

# INDEX of VHDL CODE

## C.1 VHDL Code for RS232 Serial Connection / FIFO Queues

### 1.1 Serial Top Package

**Entity name :** serialtop.vhd

**Description :** It contains all the components used to build the serial port.  
CODE COMPILED – no known bugs

#### 1.1.1 Input Fifo Package

**Entity name :** fifoin.vhd

**Description :** Implementation of a input FIFO.  
CODE COMPILED – no known bugs

##### 1.1.1.1 Receive FIFO

**Entity name :** rxfifo.vhd

**Description :** Implementation of a input FIFO.  
CODE COMPILED – no known bugs

##### 1.1.1.2 RS232 Serial Input

**Entity name :** rs\_232\_in.vhd

**Description :** Implementation of the receive portion of the serial data  
CODE COMPILED – no known bugs

##### 1.1.1.2.1 Shift Register

**Entity name :** serialshiftright.vhd

**Description :** Converts serial data to parallel data  
CODE COMPILED – no known bugs

#### 1.1.2 Output Fifo Package

**Entity name :** fifoout.vhd

**Description :** Implementation of a output FIFO.  
CODE COMPILED – no known bugs

#### 1.1.2.1 Transmit FIFO

**Entity name :** txfifo.vhd

**Description :** Implementation of a output FIFO.

CODE COMPILED – no known bugs

#### 1.1.2.2 RS 232 Serial Output

**Entity name :** rs\_232\_out.vhd

**Description :** Implementation of the send portion of the serial data

CODE COMPILED – no known bugs

##### 1.1.2.2.1 Shift Register

**Entity name :** myshiftout.vhd

**Description :** Converts parallel data to serial

CODE COMPILED – no known bugs

##### 1.1.2.2.2 Synchronizer

**Entity name :** sync.vhd

**Description :** synchronize a signal with the clock

CODE COMPILED – no known bugs

## C.2 VHDL Code for LPM\_RAM

**Entity name :** our\_ram.vhd

**Description :** Instantiates LPM RAM and provides internal addressing so that the control is not concerned with it.

CODE COMPILED – no known bugs.

## C.3 VHDL Code for BusMaster

Entity Name : **busmaster.vhd**

**Description :** This component drives the information coming from the RAM that is storing the training data and the random numbers generated into the neural network datapath. It also sends the address of the neuron and the address of the selected weight of that specific neuron that is going to be replaced in order to run the training algorithm.

CODE COMPILED and DEBUGGED- no known bugs.

## C.4 VHDL Code for Neural Network Datapath

### 4.1 Neural Network Package

**Entity name :** nn\_pack.vhd

**Description :** It contains all the components used to build the Neural Network Datapath, the neuron prototypes and the random number generator  
CODE COMPILED – no known bugs

### 4.1 Neuron Prototypes

#### 4.1.1 Neuron Prototype I: Booth Multiplier Implementation

**Entity name :** Neuron.vhd

**Description :** Implementation of a Hebbian neuron using the architecture depicted on Figure PAGE#. The multiplier uses the booth algorithm.  
**CODE COMPILED and DEBUGGED-** no known bugs.

##### 4.1.1.1 Component Datapath

##### 4.1.1.2 Component Control Unit Implementation

#### 4.1.2 Neuron Prototype II: Carry Save Adder

**Entity name :** Neuron2.vhd

**Description :** Implementation of a Hebbian neuron using the architecture depicted on Figure PAGE#. A carry-save-adder multiplier was implemented.

**CODE COMPILED and DEBUGGED-**no known bugs.

##### 4.1.2.1 Component Datapath

##### 4.1.2.2 Component Control Unit

#### 4.1.3 Neuron Prototype III: LPM\_mult megafunction Implementation

**Entity name :** Neuron3.vhd

**Description :** Implementation of a Hebbian neuron using the architecture depicted on Figure PAGE#. The lpm\_mult megafunction was used as multiplier component.

**CODE COMPILED and DEBUGGED-** no known bugs

##### 4.1.3.1 Component Datapath

##### 4.1.3.2 Component Control Unit

#### 4.1.4 Neuron Prototype IV: Behavioral Implementation

**Entity name :** Neuron4a.vhd

**Description :** Behavioral implementation of a hebbian neuron containing two inputs and two weights.

**CODE COMPILED and DEBUGGED-** no known bugs

## 4.2 Neural Network Datapath

**Entity name:** nn\_dp\_2n.vhd

**Description :** This prototype uses separated input and output buses to drive the information. Its design is very flexible and scalable, however it was built using the design from the old datapath that used a bidirectional bus. Additional signals were also included.

## C.5 VHDL Code for Pseudo Random Number Generator

**Entity name :** prng.vhd

**Description :** Component that generates random numbers continuously.  
CODE COMPILED and DEBUGGED- no known bugs.

## C.6 VHDL Code for Error Calculator

**Entity name:** error\_calculator.vhd

**Description :** Keeps a count of how many times the network misclassifies input data and uses this to tell the control unit whether a new weight should be kept or discarded.

## C.7 VHDL Code for Control Unit

7.1 Package: Control Unit

7.2 Component: Control Unit

**Entity name:** control\_unit2.vhd

**Description:** Used to control the NN and trainer. Wires together the trainer control path, the memory unit, the bus master, and the error calculator to

completely control the activity of the  
NN.

7.3 Component: Trainer Control Path

**Entity name:** trainer\_control.vhd

**Description:** This is the state machine that implements the training algorithm and coordinates the behaviour of the entire system.

## C.8 VHDL Package for Entire Device

8.1 Neural Network Trainer

8.2 Top Level Package of Hardware Device



## Appendix C.1 RS232-Serial Communication

### Description of RS232 Code

The main body of the RS232 code was taken from the RS-232 Serial Port1999 winter app note, written by Tim Bensler & Eric Chan. However, the code did not work initially, thus some modifications to the code was needed. Ben Talbot fixed the input line and our group fixed the output line.

The RS232 port consists of 3 main components: Input Line (RS232IN), Output Line (RS232OUT), and the User Interface.

**Input Line:** The Input Line consists of one input data line, an input clock, an output byte and a valid byte. Data is sampled on the line on every rising edge of the clock cycle, each bit of data sampled 16 times. 16 samples are taken to ensure that the bit is not noise on the line. See the test waveforms to see this in action. Once valid data has been accepted, the data valid bit will assert high for one clock cycle and the data will be held on the output data byte. The data byte will be held there until new data is accepted.

The Input Line was implemented in package rs\_232\_in.

**Output Line:** The Output Line consists of several more signals than the input. It has the standard input data byte, load signal, output data line, and input clock. On top of that, it has an enable signal, a reset, and a ready bit. The data byte is a buffer to hold the next byte to be sent out. The load signal, when asserted, sends the current value in the data buffer to the serial data line. Each bit is sent out on every rising edge of the clock. The enable signal needs to be held high to enable the output data line. The reset clears the data buffer when asserted, and the ready bit goes to high when the serial data line is not busy sending data.

The Output Line was implemented in package rs\_232\_out.

**User Interface:** To ensure that the RS-232 code was working properly, incoming data needed to be viewed and output data needed to be configurable. The LED display built on the UPI board served the purpose of displaying incoming data bytes and the output data was configured using the on board dip switches.

The User Interface was implemented in the top-level entity serial2led.



### C.1.1 Top Level Entity of RS232 Port PACKAGE

```
-----
-- Serialport top level entity
-- Author: Tim Li
-- Purpose: Wraps the Fifo in and Fifo out buffers
--
-- last Modified 02/13/03
--           comment: made the 24 bit receive fifo, it doesn't work
--           stupid write process isn't functioning properly
-- 02/14/03
--           comment: got latch to work, 24 bit fifo seems to work...
-- 02/15/03
--           finally finished!!
-----

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;                                -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.led_pkg.all;
use work.fifo_in_pkg.all;
use work.fifo_out_pkg.all;

package serialtop_pkg is

component serialtop is
    generic (
        InDivisor: positive := 82;           -- 7 115200bps --82 9600bps;
        OutDivisor: positive := 1311        -- 109 115200bps --1311 9600bps
    );
    port(
        clock, reset: in std_logic;
        -----
        tx_fifo_write: in std_logic;         -- Set high to write data into fifo
        tx_fifo_input: in std_logic_vector(15 downto 0); -- 16 bit data to send to fifo
        cmpserialout : out std_logic;        -- Sends the 16 bit data serially to computer
        tx_fifo_full: out std_logic;         -- flags high when the fifo is full
        -----
        rx_fifo_output : out std_logic_vector(31 downto 0); -- outputs
        rx_fifo_read_ack: in std_logic;     -- Set high ack that you read from the fifo
        cmpserialin : in std_logic;         -- Serial data from the computer
        rx_full : out std_logic;            -- O/p to computer fifo is full & stop sending
        rx_empty : out std_logic;           -- Flags high indicate that the fifo is empty.
    );
end component serialtop;

end package serialtop_pkg;
```

### C.1.1 SerialTop Component

-- The Serial Top component used in the above Package.

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm; -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.led_pkg.all;
use work.fifo_in_pkg.all;
use work.fifo_out_pkg.all;

entity serialtop is
    generic (
        InDivisor: positive := 82; --7 115200bps --82 9600bps;
        OutDivisor: positive := 1311 --109 115200bps --1311 9600bps
    );
    port(
        clock, reset: in std_logic;
        -----
        tx_fifo_write: in std_logic;
        tx_fifo_input: in std_logic_vector(15 downto 0);
        cmpserialout : out std_logic;
        tx_fifo_full: out std_logic;
        -----
        rx_fifo_output : out std_logic_vector(31 downto 0);
        rx_fifo_read_ack: in std_logic;
        cmpserialin : in std_logic;
        rx_full : out std_logic;
        rx_empty : out std_logic
    );
end serialtop;

architecture behaviour of serialtop is
    signal const_one : std_logic; -- Constant '1'
    signal const_zero : std_logic; -- Constant '0'
    -----
begin
    const_one <= '1';
    const_zero <= '0';

    fifoout_inst : fifoout
        generic map (
            InDivisor => InDivisor,
            OutDivisor => OutDivisor
        )
        port map(
            clock => clock,
            reset => reset,
            inputfifo => tx_fifo_input,
            writefifo => tx_fifo_write,
            outputserial => cmpserialout,
            fifofull => tx_fifo_full
        );
end;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
fifoin_inst : fifoin
  generic map
    (
      InDivisor => InDivisor,
      OutDivisor => OutDivisor
    )
  port map
    (
      clock => clock,
      reset => reset,
      outputfifo => rx_fifo_output,
      read_ack => rx_fifo_read_ack,
      inputserial => cmpserialin,
      rx_fifofull => rx_full,
      rx_fifoempty => rx_empty
    );
end behaviour;
```

### C.1.1.1 Input FIFO Package

```
-----
-- Author: Andrew Ling
-- Last Modified: March 14, 2002
-- Purpose: Connects the RS232 input port to the rx FIFO buffer
--          Accepts data from rx data line and stores in into the FIFO where
--          it can be take from the control unit
--          Takes in four bytes and concatenates bytes to create a 32-bit
--          word. The 32-bit word is then added to FIFO
-----

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;                      -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.rs_232_in_pkg.all;
use work.sync_pkg.all;

package fifoin_pkg is

component fifoin
    generic (
        InDivisor: positive := 82;  --7 115200bps --82 9600bps;
        OutDivisor: positive := 1311 --109 115200bps --1311 9600bps
    );
    port(
        clock, reset : in std_logic; -- active low reset
        outputfifo : out std_logic_vector(31 downto 0); -- the o/p of the fifo, next word
                                                         -- to be read
        read_ack : in std_logic; -- ack signal that word on outputfifo has been read, get
                                  -- the next word in fifo once read_ack goes high, updates
                                  -- outputfifo on the next rising edge of clock
        inputserial : in std_logic; -- input data line from PC to FPGA
        rx_fifofull : out std_logic; -- indicates that the fifo is full
        rx_fifoempty : out std_logic -- indicates if the fifo is empty
    );
end component fifoin;

end package fifoin_pkg;
```

### C.1.1.1 Input FIFO Component

-- The RX FIFO component used in the above Package.

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm; -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.rs_232_in_pkg.all;
use work.sync_pkg.all;

entity fifoin is
    generic (
        InDivisor: positive := 82; --7 115200bps --82 9600bps;
        OutDivisor: positive := 1311 --109 115200bps --1311 9600bps
    );
    port
    (
        clock, reset : in std_logic; -- active low reset
        outputfifo : out std_logic_vector(31 downto 0); -- the o/p of fifo, next word read
        read_ack : in std_logic; -- ack. hat word on o/p o has been read, get next word
        -- in fifo once read_ack goes high, updates o/p fifo on the
        -- next rising edge of clock
        inputserial : in std_logic; -- input data line from PC to FPGA
        rx_fifofull : out std_logic; -- indicates that the fifo is full
        rx_fifoempty : out std_logic -- indicates if the fifo is empty
    );
end fifoin;

architecture behaviour of fifoin is

    COMPONENT rxfifo IS
        PORT
        (
            data : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
            wrreq : IN STD_LOGIC ;
            rdreq : IN STD_LOGIC ;
            clock : IN STD_LOGIC ;
            sclr : IN STD_LOGIC ;
            q : OUT STD_LOGIC_VECTOR (31 DOWNT0 0);
            full : OUT STD_LOGIC ;
            empty : OUT STD_LOGIC
        );
    END COMPONENT rxfifo;

    --component debounce IS
    --    PORT(pb, clock_100Hz : IN STD_LOGIC;
    --        pb_debounced : OUT STD_LOGIC);
    --END component debounce;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component clk_div IS
  PORT
  (
    clock_25Mhz      : IN  STD_LOGIC;
    clock_1MHz       : OUT  STD_LOGIC;
    clock_100KHz     : OUT  STD_LOGIC;
    clock_10KHz      : OUT  STD_LOGIC;
    clock_1KHz       : OUT  STD_LOGIC;
    clock_100Hz      : OUT  STD_LOGIC;
    clock_10Hz       : OUT  STD_LOGIC;
    clock_1Hz        : OUT  STD_LOGIC
  );
end component clk_div;

component clkdiv is
  -- default to .75H
  -- 25.175 MHz/ (2**24) /2
  generic (Divisor: positive :=33554432); -- clock division rate
  port
  (
    fast_clock :    in STD_LOGIC;
    reset      :    in STD_LOGIC;
    slow_clock  :    buffer STD_LOGIC
  );
end component clkdiv;

-----
signal inputbyte :      std_logic_vector (7 downto 0);
signal rx_input_reg :   std_logic_vector (31 downto 0);
signal store_new_data : std_logic;
signal serial_input_clock : std_logic;      -- Clock for Receiveing Data
signal inputvalid :     std_logic;
signal send_data :      std_logic;
signal write_latch :     std_logic;
signal fifo_full :       std_logic;
-----

signal const_one :      std_logic;
signal const_zero:      std_logic;
-----

type state_type is
(load_first_byte,load_second_byte,load_third_byte,load_fourth_byte,send_to_fifo);
signal fifo_state : state_type := load_first_byte;

begin
const_one <= '1';
const_zero <='0';

serialoutclkdiv : clkdiv
  generic map( Divisor => InDivisor)
  port map(
    fast_clock => clock,
    reset => reset,
    slow_clock => serial_input_clock
  );
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

RS232IN: RS\_232\_In

Port map

```
(
    clock      =>    serial_input_clock,
    shiftin    =>    inputserial,
    q          =>    inputbyte,
    data_valid  =>    Inputvalid
);
```

rx\_fifo\_inst : rxfifo PORT MAP

