 demonstrates how two NEURON CHIPs can be utilized to generate an interrupt signal to a foreign processor master.

When NEURON CHIP 1 (slave) is ready to transfer information, a signal is sent to NEURON CHIP 2 (interrupt generator) over the network. This can be done with the same message (i.e., network variable) that brings the information into NEURON CHIP 1, or it can be a separate network signal.

NEURON CHIP 2 utilizes an I/O pin to generate the interrupt signal to the master.

The approach in Figure 18 can be employed with a master/slave A interface or a master/slave B interface, and either version of the NEURON CHIP.

The interrupt service routine would begin with the foreign processor performing a data write or a NULL write, if the foreign processor has no data. The master would then perform a read cycle to obtain the data from the slave. At the conclusion of the master read, the master owns the token again. In order to avoid watchdog timeouts, the interrupt service routine concludes with the master owning the token.

Utilizing Memory Mapped I/O to Generate an Interrupt Signal to the Foreign Processor Master

Utilizing the schematic shown in Figure 19, an MC143150 NEURON CHIP can utilize an interrupt approach similar to memory mapped I/O to generate an interrupt signal when the NEURON CHIP has data to transfer, in either slave A or slave B mode. The MC143120 does not support an expanded address and data bus and cannot be used to generate a memory mapped interrupt.

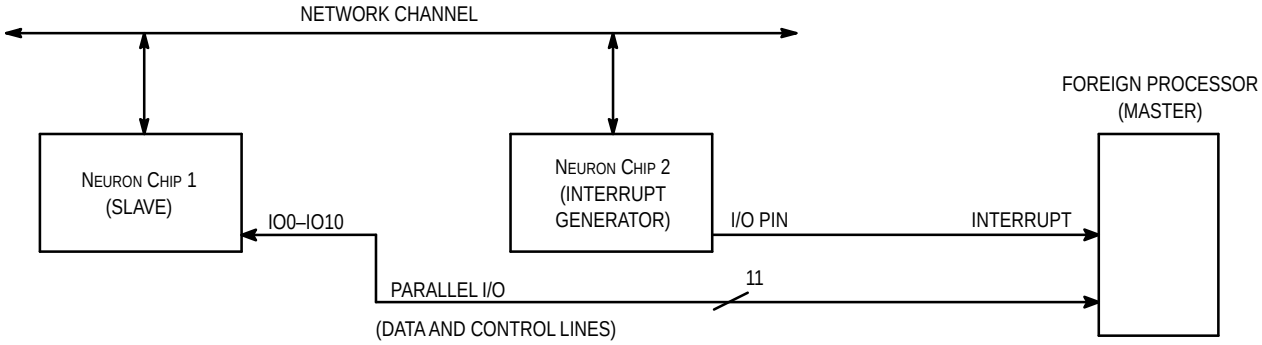


Figure 18. LONTALK Interrupt Option for the Parallel I/O Interface Between the NEURON CHIP Slave and a Foreign Processor Master

In the partial NEURON C program shown in Figure 20, INTRPT_ADRS is the interrupt generating address to be determined by the application. Accessing this address will cause the IRQ pin to pulse low for a time period of 100 ns at 10 MHz.

The IRQ pin should be configured as a negative–going edge–detect input (set IRQE=1 in the OPTION control register within 64 clock cycles after reset). This configuration allows only a single interrupt source to use the IRQ pin. As demonstrated in Figure 19, the decoded interrupt address is gated with the Eclock and generates an input to the IRQ pin.

The interrupt service routine would begin with the master owning the token. When an interrupt occurs, the master would pass the token to the slave, which would enable the slave to write data to the bus. After the slave write command is completed, the interrupt routine concludes with the master owning the token.

Utilizing a State Machine to Monitor the NEURON CHIP for Data Transfers

A state machine which passes the token in a ping–pong fashion between itself and the NEURON CHIP can be designed using a programmable logic device. When the state machine owns the token, a pass token command is emulated, as described in Figure 6, to pass the token to the NEURON CHIP.

When the NEURON CHIP has the token and is ready to access the bus, the state machine monitors the least significant bit of the data bus. A zero on the least significant bit indicates a NULL (the NEURON CHIP has no data) and a logic one indicates a CMD_XFER.

If the NEURON CHIP has data, the state machine generates an interrupt signal to the foreign processor. The foreign processor's interrupt handler would release the state machine and accept the remaining bytes (*length* and data) from the NEURON CHIP.

At any time, the foreign processor could also release the state machine and transfer data to the NEURON CHIP.

The slave can be monitored between every byte transfer to ensure a low on the HS output. The state machine can emulate a master, slave A or slave B mode provided the timing parameters can be met by the state machine. See Figures 8, 9, and 10 for detailed timing information. Either version of the NEURON CHIP (MC143120 or MC143150) can be utilized.