```
(
    data      =>    rx_input_reg,
    wrreq     =>    store_new_data,
    rdreq     =>    send_data,
    clock     =>    clock,
    sclr      =>    reset,
    q         =>    outputfifo,
    full      =>    fifo_full,
    empty     =>    rx_fifoempty
);
```

rx\_fifo\_full <= fifo\_full;

-----  
--sync\_inst: sync

--port map (longpulse=>read,

-- shortpulse=>store\_new\_data,

-- clock =>clock

-- );

load\_to\_fifo : process (clock, reset)

--load\_to\_fifo : process (serial\_input\_clock, reset, send\_data)

begin

if (reset = '1') then

    fifo\_state <= load\_first\_byte;

elsif rising\_edge(clock) then

case fifo\_state is

    when load\_first\_byte => -- attempt to load first byte from receive buffer

        if write\_latch = '1' then

            rx\_input\_reg(7 downto 0) <= inputbyte;

            fifo\_state <= load\_second\_byte;

        end if;

        store\_new\_data <= '0';

    when load\_second\_byte => -- attempt to load 2nd byte from receive buffer

        if write\_latch = '1' then

            rx\_input\_reg(15 downto 8) <= inputbyte;

            fifo\_state <= load\_third\_byte;

        end if;

    when load\_third\_byte => -- attempt to load 3rd byte from receive buffer

        if write\_latch = '1' then

            rx\_input\_reg(23 downto 16) <= inputbyte;

            fifo\_state <= load\_fourth\_byte;

        end if;

    when load\_fourth\_byte => -- attempt to load 3rd byte from receive buffer

        if write\_latch = '1' then

            rx\_input\_reg(31 downto 24) <= inputbyte;

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
        fifo_state <= send_to_fifo;
    end if;
    when send_to_fifo => -- attempt to load 24-bit word to the fifo
        if (fifo_full = '0') then
            store_new_data <= '1';
            fifo_state <= load_first_byte;
        end if;
    when others =>
        fifo_state <= load_first_byte;
end case;

end if;

end process load_to_fifo;

write_process : process(clock)
variable output : integer range 0 to 1 := 0;
variable but_output : integer range 0 to 1 := 0;
begin
    if falling_edge(clock) then
        if (reset = '1') then
            output := 0;
            write_latch <= '0';
        elsif (inputvalid = '1') and (output = 0) then --write_latch
            output := 1;
            write_latch <= '1'; --store_new_data
        elsif (inputvalid = '0') and (output = 1) then
            output := 0;
            write_latch <= '0';
        else
            write_latch <= '0';
        end if;

        if (reset = '1') then
            but_output := 0;
            send_data <= '0';
        elsif (read_ack = '1') and (but_output = 0) then
            but_output := 1;
            send_data <= '1';
        elsif (read_ack = '0') and (but_output = 1) then
            but_output := 0;
            send_data <= '0';
        else
            send_data <= '0';
        end if;
    end if;

end if;

end process write_process;

end behaviour;
```



### C.1.1.1.1 Receive FIFO

-----  
-- Created by Altera's Megafunction Wizard  
-- this is the RX Fifo  
-----

LIBRARY ieee;  
USE ieee.std\_logic\_1164.all;

```
ENTITY rxfifo IS
  PORT
  (
    data      : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
    wrreq     : IN STD_LOGIC ;
    rdreq     : IN STD_LOGIC ;
    clock     : IN STD_LOGIC ;
    sclr      : IN STD_LOGIC ;
    q         : OUT STD_LOGIC_VECTOR (31 DOWNT0 0);
    full      : OUT STD_LOGIC ;
    empty     : OUT STD_LOGIC
  );
END rxfifo;
```

ARCHITECTURE SYN OF rxfifo IS

```
  SIGNAL sub_wire0    : STD_LOGIC ;
  SIGNAL sub_wire1    : STD_LOGIC_VECTOR (31 DOWNT0 0);
  SIGNAL sub_wire2    : STD_LOGIC ;
```

```
  COMPONENT lpm_fifo
  GENERIC (
    lpm_width          : NATURAL;
    lpm_numwords       : NATURAL;
    lpm_widthu         : NATURAL;
    lpm_showahead      : STRING;
    lpm_hint           : STRING
  );
  PORT (
    rdreq  : IN STD_LOGIC ;
    sclr   : IN STD_LOGIC ;
    empty  : OUT STD_LOGIC ;
    clock  : IN STD_LOGIC ;
    q      : OUT STD_LOGIC_VECTOR (31 DOWNT0 0);
    wrreq  : IN STD_LOGIC ;
    data   : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
    full   : OUT STD_LOGIC
  );
END COMPONENT;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
BEGIN
    empty    <= sub_wire0;
    q        <= sub_wire1(31 DOWNT0 0);
    full     <= sub_wire2;

    lpm_fifo_component : lpm_fifo
    GENERIC MAP (
        LPM_WIDTH => 32,
        LPM_NUMWORDS => 4,--16,
        LPM_WIDTHU => 2,--4,
        LPM_SHOWAHEAD => "ON",
        LPM_HINT => "USE_EAB=ON,MAXIMIZE_SPEED=5"
    )
    PORT MAP (
        rdreq => rdreq,
        sclr => sclr,
        clock => clock,
        wrreq => wrreq,
        data => data,
        empty => sub_wire0,
        q => sub_wire1,
        full => sub_wire2
    );

END SYN;
```

### C.1.1.1.2 RS\_232\_In Package

-----

- This package may be used to accept data from an RS\_232 Port. The clock to this package
- should be at a rate 16 times that of the RS\_232 port. ie. the input signal will be sampled at 16
- times the baud rate for more accurate detection of the start bit for asynchronous
- communication. Data must be read off immediately when the data signal valid goes high there
- is no method provided here to delay the incoming data stream.
- 
- Adapted from the 1998\_w application note by Tim Bensler & Eric Chan
- I recommend a 0.5 to 1ms delay between characters, but it is not essential

-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;
library work;

PACKAGE RS_232_In_pkg IS

component RS_232_In
port(
        clock      : in STD_LOGIC ;
        shiftin    : in STD_LOGIC ;
        q           : out STD_LOGIC_VECTOR (7 DOWNTO 0);
        data_valid : out std_logic
    );

end component RS_232_In;

END RS_232_In_pkg;
```

### C.1.1.1.2 RS\_232\_In Component

-- The RS\_232\_In component used in the above Package.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity RS_232_In is
port(
    clock      : in STD_LOGIC ; --this clock should be SAMPLE times greater than
    baud rate  : in STD_LOGIC ; -- RS_232 data stream in
    shiftin    : in STD_LOGIC ; -- RS_232 data stream in
    q          : out STD_LOGIC_VECTOR (7 DOWNTO 0); -- 8 bit word
    data_valid : out std_logic -- complete word has been received
);
end RS_232_In;

architecture mixed of RS_232_In is

component serialShiftRight IS
    PORT
    (
        clock      : IN STD_LOGIC ;
        enable     : IN STD_LOGIC ;
        shiftin    : IN STD_LOGIC ;
        aclr       : IN STD_LOGIC ;
        q          : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
    );
end component serialShiftRight;

constant SAMPLE:natural:= 16; -- how many times faster than baud rate we are sampling at
constant clear_time: natural :=4; -- how long to hold the clear low to wipe the shift register
constant counterOffset: natural:= 8; --counter offset to take sample in the middle of pulse
constant counterMax: natural:= 164; -- 10bits * SAMPLE + SAMPLE/2 (SAMPLE/2) to sample
middle of pulse

signal qout: std_logic_vector(9 downto 0); -- temporary variable used by the shift register
signal delayValid: std_logic; -- delays the valid signal until the character has stabilized on the
bus
signal clearshift : std_logic; -- used to clear the shift register when a startbit is found
signal enableSignal : std_logic; -- enables the shift register to process another bit

begin
    -- Converts the incoming bit stream from LSB first to an 8-bit word MSB on the left
    shift_register : serialShiftRight
        port map(
            clock => clock,
            enable => enableSignal,
            shiftin => shiftin,
            aclr => clearshift,
            q      => qout
        );
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
process(clock)
variable counter: integer range 0 to SAMPLE*10 := 0;
variable counter_target: integer range 0 to SAMPLE*10 := 16;
variable startbit: integer range 0 to 1; -- when a valid packet is being recieved signal goes to 1

begin

if rising_edge(clock) then
    if shiftin = '0' and startbit = 0 then
        startbit := 1; -- have first bit of packet
        counter_target := counterOffset; -- set the target for the counter,
        clearshift <= '1'; -- clear the shift register
    end if;

    -- increment the sample counter
    if startbit = 1 then
        counter := counter + 1;
    else
        counter := 0;
    end if;

    -- when the sample counter reach 4, set the clear signal back to 0 on the shift register
    if counter = clear_time then
        clearshift <= '0';
    end if;

    -- if the counter reaches the target value add the current serial input to the shift register
    if counter = counter_target then
        enableSignal <= '1';
        counter_target := counter_target + SAMPLE; -- increment the target to next bit value
    else
        enableSignal <= '0';
    end if;

    -- when the start bit is received reset the counter and startbit values back
    if shiftin = '1' and counter = counterMax then
        startbit := 0;
        counter := 0;
        counter_target := counterOffset;
        q <= qout(8 downto 1);
        delayValid <= '1'; -- delay datavalid until qout stabalized
    else
        data_valid <= delayValid; -- tell the system there is valid data on the bus
        delayValid <= '0';
    end if;
end if;
end process;
end mixed;
```

### C.1.1.1.2.1 serialShiftRight

---

-- Author: N/A  
-- Created using Altera Wizard  
-- Description: Shift register for input line, accepts data from rx data line and after 8 bits accepted  
-- sets it onto q

---

LIBRARY ieee;  
USE ieee.std\_logic\_1164.all;

ENTITY serialShiftRight IS  
PORT

(  
    clock          : IN STD\_LOGIC ;  
    enable         : IN STD\_LOGIC ;  
    shiftin,aclr   : IN STD\_LOGIC ;  
    q              : OUT STD\_LOGIC\_VECTOR (9 DOWNT0 0)  
);  
END serialShiftRight;

ARCHITECTURE SYN OF serialShiftRight IS

    SIGNAL sub\_wire0      : STD\_LOGIC\_VECTOR (9 DOWNT0 0);  
    COMPONENT lpm\_shiftreg  
    GENERIC  
    (  
        lpm\_width          : NATURAL;  
        lpm\_direction      : STRING  
    );  
    PORT  
    (  
        enable : IN STD\_LOGIC ;  
        aclr   : IN STD\_LOGIC ;  
        clock  : IN STD\_LOGIC ;  
        q      : OUT STD\_LOGIC\_VECTOR (9 DOWNT0 0);  
        shiftin : IN STD\_LOGIC  
    );  
    END COMPONENT;

BEGIN

    q <= sub\_wire0(9 DOWNT0 0);  
    lpm\_shiftreg\_component : lpm\_shiftreg  
    GENERIC MAP  
    (  
        LPM\_WIDTH => 10,  
        LPM\_DIRECTION => "RIGHT"  
    )  
    PORT MAP  
    (  
        enable => enable,  
        aclr => aclr,  
        clock => clock,  
        shiftin => shiftin,  
        q => sub\_wire0  
    );  
END SYN;

### C.1.1.2 Output FIFO Package

```
-----
-- Author: Tim Li
-- Last Modified: March 14, 2002
-- Purpose: Connects the RS232 output port to the tx FIFO buffer
--          Accepts 16-bit data from the control unit then sends it out as
--          two 8-bit words
-----

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm; -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.rs_232_out_pkg.all;
use work.sync_pkg.all;

package fifoout_pkg is
component fifoout is
    generic (
        InDivisor: positive := 82; --7 115200bps --82 9600bps;
        OutDivisor: positive := 1311 --109 115200bps --1311 9600bps
    );
    port(
        clock, reset: in std_logic;
        inputfifo: in std_logic_vector(15 downto 0);
        writefifo: in std_logic;
        outputserial: out std_logic;
        fifofull: out std_logic
    );
end component fifoout;

end package fifoout_pkg;
```

### C.1.1.2 Output FIFO Component

-- The TX FIFO component used in the above Package.

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;                      -- required for the use of lpm functions
USE lpm.lpm_components.ALL;
library work;
use work.rs_232_out_pkg.all;
use work.sync_pkg.all;

entity fifoout is
    generic (
        InDivisor: positive := 82;          --7 115200bps --82 9600bps;
        OutDivisor: positive := 1311        --109 115200bps --1311 9600bps
    );
    port(
        clock, reset:    in std_logic;
        inputfifo:       in std_logic_vector(15 downto 0);
        writefifo:       in std_logic;
        outputserial:    out std_logic;
        fifofull:        out std_logic
    );
end fifoout;
```

architecture behaviour of fifoout is

```
COMPONENT txfifo IS
    PORT
    (
        data          : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
        wrreq         : IN STD_LOGIC ;
        rdreq         : IN STD_LOGIC ;
        clock         : IN STD_LOGIC ;
        sclr          : IN STD_LOGIC ;
        q             : OUT STD_LOGIC_VECTOR (15 DOWNT0 0);
        full          : OUT STD_LOGIC ;
        empty         : OUT STD_LOGIC
    );
END COMPONENT txfifo;
```

```
component clk_div IS
    PORT
    (
        clock_25Mhz   : IN  STD_LOGIC;
        clock_1MHz     : OUT  STD_LOGIC;
        clock_100KHz   : OUT  STD_LOGIC;
        clock_10KHz    : OUT  STD_LOGIC;
        clock_1KHz     : OUT  STD_LOGIC;
        clock_100Hz    : OUT  STD_LOGIC;
        clock_10Hz     : OUT  STD_LOGIC;
        clock_1Hz      : OUT  STD_LOGIC
    );
end component clk_div;
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component clkdiv is
  -- default to .75H
  -- 25.175 MHz/ (2**24) /2
  generic (Divisor: positive :=33554432); -- clock division rate
  port( fast_clock :    in STD_LOGIC;
        reset      :    in STD_LOGIC;
        slow_clock  :    buffer STD_LOGIC);
end component clkdiv;

-----
signal serialdataload:      std_logic_vector (7 downto 0);
signal serial_output_clock : std_logic;      -- Clock for sending data
signal serial_input_clock  : std_logic;      -- Clock for Receieveing Data
signal enable:              std_logic;
signal sset:                std_logic;
signal loadserial:          std_logic;
signal outsent :            std_logic;
-----
Signal input:  std_logic_vector (15 downto 0);
signal write:  std_logic;
signal read:   std_logic;
signal read_latch: std_logic;
signal output: td_logic_vector (15 downto 0);
signal empty:  std_logic;
signal empty_  slow:std_logic;
-----
signal const_one :    std_logic;
signal const_zero:    std_logic;
-----
type state_type is (s0,s1,s2,s3,s4);
signal state, next_state: state_type;
-----
signal load_now : std_logic;

begin

serialoutclkdiv : clkdiv
  generic map( Divisor => OutDivisor)
  port map(
    fast_clock => clock,
    reset      => reset,
    slow_clock => serial_output_clock
  );

RS232OUT: RS_232_Out
  Port map
  (
    clock => serial_output_clock,
    enable=> const_one,
    sset=> const_zero,
    load=> load_now,
    data=> serialdataload,
    shiftout=> outputserial,
    ready => outsent
  );
load_now <= loadserial and outsent;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
tx_fifo_inst : txfifo PORT MAP (      data => inputfifo,
                                     wrreq => writefifo,
                                     rdreq => read_latch,
                                     clock => clock,
                                     sclr => reset,
                                     q => output,
                                     full => fifofull,
                                     empty => empty);

sync_inst: sync
port map (      longpulse=>read,
            shortpulse=>read_latch,
            clock =>clock   );

const_one <= '1';
const_zero <= '0';

send_out_fsm: process (serial_output_clock,reset)
begin
if reset = '1' then
    state <=s0;
elsif rising_edge (serial_output_clock) then
    case state is
        when s0 =>
            read <='0';
            if empty = '0' then
                state <= s1;
            else
                state <= s0;
            end if;
        when s1 =>
            if outsent = '1' then
                serialdataload <= output (7 downto 0);
                loadserial <= '1';
                state <=s2;
            end if;
        when s2 =>
            loadserial <= '0';
            state <=s3;
        when s3 =>
            if outsent = '1' then
                serialdataload <= output (15 downto 8);
                loadserial <= '1';
                state <=s4;
            end if;
        when s4 =>
            if outsent = '1' then
                read <='1';
                state<=s0;
            end if;
            loadserial <= '0';
        when others =>
            state <= s0;
        end case;
    end if;
end process send_out_fsm;

end behaviour;
```

### C.1.1.2.1 Transmit FIFO

-----  
-- Created by Altera's Megafunction Wizard  
-- this is the TX Fifo  
-----

LIBRARY ieee;  
USE ieee.std\_logic\_1164.all;

```
ENTITY txfifo IS
  PORT
  (
    data      : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    wrreq     : IN STD_LOGIC ;
    rdreq     : IN STD_LOGIC ;
    clock     : IN STD_LOGIC ;
    sclr      : IN STD_LOGIC ;
    q         : OUT STD_LOGIC_VECTOR (15 DOWNT0 0);
    full      : OUT STD_LOGIC ;
    empty     : OUT STD_LOGIC
  );
END txfifo;
```

ARCHITECTURE SYN OF txfifo IS

```
  SIGNAL sub_wire0    : STD_LOGIC ;
  SIGNAL sub_wire1    : STD_LOGIC_VECTOR (15 DOWNT0 0);
  SIGNAL sub_wire2    : STD_LOGIC ;
```

```
  COMPONENT lpm_fifo
  GENERIC (
    lpm_width          : NATURAL;
    lpm_numwords       : NATURAL;
    lpm_widthu         : NATURAL;
    lpm_showahead      : STRING;
    lpm_hint           : STRING
  );
  PORT (
    rdreq  : IN STD_LOGIC ;
    sclr   : IN STD_LOGIC ;
    empty  : OUT STD_LOGIC ;
    clock  : IN STD_LOGIC ;
    q      : OUT STD_LOGIC_VECTOR (15 DOWNT0 0);
    wrreq  : IN STD_LOGIC ;
    data   : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    full   : OUT STD_LOGIC
  );
END COMPONENT;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
BEGIN
    empty  <= sub_wire0;
    q      <= sub_wire1(15 DOWNT0 0);
    full   <= sub_wire2;

    lpm_fifo_component : lpm_fifo
    GENERIC MAP (
        LPM_WIDTH => 16,
        LPM_NUMWORDS => 4,
        LPM_WIDTHU => 2,
        LPM_SHOWAHEAD => "ON",
        LPM_HINT => "USE_EAB=ON"
    )
    PORT MAP (
        rdreq => rdreq,
        sclr => sclr,
        clock => clock,
        wrreq => wrreq,
        data => data,
        empty => sub_wire0,
        q => sub_wire1,
        full => sub_wire2
    );

END SYN;
```

```

(
    clock      : IN STD_LOGIC ; --clock to register
    enable     : IN STD_LOGIC ; -- set high
    sset      : IN STD_LOGIC ; -- synchronous clear
    load       : IN STD_LOGIC ; -- load should be high for 1 clock cycle to load
    register   : IN STD_LOGIC_VECTOR (7 DOWNTO 0); --8 bits of data to
send
    shiftout   : out STD_LOGIC; -- data sent out to level converter
    ready      : out STD_LOGIC); -- handshaking signal to indicate register is
ready to
                                -- receive the next packet
end component RS_232_Out;
end package RS_232_Out_pkg;
```

C.1.1.2.2 Component: RS\_232\_Out

---

-- Authors: Tim Bensler &  
Eric Chan

--

-- File Name:  
RS\_232\_Out.vhd

--

-- Description: The RS\_232\_Out Component used for the actual communication.

---

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity RS_232_out is
  PORT
  (
    clock      : IN STD_LOGIC ; --clock to register
    enable     : IN STD_LOGIC ; -- register enable
    sset       : IN STD_LOGIC ; -- synchronous clear - sets output to one
    load       : IN STD_LOGIC ; -- load high pulsed to load register
    data       : IN STD_LOGIC_VECTOR (7 DOWNTO 0); --8 bits of data to send start
                                                    --& stop bits automatically -
                                                    --added on
    shiftout   : out STD_LOGIC; -- data sent out to level converter
    ready      : out STD_LOGIC -- handshaking signal to indicate register ready to
                                -- receive next packet
  );
end RS_232_out;

architecture tenbit of RS_232_Out is

  component myShiftOut
    PORT
    (
      clock      : IN STD_LOGIC ;
      enable     : IN STD_LOGIC ;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
sset      : IN STD_LOGIC ;
load      : IN STD_LOGIC ;
data      : IN STD_LOGIC_VECTOR (9 DOWNT0 0);
shiftout  : OUT STD_LOGIC;
shiftin   : IN STD_LOGIC  -- used to shift in ones behind data going out
);
end component;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
signal tied1: std_logic;
signal dataOut: std_logic_vector(9 downto 0);
constant datawidth: positive:= 8;

begin
tied1 <= '1';
dataOut <= '1' & data & '0'; --& '1';--'1' & data & '0'; -- add start and stop bits to frame
-- least significant bit sent first

register1 : myShiftOut

    PORT map (  clock=> clock,
                enable=> enable,
                sset => sset,
                shiftin => tied1,          --fills in ones behind data shifted out
                data=> dataOut,
                load => load,
                shiftout => shiftout
                );

process(clock)
variable counter : natural range 0 to 15 := 0;

begin
if rising_edge(clock) then
    if counter > datawidth and load = '0' then
        ready <= '1';
    elsif load = '1' then
        ready <= '0';
        counter := 0;
    else
        counter:= counter + 1;
    end if;
end if;

end process;

end tenbit;
```



### C.1.1.2.2.1 Component: myShiftOut

```
-- Component that takes parallel data and shifts the data
---- Author      : N/A
-- Student ID   : N/A
-- Created      : March 2000
-- File Name    : myShiftOut.vhd
-- Description   : Created using the Altera code Wizard. Shift register used to shift out data bit by bit to the
--                RS232 tx data line
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
```

```
ENTITY myShiftOut IS
  PORT
  (
    clock      : IN STD_LOGIC ;
    enable     : IN STD_LOGIC ;
    shiftin    : IN STD_LOGIC ;
    load       : IN STD_LOGIC ;
    sset       : IN STD_LOGIC ;
    data       : IN STD_LOGIC_VECTOR (9 DOWNT0 0);
    shiftout   : OUT STD_LOGIC
  );
END myShiftOut;
```

ARCHITECTURE SYN OF myShiftOut IS

```
  SIGNAL sub_wire0      : STD_LOGIC ;

  COMPONENT lpm_shiftreg
  GENERIC (
    LPM_WIDTH           :          POSITIVE;
    LPM_DIRECTION       :          STRING;
    LPM_SVALUE          :          POSITIVE
  );
  PORT (
    enable : IN STD_LOGIC ;
    load   : IN STD_LOGIC ;
    clock  : IN STD_LOGIC ;
    data   : IN STD_LOGIC_VECTOR (9 DOWNT0 0);
    shiftout : OUT STD_LOGIC ;
    sset   : IN STD_LOGIC ;
    shiftin : IN STD_LOGIC
  );
END COMPONENT;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
BEGIN
    shiftout    <= sub_wire0;

    lpm_shiftreg_component : lpm_shiftreg
    GENERIC MAP (
        LPM_WIDTH => 10,
        LPM_DIRECTION => "RIGHT",
        LPM_SVALUE => 1
    )
    PORT MAP (
        enable => enable,
        load => load,
        clock => clock,
        data => data,
        sset => sset,
        shiftin => shiftin,
        shiftout => sub_wire0
    );
END SYN;
```

### C.1.1.2.2.2 Package: synchronizer

```
-----  
-- Component that takes parallel data and shifts the data  
-- Author      : Timmy Li  
-- Created     : March 13, 2002  
-- File Name   : synch.vhd  
-- Description : Synchronizes a long pulse with a short pulse  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;  
  
package sync_pkg is  
  
  COMPONENT sync  
    port (longpulse: in std_logic;  
          shortpulse: out std_logic;  
          clock :in std_logic  
          );  
  END COMPONENT sync;  
  
end package sync_pkg;
```

### C.1.1.2.2.2 Package: synchronizer

```
-----  
-- Component that takes parallel data and shifts the data  
-- Author      : Timmy Li  
-- Created     : March 13, 2002  
-- File Name   : synch.vhd  
-- Description  : Synchronizes a long pulse with a short pulse  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity sync is  
port (longpulse: in std_logic;  
      shortpulse: out std_logic;  
      clock :in std_logic  
      );  
end sync;
```

```
architecture behavioral of sync is  
-----
```

```
type sync_type is (upclock,downclock,hold);  
signal sync, next_sync: sync_type;  
-----
```

```
begin
```

```
sync_signal_fsm: process (sync)  
begin  
  case sync is  
    when upclock =>  
      if longpulse = '1' then  
        shortpulse <= '1';  
        next_sync <= downclock;  
      else  
        next_sync <= upclock;  
      end if;  
    when downclock =>  
      shortpulse <= '0';  
      next_sync <=hold;  
    when hold =>  
      if longpulse = '1' then  
        next_sync <=hold;  
      else  
        next_sync <=upclock;  
      end if;  
    when others =>  
      next_sync <= upclock;  
  end case;  
end process sync_signal_fsm;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
sync_signal : process (clock)
begin
if rising_edge(clock) then
sync <=next_sync;
end if;
end process sync_signal;

end behavioral ;
```

### APPENDIX C.2 VHDL Code For LPM RAM

```
-----
-- LPM RAM Module
--
-- written by: Andrew Ling & Darren Gonek
-- purpose: Instantiates LPM RAM and provides internal addressing so that the control
-- is not concerned with it
-- written on: Mar 10, 2002
-----

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity our_ram is
    generic (
        ram_data_width : positive := 24;           -- width of data in ram
        ram_address_width : positive := 8;         -- width of address in ram
        ram_data_size : positive := 256           -- number of values in ram
    );
    PORT(
        ram_data_in : in std_logic_vector(ram_data_width - 1 downto 0);    -- input data
        write : in std_logic;                                               -- determines write
        read : in std_logic;                                                -- determines read
        clock, reset : in std_logic;                                        -- synchronized with system clock
        ram_data_out : out std_logic_vector(ram_data_width - 1 downto 0)    -- output data
    );
end our_ram;

architecture behavior of our_ram is

    signal read_counter_sig : std_logic_vector(ram_address_width - 1 downto 0) := "00000000";
        -- read address ctr
    signal write_counter_sig : std_logic_vector(ram_address_width - 1 downto 0) := "00000000";
        -- write address ctr
    signal current_address : std_logic_vector(ram_address_width - 1 downto 0) := "00000000";
        -- current address
    signal write_enable : std_logic := '0';
    signal cur_read_val, cur_write_val : std_logic := '0';
    signal temp_out : std_logic_vector(ram_data_width - 1 downto 0); -- := "000000000000000000000000";

    type state_type is (initial,s0,s1,s2);
    signal state, next_state : state_type := s0;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

begin

-- instantiate the lpm\_ram module used for storage of data values. The width of each word in  
-- RAM is 24 bits. This length was chosen because on each read/write, we are expecting and output  
-- 3 8-byte values.

tiny\_ram : lpm\_ram\_dq

GENERIC MAP (

lpm\_width => ram\_data\_width, -- width of input and output data  
lpm\_widthad => ram\_address\_width, -- width of address  
lpm\_indata => "UNREGISTERED", -- unregistered for asynchronous use  
lpm\_outdat => "UNREGISTERED", -- unregistered for asynchronous use  
lpm\_numwords => ram\_data\_size, -- maximum number of data  
lpm\_address\_control => "UNREGISTERED"  
)

PORT MAP (

data => ram\_data\_in, -- signals on the ports  
we => write\_enable, -- determines write or read  
address => current\_address,  
q => ram\_data\_out -- output data

);

-- process controlling the entire state machine

ram\_state\_machine : process (clock)

begin

if rising\_edge(clock) then

case state is

when initial =>

write\_counter\_sig <= (others => '0');  
read\_counter\_sig <= (others => '0');  
current\_address <= (others => '0');  
write\_enable <= '0';  
cur\_read\_val <= '0';  
cur\_write\_val <= '0';  
next\_state <= s0;

when s0 =>

if read = '0' and write = '1' then -- received write command from the control unit  
write\_enable <= '1'; -- setting write\_enable  
current\_address <= write\_counter\_sig; -- set the current write address  
next\_state <= s1;  
cur\_read\_val <= '0'; -- store the current values inputted from the control  
cur\_write\_val <= '1'; -- unit as they are used in future states

elsif read = '1' and write = '0' then -- received read command from the control unit  
write\_enable <= '0'; -- reset write enable

if read\_counter\_sig = write\_counter\_sig then  
read\_counter\_sig <= (others => '0');  
current\_address <= (others => '0');

else

current\_address <= read\_counter\_sig; -- set current read address

end if;

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```

        next_state <= s1;
        cur_read_val <= '1'; -- store current values inputted from the control unit
        cur_write_val <= '0';
    else -- otherwise we reset to the original state
        next_state <= s0;
        cur_read_val <= '0';
        cur_write_val <= '0';
    end if;
when s1 =>
    write_enable <= '0'; -- disable writing to RAM
    current_address <= write_counter_sig; -- set current address =that of write counter,

    -- as that points to the end of where values are stored in memory
    next_state <= s2;
when s2=>
    -- if writing to RAM, ensure that the value being written still exists in the physical
    -- limitations of RAM.
    if cur_read_val = '0' and cur_write_val = '1' then
        if write_counter_sig <= "11111111" then
            write_counter_sig <= write_counter_sig + 1;
        end if;

        -- otherwise if we are reading, than ensure that the current address does not exceed the
        -- physical limits, and does not exceed the value of the write pointer, since the values
        -- in the RAM beyond that value are garbage
        elsif cur_read_val = '1' and cur_write_val = '0' then
            if read_counter_sig < write_counter_sig then
                read_counter_sig <= read_counter_sig + 1;
            else
                read_counter_sig <= (others => '0');
            end if;
        end if;
    next_state <=s0; -- reset back to original state
    cur_read_val <= '0';
    cur_write_val <= '0';
when others =>
    next_state <= s0;
    cur_read_val <= '0';
    cur_write_val <= '0';
end case;
end if;
end process ram_state_machine;

change_state: process(clock, reset)
begin
    if falling_edge(clock) then
        if reset = '1' then
            state <= next_state;
        else
            state <= initial;
        end if;
    end if;
end process change_state;

end behavior;
```



## Appendix C3 – VHDL code for Busmaster

This component has no package attached.

```
-----
-- Component that sends the weights from either the RNG or RAM
-- Send the address of weight to read and the weight to change
-- as well as the command word, to see what instruction to
--perform : cmd = "00" := Read weight
--              cmd = "01" := Write to weighth
--              cmd = "11" := Start Calculations
-- Author      : Guillermo Barreiro
-- Student ID   : 1042071
-- Date        : March 18, 2001
-- File Name    : busmaster.vhd
-- Architecture : Behavioral
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.nn_pack.all;

entity busmaster is
    generic ( datawidth : positive := 8;
              halfwidth : positive := 4);
    port ( clock : in std_logic;
          read_rand: in std_logic;
          read_data: in std_logic;
          assert_rand: in std_logic;
          assert_data: in std_logic;
          assert_saved : in std_logic;
          load_addr: in std_logic;
          clear: in std_logic;
          in_sel: in std_logic_vector(halfwidth - 1 downto 0);
          incr_sel : in std_logic;
          load_sel : in std_logic;
          data: inout std_logic_vector(datawidth - 1 downto 0);
          addr: out std_logic_vector(halfwidth - 1 downto 0);
          sel: out std_logic_vector(halfwidth - 1 downto 0);
          in_rng: in std_logic_vector(datawidth - 1 downto 0);
          in_data: in std_logic_vector(datawidth - 1 downto 0);
          in_addr: in std_logic_vector(halfwidth - 1 downto 0);
          done: out std_logic;
          stored_data: out std_logic_vector(datawidth - 1 downto 0));

end busmaster;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

architecture behavioural of busmaster is

```
    signal rnd_data, saved_data : std_logic_vector (datawidth -1 downto 0);
    signal zero : std_logic_vector ( halfwidth - 1 downto 1);
    signal donedummy, incr_addr, addr_done, sel_done : std_logic;

begin

random_data : process(clear, clock, data)
begin
    if clear = '1' then
        rnd_data <= (others => '0');
    elsif clock'event and clock = '1' then
        if read_rand = '1' then
            rnd_data <= in_rng;
        end if;
    end if;
end process random_data;

bus_data : process(clear, clock, data)
begin
    if clear = '1' then
        saved_data <= (others => '0');
    elsif clock'event and clock = '1' then
        if read_data = '1' then
            saved_data <= data;
        end if;
    end if;
end process bus_data;

Stored_data <= saved_data;

dataout_mux : process(saved_data, rnd_data, in_data, assert_saved, assert_rand, assert_data)
begin
    if assert_saved = '1' then
        data <= saved_data;
    elsif assert_rand = '1' then
        data <= rnd_data;
    elsif assert_data = '1' then
        data <= in_data;
    else
        data <= (others => 'Z');
    end if;
end process dataout_mux;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
counters : process (in_sel, load_sel, incr_sel, clock, clear, in_addr, load_addr, incr_addr)
    variable count_addr : integer range 0 to No_neurons - 1;
    variable count_sel: integer range 0 to No_weights - 1;
begin
    if clear = '1' then
        count_addr := 0;
        count_sel := 0;
        done <= '0';
    elsif clock'event and clock = '1' then
        if load_sel = '1' then
            count_sel := conv_integer(in_sel);
            done <= '0';
        elsif incr_sel = '1' then
            if count_sel = No_weights - 1 then
                count_sel := 0;
                if count_addr = No_neurons - 1 then
                    done <= '1';
                    count_addr := 0;
                else
                    count_addr := count_addr + 1;
                    done <= '0';
                end if;
            else
                count_sel := count_sel + 1;
                done <= '0';
            end if;
        end if;
    end if;

    if load_addr = '1' then
        count_addr := conv_integer(in_addr);
        donedummy <= '0';
    end if;

    addr <= conv_std_logic_vector( count_addr, halfwidth);
    sel <= conv_std_logic_vector( count_sel, halfwidth);
end process;

end behavioural;
```

# Appendix C4 – VHDL code for Neural Network Datapath

## C.4.1 – Neural Network Package

---