Slave A Generates an Interrupt Signal to a Foreign Processor Master for Byte Transfers

A NEURON CHIP in slave A mode supports a hardwired HS line, which transitions from high to low when the NEURON CHIP is ready for the next byte transaction. The low state or transition of this HS line can be used to generate an interrupt signal to the foreign processor master, indicating the NEURON CHIP is available for the next byte transfer. This application is useful to free the foreign processor between byte transfers. The interrupt service routine would handle each byte transfer accordingly.

To avoid interrupts between each byte, the interrupt handler would need to release the slave A HS line from the interrupt input signal. As seen in Table 1, some of the byte transfers are more timely than others; therefore, releasing the HS line could be advantageous. At the end of the interrupt handler routine, the HS line can again be enabled to interrupt the master.

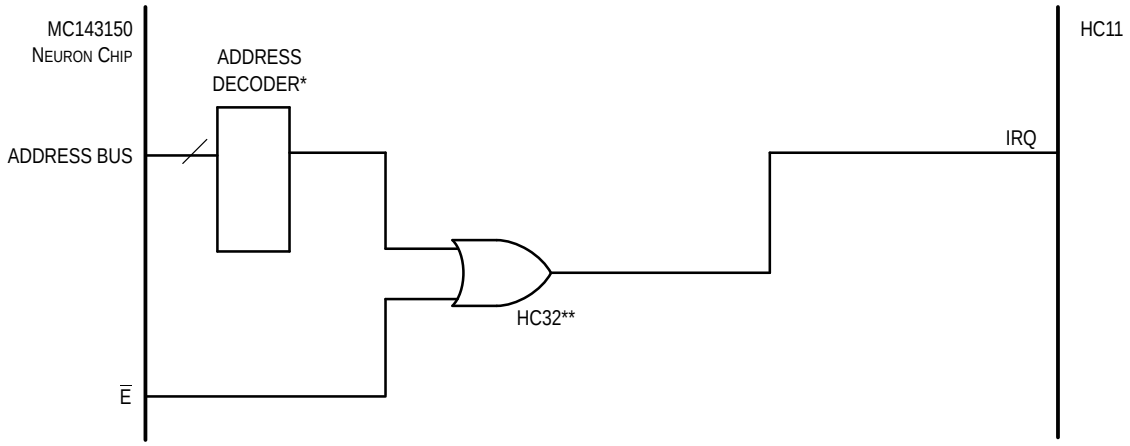
Since the HS line is identical in the MC143120 and MC143150, either chip can be used if memory requirements permit.

CONCLUSION

Use of the parallel I/O interface has tremendous value for coprocessors, gateways, and routers.

Various considerations come into play when interfacing to microprocessors not covered in this application note. The concepts are the same for all foreign processors; however, the timing issues should be closely considered.

The timing diagrams, in Figures 8, 9, and 10 should be useful in determining the specific hardware needed for each application.



* The address decoder can be an HC138 depending on application memory map.
** Either an OR gate or a NOR gate can be used as IRQ is configured as an edge detect interrupt in this example.

Figure 19. NEURON CHIP Generates Interrupt to HC11 Master

```
*******************************************************************************
* Example 5
* Conceptual partial program using the slave B NEURON CHIP address bus
* to generate an interrupt to the HC11 master. Note that slave A
* can also be utilized.
*
*******************************************************************************
#define MAX_ 10                         //buffer size is 10
#define INTRPT_ADRS ????                //interrupt addrs; application dependent
IO_0 parallel slave_b p_bus;

unsigned int * intrpt ;                 //interrupt from NEURON--IRQ
unsigned char maxin=10, len_out=7;      //# of bytes buffers

struct parallel_io                      //structure for transmitting data
{
      unsigned char len;
      unsigned char buf[MAX_];
}pio;

when (reset)
{
      intrpt = (int*)INTRPT_ADRS;       //assign interrupt ptr to addrs
 }

when (...)                              //NEURON CHIP has recvd data to transmit
{
      intrpt = 1;                       //enable IRQ
}

when(io_in_ready(p_bus))                //NEURON CHIP must recv token from master
{
 pio.len = maxin;
 io_in(p_bus,&pio);
 io_out_request(p_bus);                 //NEURON CHIP has token, request to output
}

when (io_out_ready(p_bus))
{
      .
      .                                 //store data into pio structure
      .
 pio.len = len_out;
 io_out(p_bus,&pio);                    //output the data
}
```

Figure 20. Example 5 — NEURON C Code for a NEURON CHIP Slave B to Generate an Interrupt Signal to the Master