```
-- Package for Components for Neural Network Datapath
-- Author      : Guillermo Barreiro
-- Student ID   : 1042071
-- Date        : December 6 2001 - March 20 2002
-- File Name    : nn_pack.vhd
```

---

```
library ieee;
use ieee.std_logic_1164.all;
```

```
package nn_pack is
```

```
constant bit_size : positive := 8;
```

```
constant No_weights : integer := 2;
constant No_Neurons : integer := 3;
```

```
subtype neuron_address is integer range 0 to 3;
```

```
component nn_dp_2 is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : in neuron_address;
          datain : in std_logic_vector ( data_bitsize -1 downto 0);
          dataout : out std_logic_vector ( data_bitsize -1 downto 0);
          In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic;
          result : out std_logic_vector ( data_bitsize -1 downto 0);
          done : out std_logic);
```

```
end component;
```

```
component nn_dp_3 is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : in std_logic_vector( 1 downto 0); --neuron_address;
          databus : inout std_logic_vector ( data_bitsize -1 downto 0);
          In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic;
          result : out std_logic_vector ( data_bitsize -1 downto 0);
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
        done : out std_logic);
end component;

component neuron_weights_2 is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8;
              my_address : neuron_address := 0);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : in neuron_address;
          datain : in std_logic_vector ( data_bitsize -1 downto 0);
          dataout : out std_logic_vector ( data_bitsize -1 downto 0);
          In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic;
          out1 : out std_logic_vector ( data_bitsize -1 downto 0);
          done : out std_logic);
end component;

component hold_weights_2 is
    generic ( cmd_bitsize,sel_bitsize : positive := 2;
              data_bitsize: positive := 8;
              my_address : neuron_address := 0);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : neuron_address;
          datain : in std_logic_vector ( data_bitsize -1 downto 0);
          dataout : out std_logic_vector ( data_bitsize -1 downto 0);
          we1, we2 : out std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic);
end component;

component neuron_weights is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8;
              my_address : neuron_address := 0);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : in neuron_address;
          databus : inout std_logic_vector ( data_bitsize -1 downto 0);
          In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic;
          out1 : out std_logic_vector ( data_bitsize -1 downto 0);
          done : out std_logic);
end component;

component hold_weights is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8;
              my_address : neuron_address := 0);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : neuron_address;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
        databus : inout std_logic_vector ( data_bitsize -1 downto 0);
        we1, we2 : out std_logic_vector ( data_bitsize - 1 downto 0);
        clock, reset : in std_logic);
end component;
component prng is
    generic(Width: positive := 8);
    port      (
        clock: in std_logic;
        reset : in std_logic;
        output: out std_logic_vector(Width-1 downto 0)
    );
end component;

component neuron2 is
    generic( Bitsize : positive := 8;
        NumInputs : positive := 2);
    port(
        In1, In2 : in std_logic_vector( BitSize - 1 downto 0);
        We1, We2 : in std_logic_vector( BitSize - 1 downto 0);
        start, reset, clock : in std_logic;
        result: out std_logic_vector(BitSize - 1 downto 0);
        done, overflow : out std_logic);
end component;

component neuron is
    generic( Bitsize : positive := 8;
        NumInputs : positive := 2);
    port(
        In1, In2 : in std_logic_vector( BitSize - 1 downto 0);
        We1, We2 : in std_logic_vector( BitSize - 1 downto 0);
        start, reset, clock : in std_logic;
        result: out std_logic_vector(BitSize - 1 downto 0);
        done, overflow : out std_logic);
end component;

component weights_io is
    generic ( cmd_bitsize : positive := 2;
        sel_bitsize : positive := 2;
        data_bitsize: positive := 8);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
        sel : in std_logic_vector ( sel_bitsize-1 downto 0);
        addr : in std_logic;
        datain : inout std_logic_vector ( data_bitsize -1 downto 0);
        clock, reset : in std_logic;
        outneuron : out std_logic_vector( data_bitsize -1 downto 0));
end component;

component clk_div IS
    PORT
    (
        clock_25Mhz          : IN  STD_LOGIC;
        clock_5MHZ           : OUT  Std_logic;
        clock_1MHz           : OUT  STD_LOGIC;
        clock_100KHz         : OUT  STD_LOGIC;
        clock_10KHz          : OUT  STD_LOGIC;
        clock_1KHz           : OUT  STD_LOGIC;
        clock_100Hz          : OUT  STD_LOGIC;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```

        clock_10Hz          : OUT STD_LOGIC;
        clock_1Hz           : OUT STD_LOGIC);
END component;

component LFSR_GENERIC is
    generic(Width: positive := 8);           -- length of pseudo-random sequence
    port (
        clock: in std_logic;
        reset: in std_logic;                -- active high reset
        load: in std_logic;                 -- active high load (assert this to use as regular reg)
        enable: in std_logic;               -- active high enable
        parallel_in: in std_logic_vector(Width-1 downto 0); -- parallel seed input
        parallel_out: out std_logic_vector(Width-1 downto 0) -- parallel data out
    );
end component;

component controlneuron6 is
    generic( NumInputs : positive := 2;
             BitSize : positive := 8);
    port( clock : in std_logic;
          reset : in std_logic;
          start : in std_logic;
          done : out std_logic;
          loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : out std_logic;
          selinputs, selweights, selout: out std_logic;
          sign, overflow2: in std_logic
    );
end component;

component controlneuron3 is
    generic( NumInputs : positive := 2;
             BitSize : positive := 8);
    port( clock : in std_logic;
          reset : in std_logic;
          start : in std_logic;
          done : out std_logic;
          loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : out std_logic;
          selinputs, selweights, selout: out std_logic;
          sign, overflow2: in std_logic);
end component;

component neurondatapath5 is
    generic( BitSize : positive := 8);
    port(
        In1, In2 : in std_logic_vector( BitSize-1 downto 0);
        We1, We2 : in std_logic_vector( BitSize-1 downto 0);
        loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : in std_logic;
        selinputs, selweights, selout: in std_logic;
        clock : in std_logic;
        reset : in std_logic;
        output: out std_logic_vector( BitSize-1 downto 0);
        sign, overflow2 : out std_logic);
end component;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component neurondatapath3 is
generic( BitSize : positive := 8);
port(
```

```
    In1, In2 : in std_logic_vector( BitSize-1 downto 0);
    We1, We2 : in std_logic_vector( BitSize-1 downto 0);
    loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : in std_logic;
    selinputs, selweights, selout: in std_logic;
    clock : in std_logic;
    reset : in std_logic;
    output: out std_logic_vector( BitSize-1 downto 0);
    sign, overflow2 : out std_logic);
```

```
end component;
```

```
component neurondatapath2 is
generic( BitSize : positive := 8);
port(
```

```
    In1, In2 : in std_logic_vector( BitSize-1 downto 0);
    We1, We2 : in std_logic_vector( BitSize-1 downto 0);
    loadM, loadQ, loadNV, loadOV, loadOUT, loadSign: in std_logic;
    selinputs, selweights, selout: in std_logic;
    clock : in std_logic;
    reset : in std_logic;
    output: out std_logic_vector( BitSize-1 downto 0);
    sign, overflow2 : out std_logic);
```

```
end component;
```

```
component controlneuron2 is
```

```
    generic( NumInputs : positive := 2;
             BitSize : positive := 8);
```

```
    port( clock : in std_logic;
           reset : in std_logic;
           start : in std_logic;
           done : out std_logic;
           loadM, loadQ, loadNV, loadOV, loadOUT, loadSign: out std_logic;
           selinputs, selweights, selout: out std_logic;
           sign, overflow2: in std_logic);
```

```
end component;
```

```
component controlneuron is
```

```
    generic( NumInputs : positive := 2;
             BitSize : positive := 8);
```

```
    port( clock : in std_logic;
           reset : in std_logic;
           start : in std_logic;
           done : out std_logic;
           loadM, loadP, loadQ, loadF, loadOV, loadOUT, loadSign : out std_logic;
           selinputs, selweights, selout: out std_logic;
           selP, selQ: out std_logic_vector(1 downto 0);
           clearPF: out std_logic;
           opMode : out std_logic;
           boothlogic : in std_logic_vector( 1 downto 0);
           sign, overflow1, overflow2: in std_logic);
```

```
end component;
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

component neurondatapath is

generic( BitSize : positive := 8);

port(

In1, In2 : in std\_logic\_vector( BitSize-1 downto 0);  
We1, We2 : in std\_logic\_vector( BitSize-1 downto 0);  
loadM, loadP, loadQ, loadF, loadOV, loadOUT, loadSign : in std\_logic;  
selinputs, selweights, selout : in std\_logic;  
selP, selQ : in std\_logic\_vector( 1 downto 0);  
clearPF : in std\_logic;  
clock : in std\_logic;  
reset : in std\_logic;  
opMode : in std\_logic;  
output : out std\_logic\_vector( BitSize-1 downto 0);  
boothlogic : out std\_logic\_vector( 1 downto 0);  
sign, overflow1, overflow2 : out std\_logic);

end component;

component mulp8 is -- 16 x 16 = 32 bit unsigned product multiplier

port(a : in std\_logic\_vector(7 downto 0); -- multiplicand

b : in std\_logic\_vector(7 downto 0); -- multiplier

prod : out std\_logic\_vector(15 downto 0)); -- product

end component;

component adder18bits is

port (InputA : in std\_logic\_vector(17 downto 0);

InputB : in std\_logic\_vector(17 downto 0);

cin : in std\_logic;

cout : out std\_logic;

sum : out std\_logic\_vector(17 downto 0));

end component;

component myalu is

generic( N : positive := 4);

port(

A : in std\_logic\_vector( N-1 downto 0);

B : in std\_logic\_vector( N-1 downto 0);

opMode : in std\_logic; -- indicate addition or subtraction

results : out std\_logic\_vector(N-1 downto 0);

cout : out std\_logic);

end component;

component fastadderN is

generic(N : positive := 4);

port (a : in std\_logic\_vector(N-1 downto 0);

b : in std\_logic\_vector(N-1 downto 0);

cin : in std\_logic;

sum : out std\_logic\_vector(N-1 downto 0);

cout : out std\_logic;

groupg : out std\_logic;

groupg : out std\_logic);

end component;

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component cla_adder
  port (a : in std_logic;
        b : in std_logic;
        cin : in std_logic;
        p : out std_logic;
        g : out std_logic;
        sum : out std_logic);
end component;

component clgN is
  generic(N : positive := 4);
  port (p : in std_logic_vector(N-1 downto 0);
        g : in std_logic_vector(N-1 downto 0);
        cin : in std_logic;
        cout : out std_logic_vector(N-1 downto 0);
        groupp : out std_logic;
        groupg : out std_logic);
end component;

component adderN is
  generic(N : positive := 4);
  port (a : in std_logic_vector(N-1 downto 0);
        b : in std_logic_vector(N-1 downto 0);
        cin : in std_logic;
        sum : out std_logic_vector(N-1 downto 0);
        cout : out std_logic);
end component;

component mux2N_N
  generic(N : positive := 4);
  port (
    Input0 : in std_logic_vector(N-1 downto 0);
    Input1 : in std_logic_vector(N-1 downto 0);
    Sel : in std_logic;
    Output : out std_logic_vector(N-1 downto 0)
  );
end component;

component shiftregisterN is
  generic(N : positive := 4);
  port ( Input : in std_logic_vector(N-1 downto 0);
        Load : in std_logic;
        Clock : in std_logic;
        Reset : in std_logic;
        Rin : in std_logic;
        Lin : in std_logic;
        Sel : in std_logic_vector(1 downto 0);
        Output : out std_logic_vector(N-1 downto 0));
end component;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component dfflipflop is
  port ( Input  : in std_logic;
        Load   : in std_logic;
        clk     : in std_logic;
        Reset   : in std_logic;
        Output  : out std_logic);
end component;

component registerN
  generic(N : positive := 4);
  port ( Input  : in std_logic_vector(N-1 downto 0);
        Load   : in std_logic;
        Clock   : in std_logic;
        Reset   : in std_logic;
        Output  : out std_logic_vector(N-1 downto 0));
end component;

component fullalu is
  generic( N : positive := 5);
  port(
    A : in std_logic_vector( N-1 downto 0);
    B : in std_logic_vector( N-1 downto 0);
    results: out std_logic_vector((2*N)-1 downto 0); -- calculated results
    start : in std_logic;
    done  : out std_logic;
    clock : in std_logic;
    reset : in std_logic;
    sel_operation : in std_logic_vector(1 downto 0));
end component;

component multbehavioral is
  generic(Bitsize : positive := 8);
  port ( A, B : in std_logic_vector( Bitsize - 1 downto 0);
        F  : out std_logic_vector ( (2*Bitsize) -1 downto 0));
end component;

end package;
```

### C.4.2 – Neuron Prototypes

#### **C.4.2.1 - Neuron – Prototype 1**

-----

-- Neuron implemented in hardware with booth multiplier  
-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : January 31, 2001  
-- File Name : neuron.vhd  
-- Architecture : structural  
-- Description : Neuron made of Datapath and State Machine.

-----

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.nn_pack.all;

entity neuron is
generic( BitSize : positive := 8;
        NumInputs : positive := 2);
port(
    In1, In2 : in std_logic_vector( BitSize - 1 downto 0);
    We1, We2 : in std_logic_vector( BitSize - 1 downto 0);
    start, reset, clock : in std_logic;
    result: out std_logic_vector(BitSize - 1 downto 0);
    done, overflow : out std_logic);
end neuron;

architecture structural of neuron is

signal loadmwire, loadpwire, loadqwire, loadfwire, loadovwire, loadoutwire, loadsignwire : std_logic;
signal selinputswire, selweightswire, seloutwire, clearpfwire : std_logic;
signal opmodewire, signwire, ovf1wire, ovf2wire : std_logic;
signal boothwires, selpwire, selqwire : std_logic_vector( 1 downto 0);

begin

statemachine: controlneuron
    generic map ( NumInputs, BitSize)
    port map ( clock => clock,
              reset => reset,
              start => start,
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
done => done,

    loadM => loadmwire,
    loadP => loadpwire,
    loadQ => loadqwire,
    loadF => loadfwire,
    loadOV => loadovwire,
    loadOUT => loadoutwire,
    loadSign => loadsignwire,
    selinputs => selinputswire,
    selweights => selweightswire,
    selout => seloutwire,
    selP => selpwire,
    selQ => selqwire,
    clearPF => clearpfwire,
    opMode => opmodewire,
    boothlogic => boothwires,
    sign => signwire,
    overflow1 => ovf1wire,
    overflow2 => ovf2wire);
```

```
overflow <= ovf1wire or ovf2wire;
```

```
datapath: neurondatapath
```

```
generic map (BitSize)
```

```
port map(
```

```
    In1 => In1,
    In2 => In2,
    We1 => We1,
    We2 => We2,
    loadM => loadmwire,
    loadP => loadpwire,
    loadQ => loadqwire,
    loadF => loadfwire,
    loadOV => loadovwire,
    loadOUT => loadoutwire,
    loadSign => loadsignwire,
    selinputs => selinputswire,
    selweights => selweightswire,
    selout => seloutwire,
    selP => selpwire,
    selQ => selqwire,
    clearPF => clearpfwire,
    clock => clock,
    reset => reset,
    opMode => opmodewire,
    output => result,
    boothlogic => boothwires,
    sign => signwire,
    overflow1 => ovf1wire,
    overflow2 => ovf2wire);
```

```
end architecture structural;
```

### C.4.2.1.1 – Component Datapath

-----  
-- Neuron component Datapath  
-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : January 31, 2002  
-- File Name : neurondatapath.vhd  
-- Architecture : structural  
-----

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
library work;  
use work.nn_pack.all;
```

```
entity neurondatapath is  
generic( BitSize : positive := 8);  
port(  
    In1, In2 : in std_logic_vector( BitSize-1 downto 0);  
    We1, We2 : in std_logic_vector( BitSize-1 downto 0);  
    loadM, loadP, loadQ, loadF, loadOV, loadOUT, loadSign : in std_logic;  
    selinputs, selweights, selout : in std_logic;  
    selP, selQ : in std_logic_vector( 1 downto 0);  
    clearPF : in std_logic;  
    clock : in std_logic;  
    reset : in std_logic;  
    opMode : in std_logic;  
    output : out std_logic_vector( BitSize-1 downto 0);  
    boothlogic : out std_logic_vector( 1 downto 0);  
    sign, overflow1, overflow2 : out std_logic);  
end neurondatapath;
```

architecture structural of neurondatapath is

```
signal mwires, pwires, qwires, aluwiresmult, muxtomwires, muxtoqwires, muxtooutwires, minusone, one :  
std_logic_vector( BitSize-1 downto 0); signal resultwires : std_logic_vector( (2*BitSize) + 1 downto 0); -- It  
has to be 18 bits to avoid overflow  
signal additionwires, ovwires : std_logic_vector( (2*BitSize) + 1 downto 0); -- They have to be 18 bits for  
addition to avoid overflow  
signal ground, clearresetwire, coutwire : std_logic;  
signal zeros : std_logic_vector(BitSize-1 downto 1);  
signal donothing : std_logic_vector( 3 downto 0);
```

begin

```
ground <= '0';  
clearresetwire <= reset and clearPF;
```

```
registerM : registerN  
    generic map(BitSize)  
    port map (    Input => muxtomwires,
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
Load    => loadM,
        Clock    => clock,
        Reset    => reset,
Output => mwires);

muxtoM: mux2N_N
  generic map (BitSize)
  port map (
    Input0 => In1,
    Input1 => In2,
    Sel => selinputs,
    Output => muxtomwires);

registerP : shiftregisterN
  generic map (BitSize)
  port map (
    Input => aluwiresmult,
    Load    => loadP,
    Clock    => clock,
    Reset    => clearresetwire,
    Rin    => ground,
    Lin    => ground,
    Sel    => selP,
    Output => pwires);

muxtoQ: mux2N_N
  generic map (BitSize)
  port map (
    Input0 => We1,
    Input1 => We2,
    Sel => selweights,
    Output => muxtoqwires);

registerQ: shiftregisterN
  generic map (BitSize)
  port map (
    Input => muxtoqwires,
    Load    => loadQ,
    Clock    => clock,
    Reset    => reset,
    Rin    => pwires(0),
    Lin    => ground,
    Sel    => selQ,
    Output => qwires);

dflopF: dflop
  port map (Input => qwires(0),
    Load => loadF,
    clk    => clock,
    Reset => clearresetwire,
    Output => boothlogic(0));
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
alumultiplier: myalu
    generic map (BitSize)
    port map(
        A => pwires,
        B => mwires,
        opMode => opMode,
        results => aluwiresmult,
        cout => overflow1);

boothlogic(1) <= qwires(0);
resultwires <= pwires(BitSize - 1) & pwires(BitSize-1) & pwires & qwires;

regoldvalue : registerN
    generic map (18)--(2*BitSize + 2)
    port map (
        Input => additionwires,
        Load  => loadOV,
        Clock  => clock,
        Reset  => reset,
        Output => ovwires);

fastadder : adder18bits
    port map ( InputA => resultwires,
        InputB => ovwires,
        cin => ground,
        cout => overflow2,
        sum => additionwires);

dflipflopSign: dflipflop
    port map (
        Input => ovwires((2*BitSize) + 1),
        Load => loadSign,
        clk   => clock,
        Reset => reset,
        Output => sign); -- Takes last bit to see sign

minusone <= (others => '1');
zeros <= ( others => '0');
one <= zeros & '1';

muxtosign: mux2N_N
    generic map (BitSize)
    port map (
        Input0 => minusone,
        Input1 => one,
        Sel => selout,
        Output => muxtooutwires);

registerOUT : registerN
    generic map (BitSize)
    port map (
        Input => muxtooutwires,
        Load  => loadOUT,
        Clock  => clock,
        Reset  => reset,
        Output => output);
end architecture structural;
```



### C.4.2.1.2 – Component Control Unit

```
-----
-- Neuron control unit
-- Author      : Guillermo Barreiro
-- Student ID   : 1042071
-- Date        : January 31, 2002
-- File Name    : controlneuron.vhd
-- Architecture : Behavioural
-- Description   : State machine with asynthonous reset.
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity controlneuron is
    generic( NumInputs : positive := 2;
             BitSize : positive := 8);
    port( clock : in std_logic;
          reset : in std_logic;
          start : in std_logic;
          done : out std_logic;
          loadM, loadP, loadQ, loadF,loadOV, loadOUT, loadSign : out std_logic;
          selinputs, selweights, selout: out std_logic;
          selP, selQ: out std_logic_vector(1 downto 0);
          clearPF: out std_logic;
          opMode : out std_logic;
          boothlogic : in std_logic_vector( 1 downto 0);
          sign, overflow1, overflow2: in std_logic);
end controlneuron;

architecture behavioral of controlneuron is
    -- Declaration of states
    type state_type is (Sidle, Sinitmult, Sbooth, Sshiftright, Sadd, Ssign,Sfinal, Swait, Sloadp, Swaitadd,
    swaitsign, Swaitfinal, Sdone);
    signal present_state: state_type;
begin
    -- Create a process that handles the state transition
    comb_logic : process (clock, reset, start, boothlogic, sign, overflow1, overflow2)
        variable inputcounter: positive range 0 to NumInputs;
        variable bitcounter: positive range 0 to BitSize;
    begin
        if reset = '0' then
            present_state <= Sidle;
        elsif clock'EVENT and clock = '1' then
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

case present\_state is

```
    when Sidle =>
        done <= '0';

        loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';
        loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
        selP <= "00"; selQ <= "00";
        clearPF <= '1'; opMode <= '0';
        selinputs <= '0'; selweights <= '0'; selout <= '0';
        inputcounter := NumInputs;
        if start = '1' then
            present_state <= Sidle;

        else
            present_state <= Sinitmult;
        end if;

    when Sinitmult =>
        done <= '0';

        loadM <= '1'; loadP <= '0'; loadQ <= '1'; loadF <= '0';
        loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
        selP <= "00"; selQ <= "00";
        clearPF <= '0'; opMode <= '0';
        selout <= '0';
        bitcounter := BitSize;
        if inputcounter /= 2 then -- Is not generic. Has to be fixed !
            selinputs <= '1'; selweights <= '1';
        end if;
        present_state <= Swait;

        when Swait =>
            loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';
            loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
            present_state <= Sbooth;

    when Sbooth =>
        done <= '0';

        loadM <= '0'; loadQ <= '0'; loadF <= '0';
        loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
        selP <= "00"; selQ <= "00";
        clearPF <= '1';
        selout <= '0';
        if boothlogic = "10" then
            loadP <= '1';
            opMode <= '1';
        elsif boothlogic = "01" then
            loadP <= '1';
            opMode <= '0';
        else
            loadP <= '0';
        end if;
        present_state <= SloadP;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

when Sloadp =>

```
loadP <= '0';  
present_state <= Sshiftright;
```

when Sshiftright =>

```
done <= '0';  
loadM <= '0'; loadP <= '1'; loadQ <= '1'; loadF <= '1';  
loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';  
selP <= "11"; selQ <= "01";  
clearPF <= '1'; selout <= '0';  
opMode <= '0';  
bitcounter := bitcounter - 1 ;  
if bitcounter > 0 then  
  
    present_state <= Swait;--Sbooth;  
else  
    present_state <= Swaitadd;  
end if;
```

when Swaitadd =>

```
loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';  
loadOV <= '0' ; loadOUT <= '0'; loadSign <= '0';  
present_state <= Sadd;
```

when Sadd =>

done <= '0';

```
loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';  
loadOV <= '1' ; loadOUT <= '0'; loadSign <= '0';  
selP <= "00"; selQ <= "00";  
clearPF <= '1'; opMode <= '0';  
selout <= '0';  
inputcounter := inputcounter -1;  
if inputcounter > 0 then  
    present_state <= Sinitmult;  
  
else  
    present_state <= Swaitsign;  
end if;
```

when Swaitsign =>

```
loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';  
loadOV <= '0' ; loadOUT <= '0'; loadSign <= '0';  
present_state <= Ssign;
```

when Ssign =>

done <= '0';

```
loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';  
loadOV <= '0'; loadOUT <= '0'; loadSign <= '1' ;  
selP <= "00"; selQ <= "00";  
clearPF <= '1'; opMode <= '0';  
present_state <= Swaitfinal;
```

when Swaitfinal =>

```
loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';  
loadOV <= '0' ; loadOUT <= '0'; loadSign <= '0';  
present_state <= Sfinal;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
when Sfinal =>
    done <= '0';

    loadM <= '0'; loadP <= '0'; loadQ <= '0'; loadF <= '0';
    loadOV <= '0'; loadOUT <= '1'; loadSign <= '0';
    selP <= "00"; selQ <= "00";
    clearPF <= '0'; opMode <= '0';
    if sign = '1' then
        selout <= '0';
    else
        selout <= '1';
    end if;
    present_state <= Sdone;

when Sdone =>
    loadOUT <= '0';
    done <= '1';
    present_state <= Sidle;

end case;
end if;
end process comb_logic;
end architecture behavioral;
```

### C.4.2.2 - Neuron – Prototype2

-----  
-- Neuron implemented in hardware with booth multiplier  
-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : January 31, 2002  
-- File Name : neuron.vhd  
-- Architecture : structural  
-- Description : Neuron made of Datapath and State Machine.  
-----

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.nn_pack.all;

entity neuron2 is
generic(
    BitSize : positive := 8;
    NumInputs : positive := 2);
port(
    In1, In2 : in std_logic_vector( BitSize - 1 downto 0);
    We1, We2 : in std_logic_vector( BitSize - 1 downto 0);
    start, reset, clock : in std_logic;
    result: out std_logic_vector(BitSize - 1 downto 0);
    done, overflow : out std_logic);
end neuron2;

architecture structural of neuron2 is

    signal loadmwire, loadqwire, loadovwire, loadoutwire, loadnvwire, loadsignwire : std_logic;
    signal selinputswire, selweightswire, seloutwire, clearpfwire, ovf2wire : std_logic;
    signal signwire: std_logic;

begin

    statemachine: controlneuron2
        generic map ( NumInputs, BitSize)
        port map (
            clock => clock,
            reset => reset,
            start => start,
            done => done,
            loadM => loadmwire,
            loadQ => loadqwire,
            loadNV => loadnvwire,
            loadOV => loadovwire,
            loadOUT => loadoutwire,
            loadSign => loadsignwire,
            selinputs => selinputswire,
            selweights => selweightswire,
            selout => seloutwire,
            sign => signwire,
            overflow2 => ovf2wire
        );
        overflow <= ovf2wire;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
datapath: neurondatapath2
generic map (BitSize)
port map(
    In1 => In1,
    In2 => In2,
    We1 => We1,
    We2 => We2,
    loadM => loadmwire,
    loadQ => loadqwire,
    loadNV => loadnvwire,
    loadOV => loadovwire,
    loadOUT => loadoutwire,
    loadSign => loadsignwire,
    selinputs => selinputswire,
    selweights => selweightswire,
    selout => seloutwire,
    clock => clock,
    reset => reset,
    output => result,
    sign => signwire,
    overflow2 => ovf2wire);

end architecture structural;
```

### C.4.2.2.1 – Component Datapath

-----

-- Neuron component Datapath - Second Model

-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : January 31, 2002  
-- File Name : neurondatapath.vhd  
-- Architecture : structural

-----

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

library work;

use work.nn\_pack.all;

entity neurondatapath2 is

generic( BitSize : positive := 8);

port(

    In1, In2 : in std\_logic\_vector( BitSize-1 downto 0);

    We1, We2 : in std\_logic\_vector( BitSize-1 downto 0);

    loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : in std\_logic;

    selinputs, selweights, selout : in std\_logic;

    clock : in std\_logic;

    reset : in std\_logic;

    output: out std\_logic\_vector( BitSize-1 downto 0);

    sign, overflow2 : out std\_logic);

end neurondatapath2;

architecture structural of neurondatapath2 is

    signal mwires, qwires, muxtomwires, muxtoqwires, muxtooutwires, minusone, one : std\_logic\_vector( BitSize-1 downto 0);

    signal resultwires: std\_logic\_vector( (2\*BitSize) + 1 downto 0); -- It has to be 18 bits to avoid overflow

    signal additionwires, ovwires, nvwires: std\_logic\_vector( (2\*BitSize) + 1 downto 0); -- They have to be 18 bits for addition to avoid overflow

    signal ground, coutwire : std\_logic;

    signal zeros : std\_logic\_vector(BitSize-1 downto 1);

    signal donothing : std\_logic\_vector( 3 downto 0);

begin

    ground <= '0';

    registerM : registerN

        generic map(BitSize)

        port map (     Input => muxtomwires,  
                    Load   =>loadM,  
                    Clock   => clock,  
                    Reset   => reset,  
                    Output => mwires);

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
muxtoM: mux2N_N
  generic map (BitSize)
  port map (
    Input0 => In1,
    Input1 => In2,
    Sel => selinputs,
    Output => muxtomwires);

muxtoQ: mux2N_N
  generic map (BitSize)
  port map (
    Input0 => We1,
    Input1 => We2,
    Sel => selweights,
    Output => muxtoqwires);

registerQ: registerN
  generic map (BitSize)
  port map (
    Input => muxtoqwires,
    Load => loadQ,
    Clock => clock,
    Reset => reset,
    Output => qwires);

multiplier: mulp8
  port map (
    a => mwires,
    b => qwires,
    prod => resultwires((2*BitSize) - 1 downto 0));

resultwires((2*BitSize) + 1) <= resultwires(BitSize - 1);
resultwires( 2*BitSize) <= resultwires(BitSize - 1);

regnewvalue : registerN
  generic map (18)--(2*BitSize + 2)
  port map (
    Input => resultwires,
    Load => loadNV,
    Clock => clock,
    Reset => reset,
    Output => nvwires);

regoldvalue : registerN
  generic map (18)--(2*BitSize + 2)
  port map (
    Input => additionwires,
    Load => loadOV,
    Clock => clock,
    Reset => reset,
    Output => ovwires);

fastadder : adder18bits
  port map (
    InputA => nvwires,
    InputB => ovwires,
    cin => ground,
    cout => overflow2,
    sum => additionwires);

dflipflopSign: dflipflop
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
port map (
    Input => ovwires((2*BitSize) + 1),
    Load => loadSign,
    clk    => clock,
    Reset  => reset,
    Output => sign); -- Takes last bit to see sign

minusone <= (others => '1');
zeros <= (others => '0');
one <= zeros & '1';

muxtosign: mux2N_N
    generic map (BitSize)
    port map (
        Input0 => minusone,
        Input1 => one,
        Sel => selout,
        Output => muxtooutwires
    );

registerOUT : registerN
    generic map (BitSize)
    port map (
        Input => muxtooutwires,
        Load  => loadOUT,
        Clock  => clock,
        Reset  => reset,
        Output => output
    );

end architecture structural;
```

### C.4.2.2.2 – Component Control Unit

```
-----  
-- Neuron control unit  
-- Author       : Guillermo Barreiro  
-- Student ID   : 1042071  
-- Date        : January 31, 2002  
-- File Name    : controlneuron.vhd  
-- Architecture : Behavioural  
-- Description  : State machine with asynthonous reset.  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity controlneuron2 is  
    generic( NumInputs : positive := 2;  
             BitSize : positive := 8);  
    port( clock : in std_logic;  
          reset : in std_logic;  
          start : in std_logic;  
          done : out std_logic;  
          loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : out std_logic;  
          selinputs, selweights, selout: out std_logic;  
          sign, overflow2: in std_logic);  
end controlneuron2;  
  
architecture behavioral of controlneuron2 is  
    -- Declaration of states  
    type state_type is (Sidle, Sinitmult, Sholdmult, Sadd, Ssign, Sfinal);  
    signal present_state: state_type;  
begin  
    -- Create a process that handles the state transition  
    comb_logic : process (clock, reset, start, sign, overflow2)  
        variable inputcounter: positive range 0 to NumInputs;  
    begin  
        if reset = '0' then  
            present_state <= Sidle;  
        elsif clock'EVENT and clock = '1' then  
  
            case present_state is  
  
                when Sidle =>  
                    done <= '0';  
                    loadM <= '0'; loadQ <= '0';  
                    loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';  
                    selinputs <= '0'; selweights <= '0'; selout <= '0';  
                    inputcounter := NumInputs;  
                    if start = '1' then  
                        present_state <= Sidle;  
                    else  
                        present_state <= Sinitmult;  
                    end if;  
                end case;  
            end if;  
        end if;  
    end process;  
end architecture;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
when Sinitmult =>
done <= '0';

loadM <= '1'; loadQ <= '1';
loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
selout <= '0';
if inputcounter /= 2 then -- Is not generic. Has to be fixed !
selinputs <= '1'; selweights <= '1';
end if;
present_state <= Sholdmult;

when Sholdmult =>
done <= '0';

loadM <= '0'; loadQ <= '0';
loadNV <= '1'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
selout <= '0';
present_state <= Sadd;

when Sadd =>
done <= '0';

loadM <= '0'; loadQ <= '0';
loadNV <= '0'; loadOV <= '1'; loadOUT <= '0'; loadSign <= '0';
selout <= '0';
inputcounter := inputcounter -1;
if inputcounter > 0 then
present_state <= Sinitmult;

else
present_state <= Ssign;
end if;

when Ssign =>
done <= '0';

loadM <= '0'; loadQ <= '0';
loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '1';
selout <= '0';
present_state <= Sfinal;

when Sfinal =>
done <= '1';

loadM <= '0'; loadQ <= '0';
loadNV <= '0'; loadOV <= '0'; loadOUT <= '1'; loadSign <= '0';
if sign = '1' then
selout <= '0';
else
selout <= '1';
end if;
present_state <= Sidle;

end case;
end if;

end process comb_logic;
end architecture behavioral;
```

### C.4.2.3 - Neuron – Prototype 3

-----

-- Neuron implemented in hardware with lpm\_mult megafunction as multiplier  
-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : February 5, 2001  
-- File Name : neuron3.vhd  
-- Architecture : structural  
-- Description : Neuron made of Datapath and State Machine.

-----

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.nn_pack.all;

entity neuron3 is
generic( BitSize : positive := 8;
        NumInputs : positive := 2);
port(
    In1, In2 : in std_logic_vector( BitSize - 1 downto 0);
    We1, We2 : in std_logic_vector( BitSize - 1 downto 0);
    start, reset, clock : in std_logic;
    result: out std_logic_vector(BitSize - 1 downto 0);
    done, overflow : out std_logic);
end neuron3;

architecture structural of neuron3 is

signal loadmwire, loadqwire, loadovwire, loadoutwire, loadnvwire, loadsignwire : std_logic;
signal selinputswire, selweightswire, seloutwire, clearpfwire, ovf2wire : std_logic;
signal signwire: std_logic;

begin

statemachine: controlneuron6
    generic map ( NumInputs, BitSize)
    port map ( clock => clock,
               reset => reset,
               start => start,
               done => done,
               loadM => loadmwire,
               loadQ => loadqwire,
               loadNV => loadnvwire,
               loadOV => loadovwire,
               loadOUT => loadoutwire,
               loadSign => loadsignwire,
               selinputs => selinputswire,
               selweights => selweightswire,
               selout => seloutwire,
               sign => signwire,
               overflow2 => ovf2wire);
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
overflow <= ovf2wire;
```

```
datapath: neurondatapath3
```

```
generic map (BitSize)
```

```
port map(
```

```
    In1 => In1,
```

```
    In2 => In2,
```

```
    We1 => We1,
```

```
    We2 => We2,
```

```
    loadM => loadmwire,
```

```
    loadQ => loadqwire,
```

```
    loadNV => loadnvwire,
```

```
    loadOV => loadovwire,
```

```
    loadOUT => loadoutwire,
```

```
    loadSign => loadsignwire,
```

```
    selinputs => selinputswire,
```

```
    selweights => selweightswire,
```

```
    selout => seloutwire,
```

```
    clock => clock,
```

```
    reset => reset,
```

```
    output => result,
```

```
    sign => signwire,
```

```
    overflow2 => ovf2wire);
```

```
end architecture structural;
```

### C.4.2.3.1 – Component Datapath

---

```
-- Neuron component Datapath - Third Model
-- Author      : Guillermo Barreiro
-- Student ID   : 1042071
-- Date        : February 5, 2002
-- File Name    : neurondatapath3.vhd
-- Architecture : structural
```

---

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.nn_pack.all;
library lpm;
use lpm.lpm_components.all;
```

```
entity neurondatapath3 is
generic( BitSize : positive := 8);
port(
    In1, In2 : in std_logic_vector( BitSize-1 downto 0);
    We1, We2 : in std_logic_vector( BitSize-1 downto 0);
    loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : in std_logic;
    selinputs, selweights, selout : in std_logic;
    clock : in std_logic;
    reset : in std_logic;
    output : out std_logic_vector( BitSize-1 downto 0);
    sign, overflow2 : out std_logic);
end neurondatapath3;
```

architecture structural of neurondatapath3 is

```
signal mwires, qwires, muxtomwires, muxtoqwires, muxtooutwires, minusone, one : std_logic_vector(
    BitSize-1 downto 0);
signal resultwires : std_logic_vector( (2*BitSize) + 1 downto 0); -- It has to be 18 bits to avoid overflow
signal additionwires, ovwires, nvwires : std_logic_vector( (2*BitSize) + 1 downto 0); -- They have to be 18
bits for addition to avoid overflow
signal ground, coutwire, notreset : std_logic;
signal zeros : std_logic_vector(BitSize-1 downto 1);
signal donothing : std_logic_vector( 3 downto 0);
```

begin

```
ground <= '0';
notreset <= not reset;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
registerM : registerN
    generic map(BitSize)
    port map (      Input => muxtomwires,
                    Load  => loadM,
                    Clock  => clock,
                    Reset  => reset,
                    Output => mwires);

muxtoM: mux2N_N
    generic map (BitSize)
    port map (
        Input0 => In1,
        Input1 => In2,
        Sel => selinputs,
        Output => muxtomwires);

muxtoQ: mux2N_N
    generic map (BitSize)
    port map (
        Input0 => We1,
        Input1 => We2,
        Sel => selweights,
        Output => muxtoqwires);

registerQ: registerN
    generic map (BitSize)
    port map (      Input => muxtoqwires,
                    Load  => loadQ,
                    Clock  => clock,
                    Reset  => reset,
                    Output => qwires);

multiplier : lpm_mult
    GENERIC map( LPM_WIDTHA => BitSize,
                 LPM_WIDTHB => BitSize,
                 LPM_WIDTHHP => 2*BitSize,
                 LPM_WIDTHHS => BitSize,
                 LPM_REPRESENTATION => "SIGNED",
                 LPM_PIPELINE => 1)
    PORT map (dataa => qwires,
               datab => mwires,
               result => resultwires((2*BitSize) - 1 downto 0),
               aclr => notreset,
               clock => clock);

resultwires((2*BitSize) + 1) <= resultwires(BitSize - 1);
resultwires( 2*BitSize) <= resultwires(BitSize - 1);

regnewvalue : registerN
    generic map (18) -- (2*BitSize + 2)
    port map (      Input => resultwires,
                    Load  => loadNV,
                    Clock  => clock,
                    Reset  => reset,
                    Output => nvwires);
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
regoldvalue : registerN
    generic map (18)--(2*BitSize + 2)

port map (      Input => additionwires,
                Load  => loadOV,
                Clock  => clock,
                Reset  => reset,
                Output => ovwires);

fastadder : adder18bits
    port map (
        InputA => nvwires,
        InputB => ovwires,
        cin => ground,
        cout => overflow2,
        sum => additionwires
    );

dflipflopSign: dflipflop
    port map (Input => ovwires((2*BitSize) + 1),
        Load => loadSign,
            clk      => clock,
            Reset => reset,
        Output => sign); -- Takes last bit to see sign

minusone <= (others => '1');
zeros <= ( others => '0');
one <= zeros & '1';

muxtosign: mux2N_N
    generic map (BitSize)
    port map (
        Input0 => minusone,
        Input1 => one,
        Sel => selout,
        Output => muxtooutwires
    );

registerOUT : registerN
    generic map (BitSize)

port map (      Input => muxtooutwires,
                Load  => loadOUT,
                Clock  => clock,
                Reset  => reset,
                Output => output
    );

end architecture structural;
```



### C.4.2.3.2 – Component Control Unit

```
-----  
-- Neuron control unit  
-- Author       : Guillermo Barreiro  
-- Student ID   : 1042071  
-- Date        : January 31, 2002  
-- File Name    : controlneuron6.vhd  
-- Architecture : Behavioural  
-- Description  : State machine with asynthonous reset.  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity controlneuron6 is  
    generic( NumInputs : positive := 2;  
             BitSize : positive := 8);  
    port( clock : in std_logic;  
          reset : in std_logic;  
          start : in std_logic;  
          done : out std_logic;  
          loadM, loadQ, loadNV, loadOV, loadOUT, loadSign : out std_logic;  
          selinputs, selweights, selout: out std_logic;  
          sign, overflow2: in std_logic;  
          clear_accum: out std_logic);  
end controlneuron6;
```

```
architecture behavioral of controlneuron6 is  
    -- Declaration of states  
    type state_type is (Sidle, Sinitmult, Sholdmult, Sadd, Ssign, Sfinal, Swaitmult, Swaitadd, Swaitsign,  
                        Swaitfinal, Sdone);  
    signal present_state: state_type;  
begin  
    -- Create a process that handles the state transition  
    comb_logic : process (clock, reset, start, sign, overflow2)  
        variable inputcounter: positive range 0 to NumInputs;  
    begin  
        if reset = '0' then  
            present_state <= Sidle;  
        elsif clock'EVENT and clock = '1' then  
  
            case present_state is  
  
                when Sidle =>  
                    done <= '0';  
                    loadM <= '0'; loadQ <= '0'; clear_accum <= '0';  
                    loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';  
                    selinputs <= '0'; selweights <= '0'; selout <= '0';  
                    inputcounter := NumInputs;  
                    if start = '1' then  
                        present_state <= Sidle;  
                    else  
                        present_state <= Sinitmult;  
                    end if;  
            end case;  
        end if;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
when Sinitmult =>
    done <= '0'; clear_accum <= '1';
    loadM <= '1'; loadQ <= '1';
    loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
    selout <= '0';
    if inputcounter /= 2 then -- Is not generic. Has to be fixed !
        selinputs <= '1'; selweights <= '1';
    end if;
    present_state <= Swaitmult;

when Swaitmult =>
    loadM <= '0'; loadQ <= '0';
    present_state <= Sholdmult;

when Sholdmult =>
    done <= '0';
    loadM <= '0'; loadQ <= '0';
    loadNV <= '1'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '0';
    selout <= '0';
    present_state <= Sadd;

when Swaitadd =>
    loadNV <= '1';
    present_state <= Sadd;

when Sadd =>
    done <= '0';
    loadM <= '0'; loadQ <= '0';
    loadNV <= '0'; loadOV <= '1'; loadOUT <= '0'; loadSign <= '0';
    selout <= '0';
    inputcounter := inputcounter -1;
    if inputcounter > 0 then
        present_state <= Sinitmult;
    else
        present_state <= Swaitsign;
    end if;

when Swaitsign =>
    loadOV <= '0';
    present_state <= Ssign;

when Ssign =>
    done <= '0';
    loadM <= '0'; loadQ <= '0';
    loadNV <= '0'; loadOV <= '0'; loadOUT <= '0'; loadSign <= '1';
    selout <= '0';
    present_state <= Swaitfinal;

When Swaitfinal =>
    loadsign <= '0';
    present_state <= Sfinal;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
when Sfinal =>
    done <= '0';
    loadM <= '0'; loadQ <= '0';
    loadNV <= '0'; loadOV <= '0'; loadOUT <= '1'; loadSign <= '0';
    if sign = '1' then
        selout <= '0';
    else
        selout <= '1';
    end if;
    present_state <= Sdone;

when Sdone =>
    done <= '1';
    present_state <= Sidle;

    end case;
end if;
end process comb_logic;
end architecture behavioral;
```

### C.4.2.4 - Neuron – Prototype 4

---

-- Neuron implemented in hardware using behavioral design  
-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : February 5, 2001  
-- File Name : neuron4.vhd  
-- Architecture : structural  
-- Description : Neuron made of Datapath and State Machine.

---

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.nn_pack.all;

entity neuron4a is
generic( NumInpWeg : positive := 2);
port(
    In1, In2 : in integer range -127 to 127;
    We1, We2 : in integer range -127 to 127;
    start, reset, clock : in std_logic;
    result: out integer range -127 to 127;
    addition : out integer range -131071 to 131071;
    done : out std_logic);
end neuron4a;

architecture behavioral of neuron4a is
    signal resultdummy : integer range -131071 to 131071;
begin
    pr1 : process is
    begin
        wait until rising_edge(clock);
        if reset = '0' then
            result <= 0;
            done <= '0';
        elsif start = '0' then
            if resultdummy < 0 then
                result <= -1;
                done <= '1';
            else
                result <= 1;
                done <= '1';
            end if;
        end if;
        addition <= resultdummy;
    end process;

    pr2: process is
    begin
        resultdummy <= (In1*We1) + (In2*We2);
    end process;

end architecture behavioral;
```



---

## C.4.3 – Neural Network Datapath

-----  
-- Neural Network datapath - 3 hebbian Neurons connected in two layers (2-2-1)  
-- Bidirectional databus used. All datapath included in this design

-- Author : Guillermo Barreiro  
-- Student ID : 1042071  
-- Date : March 10, 2002  
-- File Name : nn\_dp\_2n.vhd  
-- Architecture : Structural  
-----

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
library work;  
use work.nn_pack.all;
```

```
entity nn_dp_2n is  
generic ( cmd_bitsize : positive := 2;  
          sel_bitsize : positive := 2;  
          data_bitsize: positive := 8);  
port ( cmd : in std_logic_vector ( cmd_bitsize-1 downto 0);  
      sel : in std_logic_vector ( sel_bitsize-1 downto 0);  
      addr : in std_logic_vector( 1 downto 0);  
      databus : in std_logic_vector ( data_bitsize -1 downto 0);  
      databusout : out std_logic_vector ( data_bitsize -1 downto 0);  
      In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);  
      clock, reset : in std_logic;  
      result : out std_logic_vector ( data_bitsize -1 downto 0);  
      done : out std_logic;  
      ack : out std_logic);  
end entity nn_dp_2n;
```

```
architecture mixed of nn_dp_2n is  
signal regw1_h1, regw2_h1, output_h1, mux1to3state : std_logic_vector( data_bitsize - 1  
downto 0);  
signal regw1_h2, regw2_h2, output_h2, mux2to3state : std_logic_vector( data_bitsize - 1  
downto 0);  
signal regw1_o1, regw2_o1, output_o1, mux3to3state, outtobus : std_logic_vector(  
data_bitsize - 1 downto 0);  
signal startw, start2w, ov1, ov2, ov3, addr0, addr1, addr2, done_H1, done_H2 : std_logic;  
signal cmd_wire, loadW1H1, loadW2H1, loadW1H2, loadW2H2, loadW1O1, loadW2O1,  
outbus : std_logic;  
signal ackw1h1,loadW1H1_dummy, ackw2h1,loadW2H1_dummy,  
ackw1h2,loadW1H2_dummy, ackw2h2,loadW2H2_dummy : std_logic;  
signal ackw1o1,loadW1o1_dummy, ackw2o1,loadW2o1_dummy, outbus_dummy : std_logic;  
begin
```

```
ack <= ackw1h1 or ackw2h1 or ackw1h2 or ackw2h2 or ackw1o1 or ackw2o1 or outbus;
```

--Hidden Neuron 1

-- cmd = "01" to write weight, Sel "00" is the weight to change

```
loadw1H1_dummy <= cmd(0) and not cmd(1) and addr0 and not sel(0) and not sel(1);
```

```
turn_off_load : process(clock, reset, loadw1H1_dummy)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            loadw1h1 <= '0';
        elsif falling_edge(clock) then
            if (loadw1h1_dummy = '1') and (flag = 0) then
                flag := 1;
                loadw1h1 <= '1';
            elsif (loadw1h1_dummy = '0') and (flag = 1) then
                flag := 0;
                loadw1h1 <= '0';
            else
                loadw1h1 <= '0';
            end if;
        end if;
    end process;
```

```
Hidden1_Weight1 : process(reset, clock, loadw1h1)
    begin
        if (reset = '0') then
            regw1_h1 <= (others => '0');
        elsif clock = '1' and clock'event then
            if (loadw1h1 = '1') then
                regw1_h1 <= databus;
                ackw1h1 <= '1';
            else
                ackw1h1 <= '0';
            end if;
        end if;
    end process;
```

```
-- cmd = "01" to write weight, Sel "01" is the weight to change
loadw2h1_dummy <= cmd(0) and not cmd(1) and addr0 and sel(0) and not sel(1);
```

```
turn_off_load2 : process(clock, reset, loadw2H1_dummy)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            loadw2h1 <= '0';
        elsif falling_edge(clock) then
            if (loadw2h1_dummy = '1') and (flag = 0) then
                flag := 1;
                loadw2h1 <= '1';
            elsif (loadw2h1_dummy = '0') and (flag = 1) then
                flag := 0;
                loadw2h1 <= '0';
            else
                loadw2h1 <= '0';
            end if;
        end if;
    end process;
```



```
Hidden1_Weight2 : process(reset,clock,loadw2h1)
begin
    if(reset = '0') then
        regw2_h1 <= (others => '0');
    elsif clock = '1' and clock'event then
        if (loadw2h1 = '1') then
            regw2_h1 <= databus;
            ackw2h1 <= '1';
        else
            ackw2h1 <= '0';
        end if;
    end if;
end process;

Hidden1_weight_output: mux2N_N
generic map (data_bitsize)
port map (
    Input0 => regw1_H1,
    Input1 => regw2_H1,
    Sel    => sel(0),
    Output => mux1to3state);

Fw_calculation_Hidden1 : neuron3
generic map( data_bitsize, 2)
port map( In1 => In1,
          In2 => In2,
          We1 => regw1_h1,
          We2 => regW2_h1,
          start => startW,
          reset => reset,
          clock => clock,
          result => output_H1,
          done => done_H1,
          overflow => ov1);

--Hidden Neuron 2
-- cmd = "01" to write weight, Sel "00" is the weight to change
loadw1H2_dummy <= cmd(0) and not cmd(1) and addr1 and not sel(0) and not sel(1);
turn_off_load3 : process(clock, reset, loadw1H2_dummy)
variable flag : integer range 0 to 1 := 0;
begin
    if (reset = '0') then
        flag := 0;
        loadw1h2 <= '0';
    elsif falling_edge(clock) then
        if (loadw1h2_dummy = '1') and (flag = 0) then
            flag := 1;
            loadw1h2 <= '1';
        elsif (loadw1h2_dummy = '0') and (flag = 1) then
            flag := 0;
            loadw1h2 <= '0';
        else
            loadw1h2 <= '0';
        end if;
    end if;
end process;
```

```
Hidden2_Weight1 : process(reset,clock,loadw1h2)
begin
    if(reset = '0') then
        regw1_h2 <= (others => '0');
    elsif clock = '1' and clock'event then
        if (loadw1h2 = '1') then
            regw1_h2 <= databus;
            ackw1h2 <= '1';
        else
            ackw1h2 <= '0';
        end if;
    end if;
end process;

-- cmd = "01" to write weight, Sel "01" is the weight to change
loadw2H2_dummy <= cmd(0) and not cmd(1) and addr1 and sel(0) and not sel(1);

turn_off_load4 : process(clock, reset, loadw2H2_dummy)
variable flag : integer range 0 to 1 := 0;
begin
    if (reset = '0') then
        flag := 0;
        loadw2h2 <= '0';
    elsif falling_edge(clock) then
        if (loadw2h2_dummy = '1') and (flag = 0) then
            flag := 1;
            loadw2h2 <= '1';
        elsif (loadw2h2_dummy = '0') and (flag = 1) then
            flag := 0;
            loadw2h2 <= '0';
        else
            loadw2h2 <= '0';
        end if;
    end if;
end process;

Hidden2_Weight2 : process(reset,clock,loadw2h2)
begin
    if(reset = '0') then
        regw2_h2 <= (others => '0');
    elsif clock = '1' and clock'event then
        if (loadw2h2 = '1') then
            regw2_h2 <= databus;
            ackw2h2 <= '1';
        else
            ackw2h2 <= '0';
        end if;
    end if;
end process;
```

```
Hidden2_weight_output: mux2N_N
    generic map (data_bitsize)
    port map (
        Input0 => regw1_H2,
        Input1 => regw2_H2,
        Sel    => sel(0),
        Output => mux2to3state);
Fw_calculation_Hidden2 : neuron3
    generic map( data_bitsize, 2)
    port map( In1 => In1,
        In2 => In2,
        We1 => regw1_h2,
        We2 => regW2_h2,
        start => startW,
        reset => reset,
        clock => clock,
        result => output_H2,
        done => done_H2,
        overflow => ov2);

--Output Neuron 1
-- cmd = "01" to write weight, Sel "00" is the weight to change
loadw1O1_dummy <= cmd(0) and not cmd(1) and addr2 and not sel(0) and not sel(1);

turn_off_load5 : process(clock, reset, loadw1o1_dummy)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            loadw1o1 <= '0';
        elsif falling_edge(clock) then
            if ( loadw1o1_dummy = '1') and (flag = 0) then
                flag := 1;
                loadw1o1 <= '1';
            elsif (loadw1o1_dummy = '0') and (flag = 1) then
                flag := 0;
                loadw1o1 <= '0';
            else
                loadw1o1 <= '0';
            end if;
        end if;
    end process;

Output1_Weight1 : process(reset,clock,loadw1o1)
    begin
        if(reset = '0') then
            regw1_o1 <= (others => '0');
        elsif clock = '1' and clock'event then
            if (loadw1o1 = '1') then
                regw1_o1 <= databus;
                ackw1o1 <= '1';
            else
                ackw1o1 <= '0';
            end if;
        end if;
    end process;
```

```

-- cmd = "01" to write weight, Sel "01" is the weight to change
loadw2O1_dummy <= cmd(0) and not cmd(1) and addr2 and sel(0) and not sel(1);
turn_off_load6 : process(clock, reset, loadw2o1_dummy)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            loadw2o1 <= '0';
        elsif falling_edge(clock) then
            if (loadw2o1_dummy = '1') and (flag = 0) then
                flag := 1;
                loadw2o1 <= '1';
            elsif (loadw2o1_dummy = '0') and (flag = 1) then
                flag := 0;
                loadw2o1 <= '0';
            else
                loadw2o1 <= '0';
            end if;
        end if;
    end process;

```

```

Output1_weight2 : process(reset, clock, loadw2o1)
    begin
        if(reset = '0') then
            regw2_o1 <= (others => '0');
        elsif clock = '1' and clock'event then
            if (loadw2o1 = '1') then
                regw2_o1 <= databus;
                ackw2o1 <= '1';
            else
                ackw2o1 <= '0';
            end if;
        end if;
    end process;

```

```

Out1_weight_output: mux2N_N
    generic map (data_bitsize)
    port map (
        Input0 => regw1_O1,
        Input1 => regw2_O1,
        Sel    => sel(0),
        Output => mux3to3state);

```

```

Fw_calculation_Out1 : neuron3
    generic map( data_bitsize, 2)
    port map( In1  => Output_H1,
              In2  => Output_H2,
              We1  => regw1_O1,
              We2  => regW2_O1,
              start => start2W,
              reset => reset,
              clock => clock,
              result => result,
              done  => done,
              overflow => ov3);

```

```
-- Bidirectional Bus logic
outbus_dummy <= not cmd(0) and not cmd(1);
turn_off_outbus : process(clock, reset, outbus_dummy)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            outbus <= '0';
        elsif falling_edge(clock) then
            if ( outbus_dummy = '1') and (flag = 0) then
                flag := 1;
                outbus <= '1';
            elsif (outbus_dummy = '0') and (flag = 1) then
                flag := 0;
                outbus <= '0';
            else
                outbus <= '0';
            end if;
        end if;
    end process;

x_tristate_out : process( outtobus, outbus )
    begin
        if outbus = '1' then
            databusout <= outtobus;
        else
            databusout <= (others => '0');
        end if;
    end process x_tristate_out;

addr_writer : process (addr, cmd)
    variable addrdummy : integer range 2 downto 0 := 0;
    begin
        if cmd = "01" then
            addrdummy := conv_integer(addr);
            case addrdummy IS
                WHEN 0 =>
                    addr0 <= '1';
                    addr1 <= '0';
                    addr2 <= '0';
                WHEN 1 =>
                    addr0 <= '0';
                    addr1 <= '1';
                    addr2 <= '0';
                WHEN 2 =>
                    addr0 <= '0';
                    addr1 <= '0';
                    addr2 <= '1';
                WHEN OTHERS =>
                    addr0 <= '0';
                    addr1 <= '0';
                    addr2 <= '0';
            END CASE;
        end if;
    end process;
```

```
addr_reader : process (addr)
variable addrdummy : integer range 2 downto 0 := 0;
begin
    addrdummy := conv_integer(addr);
    case addrdummy IS
        WHEN 0 =>
            outtobus <= mux1to3state;
        WHEN 1 =>
            outtobus <= mux2to3state;
        WHEN 2 =>
            outtobus <= mux3to3state;
        WHEN OTHERS =>
            outtobus <= (others => '0');
        END CASE;
    end process;

enable_outneuron : process ( done_H1, done_H2)
begin
    if done_H1 = '1' and done_H2 = '1' then
        start2w <= '0'; -- Active low
    else
        start2w <= '1';
    end if;
end process;

cmd_wire <= cmd(0) and cmd(1);

enable_hiddenneurons: process (cmd_wire, clock) is
variable output : integer range 0 to 1 := 0;
begin
    if reset = '0' then
        output := 0;
        startW <= '1';
    elsif falling_edge(clock) then
        if cmd_wire = '1' and output = 0 then
            output := 1;
            startW <= '0';
        elsif cmd_wire = '0' and output = 1 then
            output := 0;
            startw <= '1';
        else
            startW <= '1';
        end if;
    end if;
end process;
end mixed;
```

## Appendix C5 – VHDL code for Pseudo Random Number Generator

This component has no package attached.

-----

```
-- Pseudo Random Number Generator
-- Author      : Guillermo Barreiro
-- Date       : March 05, 2002
-- File Name  : prng.vhd.vhd
-- Architecture : Structural
-- Description : Divides input clock into several slower frequencies
```

-----

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
library work;
use work.nn_pack.all;

entity prng is
    generic(Width: positive := 8);
    port (
        clock: in std_logic;
        reset : in std_logic;
        output: out std_logic_vector(Width-1 downto 0)
    );
end entity prng;

architecture mixed of prng is
    signal clk5m, clk1m, clk100k, clk10k, clk1k, clk100, clk10, clk1 : std_logic;
    signal high, loadwire, load1, load2 : std_logic;
    signal zeros : std_logic_vector( Width-1 downto 1);
    signal one, counterwires : Std_logic_vector( Width-1 downto 0);
begin

    high <= '1';
    zeros <= (others => '0');
    one <= zeros & '1';
    loadwire <= load1 or load2;

    Clock_divisor: clk_div
        port map(
            clock_25Mhz => clock,
            clock_5MHZ => clk5m,
            clock_1MHz => clk1m,
            clock_100KHz => clk100k,
            clock_10KHz => clk10k,
            clock_1KHz => clk1k,
            clock_100Hz => clk100,
            clock_10Hz => clk10,
            clock_1Hz => clk1
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
);
Linear_feedback_Shregister : LFSR_GENERIC
    generic map (Width)
    port map (clock => clk5m,
               reset => reset,
               load => loadwire,
               enable => high ,
               parallel_in => counterwires,
               parallel_out => output);

pr1: process is
    begin
        wait until rising_edge(clock);
        if reset = '0' then
            counterwires <= (others => '0');
            load1 <= '0';
        else
            counterwires <= counterwires + one;
        end if;
    end process;

pr2: process is
    variable counter : positive range 0 to Width-1;
    variable flag : boolean := true;
    begin
        wait until rising_edge(clock);
        if reset = '0' then
            counter := 0;
            flag := true;
        else
            counter := counter + 1;
        end if;
        if Counterwires(counter) = '1' then
            load2 <= '1';
        else
            load2 <= '0';
        end if;
    end process;

end mixed;
```



## Appendix C6 – VHDL code for Error Calculating Module

```
--Error Calculator

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity error_calculator is
    generic ( bitsize : positive := 8);
port ( Target, Outputs_NN : in std_logic_vector ( bitsize - 1 downto 0);
      NN_done, Training, calc_error, reset, clock : in std_logic;
      Done_error_out, Result : out std_logic);
end entity error_calculator;

architecture behavioral of error_calculator is

    signal Error_stored : integer range 0 to 255;
    signal done_error : std_logic;

begin

    error_calculator : process (clock, Nn_done, reset, calc_error) is
        variable Error_accu: Integer range 0 to 255;
    begin
        if reset = '0' then
            Error_accu := 0;
            Error_stored <= 255; -- Maximum number of training points
            Done_error <= '0';
            Result <= '0';
        elsif rising_edge(clock) then
            if calc_error = '1' then
                if Error_accu < Error_stored then
                    result <= '1';
                    Error_stored <= Error_accu;
                else
                    result <= '0';
                end if;
                Error_accu := 0;
                done_error <= '1';

            elsif NN_done = '1' and training = '1' then
                if Outputs_NN = Target then
                    Error_accu := Error_accu;
                else
                    Error_accu := Error_accu + 1;
                end if;
                done_error <= '0';
            end if;
        end if;
    end process;

    Result <= '0';

    Done_error_out <= done_error;

end architecture;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
turn_off_done: process (clock, reset, done_error)
    variable flag : integer range 0 to 1 := 0;
    begin
        if (reset = '0') then
            flag := 0;
            done_error_out <= '0';
        elsif falling_edge(clock) then
            if (done_error = '1') and (flag = 0) then --write_latch
                flag := 1;
                done_error_out <= '1';
            elsif (done_error = '0') and (flag = 1) then
                flag := 0;
                done_error_out <= '0';
            else
                done_error_out <= '0';
            end if;
        end if;
    end process;

end behavioral;
```

## Appendix C7 – VHDL code Control Unit Package

### Trainer\_control entity description

Cmd\_word - this is the first byte of any frame received by the rs232, which specifies the type of action to be taken by the control unit

Last\_weight - when the counters for weights and neurons in the bus interface unit are about to roll over to zero, this line is set high. This is used when sending weights to determine that the machine is done.

Transmit\_done - this is a line from the rs232 transmission datapath that informs the controller that the transmission of the last frame is has been completed.

Ack - bus line that neurons use to indicate acknowledgement of bus commands.

New\_data - line from rs232 in that informs the control unit that it has received a new packet from the computer.

Cntr\_increment - commands the bus interface to increment the weight counter (and address counter if necessary)

Go - commands the neural net to calculate

Busy - transmitted back to the computer via one of the serial control lines to indicated the machine is working.

Clear - commands the bus interface to zero it's internal counters

Read\_rand - commands the bus interface to read in a random weight from the random number generator.

Assert\_rand - commands the bus interface to assert the last read random value on the data lines of the bus.

Assert\_data - commands the bus interface to assert inputs from a data packet to the data lines of the bus.

Trn\_counter - when training, this is used to determine whether the training counter should be set to 1 or the value contained in a data packet

Save\_to\_RAM - tells the memory interface unit to save the current data set to RAM

Sendframe - tells the rs232 interface to dump the current transmission frame  
**into the rs232 transmit buffer.**

### C.7.1 Package: Control Unit

-----  
-- Control Unit Package  
-----

-----  
-- Component declarations for the top level data path.  
-----

library ieee;

use ieee.std\_logic\_1164.all;

package control\_pkg is

component busmaster\_2 is

    generic ( datawidth : positive := 8;  
              halfwidth : positive := 2);  
    port ( clock : in std\_logic;  
          read\_rand: in std\_logic;  
          read\_data: in std\_logic;  
          assert\_rand: in std\_logic;  
          assert\_data: in std\_logic;  
          assert\_saved : in std\_logic;  
          load\_addr: in std\_logic;  
          clear: in std\_logic;  
          in\_sel: in std\_logic\_vector(halfwidth - 1 downto 0);  
          incr\_sel : in std\_logic;  
          load\_sel : in std\_logic;  
          datain: in std\_logic\_vector(datawidth - 1 downto 0);  
          dataout: out std\_logic\_vector(datawidth - 1 downto 0);  
          addr: out std\_logic\_vector(halfwidth - 1 downto 0);  
          sel: out std\_logic\_vector(halfwidth - 1 downto 0);  
          in\_rng: in std\_logic\_vector(datawidth - 1 downto 0);  
          in\_data: in std\_logic\_vector(datawidth - 1 downto 0);  
          in\_addr: in std\_logic\_vector(halfwidth - 1 downto 0);  
          done: out std\_logic;  
          stored\_data: out std\_logic\_vector(datawidth - 1 downto 0);  
          do\_nothing : out std\_logic);  
end component;

-----  
component trainer\_control is

    port(  
        cmd\_word: in std\_logic\_vector(7 downto 0);  
        activedata: in std\_logic\_vector(15 downto 0);  
        clock, reset, last\_weight, ack, fifo\_empty: in std\_logic;  
        done, ce\_done, err\_result, tb\_full: in std\_logic;  
        calc\_err, load\_data, load\_from\_ram: out std\_logic;  
        bus\_cmd: out std\_logic\_vector(1 downto 0);  
        cntr\_increment, busy, clear, read\_rand: out std\_logic;  
        assert\_rand, assert\_data, data\_ack: out std\_logic;  
        save\_to\_ram, send\_frame, assert\_stored: out std\_logic;  
        load\_address, mem\_reset, frame\_type: out std\_logic;  
        save\_bus\_value, training\_mode: out std\_logic  
    );  
end component;

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

component busmaster is

```
generic ( datawidth : positive := 8;
          halfwidth : positive := 2);
port ( clock : in std_logic;
       read_rand: in std_logic;
       read_data: in std_logic;
       assert_rand: in std_logic;
       assert_data: in std_logic;
       assert_saved : in std_logic;
       load_addr: in std_logic;
       clear: in std_logic;
       in_sel: in std_logic vector(halfwidth - 1 downto 0);
       incr_sel : in std_logic;
       load_sel : in std_logic;
       data: inout std_logic vector(datawidth - 1 downto 0);
       addr: out std_logic vector(halfwidth - 1 downto 0);
       sel: out std_logic vector(halfwidth - 1 downto 0);
       in_rng: in std_logic vector(datawidth - 1 downto 0);
       in_data: in std_logic vector(datawidth - 1 downto 0);
       in_addr: in std_logic vector (halfwidth -1 downto 0);
       done: out std_logic;
       stored_data: out std_logic vector(datawidth -1 downto 0);
       do_nothing : out std_logic);
end component;
```

```
constant ram_data_width : positive := 24;           -- width of data in ram
constant ram_address_width : positive := 8;          -- width of address in ram
constant ram_data_size : positive := 256;           -- number of values in ram
--subtype datapath is std_logic vector(data_width-1 downto 0);
```

COMPONENT ram IS

```
PORT( ram_data in : in std_logic vector(ram_data_width-1 downto 0);           -- input data
      write : in std_logic;
      -- determines write
      read : in std_logic;
      -- determines read
      clock : in std_logic;
      -- synchronized with system clock
reset : in std_logic;
      ram_data_out : out std_logic vector(ram_data_width-1 downto 0); -- output data
      busy : out std_logic
      --r add : out std_logic vector(ram_address_width -1 downto 0); -- output data
      --w add : out std_logic vector(ram_address_width -1 downto 0);
      --add : out std_logic vector(ram_address_width -1 downto 0)
);
END COMPONENT ram;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
_____  
component prng is  
_____  
    generic(Width: positive := 8);  
_____  
    port (      clock: in std_logic;  
_____  
            reset : in std_logic;  
_____  
            output: out std_logic_vector(Width-1 downto 0));  
_____  
end component;  
  
_____  
component calculate_error is  
_____  
    generic(errorsize : positive := 8);  
_____  
    port (      clock, reset : in std_logic;  
_____  
            calc_error_in : in std_logic;  
_____  
            result : in std_logic_vector(errorsize-1 downto 0);  
_____  
            expected : in std_logic_vector(errorsize - 1 downto 0);  
_____  
            calc_done : out std_logic;  
_____  
            calc_error_out : out std_logic;  
_____  
            absol : out std_logic_vector(errorsize - 1 downto 0)  
_____  
            );  
_____  
end component;  
  
_____  
component error_calculator is  
_____  
    generic ( bitsize : positive := 8);  
_____  
    port (      Target, Outputs_NN : in std_logic_vector ( bitsize - 1 downto 0);  
_____  
            NN_done, Training, calc_error, reset, clock : in std_logic;  
_____  
            Done_error_out, Result : out std_logic  
_____  
            );  
_____  
end component;  
  
end package;
```

### C.7.2 Component: Control Unit

```
-----
--
-- Control Unit
--
-----
-- Top level block wiring the control unit together.
-- Suitable for wiring in with the RS232/NN and for
-- Children ages 3 and older.
-----

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.control_pkg.all;

entity control_unit2 is
generic (BUS_ADDR_WIDTH: natural := 2;
         BUS_SEL_WIDTH: natural := 2;
         BUS_DATA_WIDTH: natural := 8;
         DATA_WIDTH: natural := 8
        );
port (
    -- Trainer Bus Signals
    bus_addr: out std_logic_vector(BUS_ADDR_WIDTH-1 downto 0);
    bus_sel: out std_logic_vector(BUS_SEL_WIDTH-1 downto 0);
    bus_data_in: in std_logic_vector(BUS_DATA_WIDTH-1 downto 0);
    bus_data_out: out std_logic_vector(BUS_DATA_WIDTH-1 downto 0);
    input1: out std_logic_vector(BUS_DATA_WIDTH-1 downto 0);
    input2: out std_logic_vector(BUS_DATA_WIDTH-1 downto 0);
    bus_cmd: out std_logic_vector(1 downto 0);
    ack, done: in std_logic;
    results: in std_logic_vector(DATA_WIDTH-1 downto 0);
    -- TESTING SIGNALS
    bo_diddly: out std_logic;
    bs_ce_done : out std_logic;
    bs_calc_error : out std_logic;
    -- RS232 Interface Signals
    data_in: in std_logic_vector(31 downto 0);
    fifo_empty, tb_full: in std_logic;
    data_acknowledge: out std_logic;
    data_out: out std_logic_vector(15 downto 0);
    send_frame: out std_logic;
    -- Miscellaneous
    reset, clock: in std_logic
);
end entity control_unit2;

architecture structural of control_unit2 is

    signal dataset: std_logic_vector(31 downto 0);
    signal do_nothing, load_data, last_weight, calc_err, cntr_increment: std_logic;
    signal clear, read_rand, assert_rand, assert_data: std_logic;
    signal save_to_ram, load_address, mem_reset, assert_stored: std_logic;
    signal err_result, ce_done, load_from_ram, go: std_logic;
    signal random_thing, stored_data: std_logic_vector(7 downto 0);
    signal error_magnitude: std_logic_vector(7 downto 0);
    signal ram_signal: std_logic_vector(23 downto 0);
    signal bus_addr_dummy : std_logic_vector(BUS_SEL_WIDTH-1 downto 0);
    signal Reids_signal2, save_bus_value: std_logic;
    signal absolute_zero: std_logic_vector(7 downto 0);
    signal tbus_addr, tbus_sel: std_logic_vector(1 downto 0);
    signal training_mode: std_logic;
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
begin
bs_ce_done <= ce_done;
bs_calc_error <= calc_err;
absolute_zero <= (others => '0');
datastuff: process(clock)
begin
if (clock'event and clock = '1') then
if load_data = '1' then
dataset <= data_in;
elsif load_from_ram = '1' then
dataset(31 downto 8) <= ram_signal;
end if;
end if;
end process;

frame_select: process(clock)
begin
if Reids_signal2 = '0' then
data_out(7 downto 0) <= results;
data_out(15 downto 8) <= absolute_zero;
else
data_out(1 downto 0) <= tbus_addr;
data_out(3 downto 2) <= absolute_zero(1 downto 0);
data_out(5 downto 4) <= tbus_sel;
data_out(7 downto 6) <= absolute_zero(1 downto 0);
data_out(15 downto 8) <= stored_data;
end if;
end process;
input1 <= dataset(15 downto 8);
input2 <= dataset(23 downto 16);
bus_addr <= tbus_addr;
bus_sel <= tbus_sel;
cntrl_unit: trainer_control port map(
cmd_word => dataset(7 downto 0),
activedata => dataset(23 downto 8),
clock => clock,
reset => reset,
last_weight => last_weight,
ack => ack,
fifo_empty => fifo_empty,
done => done,
ce_done => ce_done,
tb_full => tb_full,
calc_err => calc_err,
err_result => err_result,
bus_cmd => bus_cmd,
cntr_increment => cntr_increment,
clear => clear,
read_rand => read_rand,
assert_rand => assert_rand,
assert_data => assert_data,
assert_stored => assert_stored,
save_to_ram => save_to_ram,
load_from_ram => load_from_ram,
load_address => load_address,
mem_reset => mem_reset,
data_ack => data_acknowledge,
load_data => load_data,
frame_type => Reids_signal2,
save_bus_value => save_bus_value,
send_frame => send_frame,
training_mode => training_mode
);
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
bus_master: busmaster_2
    generic map(datawidth => 8,
                halfwidth => 2)
    port map(
        clock => clock,
        read_rand => read_rand,
        read_data => save_bus_value,
        assert_rand => assert_rand,
        assert_data => assert_data,
        assert_saved => assert_stored,
        load_addr => load_address,
        load_sel => load_address,
        clear => clear,
        in_sel => dataset(12 downto 13), --(15 downto 12), Watch Out
        in_addr => dataset( 9 downto 8), --(11 downto 8),
        in_rng => random_thing,
        in_data => dataset(23 downto 16),
        incr_sel => cntr_increment,
        datain => bus_data_in,
        dataout => bus_data_out,
        addr => tbus_addr,
        sel => tbus_sel,
        done => last_weight,
        stored_data => stored_data,
        do_nothing => do_nothing
    );

Random_numbers: prng
    generic map (Width => 8)
    port map      (clock => clock,
                   reset => reset,
                   output => random_thing);

mem_unit: ram
    port map(
        ram_data_in => dataset(31 downto 8),
        write => save_to_ram,
        read => load_from_ram,
        clock => clock,
        reset => reset,
        ram_data_out => ram_signal
    );

error_calc: error_calculator
    port map( Target => dataset(31 downto 24),
              Outputs_NN => results,
              NN_done => done,
              Training => training_mode,
              calc_error => calc_err,
              reset => reset,
              clock => clock,
              Done_error_out => ce_done,
              Result => err_result
    );

end architecture structural;
```

## C.7.3 Trainer Control Path

```

-----
--
-- Trainer Control Path
--
-----
--
-- State machine governing control signals for the
-- neural network trainer.
--
-----

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    use ieee.std_logic_unsigned.all;

    entity trainer_control is
    generic( CMD_WORD_SIZE: natural := 8;
            CMD_BUS_SIZE: natural := 2;
            BUS_ADR_SIZE: natural := 2;
            BUS_SEL_SIZE: natural := 2;
            NUM_NEURONS: natural := 3;
            NUM_INPUTS: natural := 2
            );
    port(
        cmd_word: in std_logic_vector(7 downto 0);
        activedata: in std_logic_vector(15 downto 0);
        clock, reset, last_weight, ack, fifo_empty: in std_logic;
        done, ce_done, err_result, tb_full: in std_logic;
        calc_err, load_data, load_from_ram: out std_logic;
        bus_cmd: out std_logic_vector(1 downto 0);
        cntr_increment, busy, clear, read_rand: out std_logic;
        assert_rand, assert_data, data_ack: out std_logic;
        save_to_ram, send_frame, assert_stored: out std_logic;
        load_address, mem_reset, frame_type: out std_logic;
        save_bus_value, training_mode: out std_logic
    );
    end trainer_control;

    architecture RTL of trainer_control is
    type CNTRL_STATE is (STARTUP, INIT_RAND, IDLE, TRAIN, TRAIN_FROM_RAM, --4
        SEND_WEIGHTS_HOME, GET_WEIGHT, ASSEMBLE_FRAME, INCR_COUNTER, --8
        LOAD_WEIGHTS, EVALUATE_NN, CALC_ERROR, RESTORE, SET_WEIGHT, --D
        START_TRAINING_RUN, DONE_TRAINING_RUN, E_NN, SEND_RESULTS,
        WASTE_A_CLOCK, ALMOST_IDLE, IS_LAST);
    signal controlunit_state: CNTRL_STATE := STARTUP;

    -- some states are frequently used (get weight, set weight, go...) so
    -- this is used to keep track of which state the machine should go to after
    -- they leave one of these repeated states.

    signal next_state: CNTRL_STATE;
    signal train_counter: integer range 0 to 65535;
    signal attempt_counter: integer range 0 to 5;
    signal current_datum: integer range 1 to 255;
    signal datum_counter: integer range 0 to 255; -- change this to increase
    possible data size
    --signal data_load: std_logic;

```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
begin
statechange: process ( clock, reset )

begin

--load_data <= data_load;

    if reset = '0' then
        controlunit_state <= STARTUP;
    else if (clock'event and clock='1') then
        case controlunit_state is
            when STARTUP =>
                read_rand <= '1'; -- so we have a random weight next clock
                cntr_increment <= '0'; save_to_ram <= '0';
                assert_rand <= '0'; assert_data <= '0'; assert_stored <=
                    '0';
                calc_err <= '0'; send_frame <='0'; load_from_ram <= '0';
                bus_cmd <= "10"; clear <= '0'; -- set counters to 0 -- GBR
                changed value of clear
                training_mode <= '0'; frame_type <= '0';
                datum_counter <= 0; current_datum <= 1;
                mem_reset <= '1'; -- reset ram unit so it's happy.
                controlunit_state <= INIT_RAND;

            when INIT_RAND =>    read_rand <= '1';
                cntr_increment <= '1'; clear <= '1'; -- GBR  changed value
                of clear
                mem_reset <= '0';
                if last_weight = '1' then
                    controlunit_state <= ALMOST_IDLE; -- all weights set,
                    idle
                else
                    assert_rand <= '1';
                    next_state <= INIT_RAND;
                    controlunit_state <= SET_WEIGHT;
                end if;

-- Almost idle, makes sure that data gets loaded.  To revert to old version,
                remove ALMOST_IDLE,
-- and replace all references with IDLE.

                when ALMOST_IDLE =>
                    busy <= '0'; assert_rand <= '0'; assert_data <= '0';
                    cntr_increment <= '0'; assert_stored <= '0';
                    send_frame <= '0'; read_rand <= '0'; save_to_ram <= '0';
                    bus_cmd <= "10"; frame_type <= '0'; clear <= '1'; -- GBR
                    changed value of clear
                    load_data <= '1'; attempt_counter <= 0;
                    controlunit_state <= IDLE;

                    when IDLE =>
                        training_mode <= '0';
                        busy <= '0'; assert_rand <= '0'; assert_data <= '0';
                        cntr_increment <= '0'; assert_stored <= '0'; save_to_ram <=
                            '0';
                        send_frame <= '0'; read_rand <= '0';
                        bus_cmd <= "10"; frame_type <= '0'; clear <= '1'; -- GBR
                        changed value of clear
                        attempt_counter <= 0;
                        if fifo_empty = '0' then--and load_data = '1' then
                            load_data <= '0'; data_ack <= '1';
                            controlunit_state <= WASTE_A_CLOCK;
                        end if;
                    end if;
                end if;
            end case;
        end if;
    end if;
end;
```

```
when WASTE_A_CLOCK =>
  case cmd_word is
    when "00000001" =>
      controlunit_state <= TRAIN;
    when "00000010" =>
      controlunit_state <= TRAIN_FROM_RAM;
    when "00000100" =>
      controlunit_state <= SEND_WEIGHTS_HOME;
    when "00001000" =>
      controlunit_state <= LOAD_WEIGHTS;
    when "00010000" =>
      next_state <= SEND_RESULTS;
      controlunit_state <= EVALUATE_NN;
    when OTHERS =>
      controlunit_state <= ALMOST_IDLE;
    end case;
    data_ack <= '0';

    when TRAIN =>
      busy <= '1'; train_counter <= 1;
      training_mode <= '0';
      save_to_ram <= '1'; -- save stuff to ram
      datum_counter <= datum_counter + 1;
      controlunit_state <= ALMOST_IDLE;

    when TRAIN_FROM_RAM =>
      busy <= '1'; train_counter <= conv_integer(activatedata);
      current_datum <= 1;
      training_mode <= '1';
      next_state <= START_TRAINING_RUN;
      controlunit_state <= GET_WEIGHT;

    when START_TRAINING_RUN =>
      assert_rand <= '1';
      read_rand <= '1';
      next_state <= E_NN;
      controlunit_state <= SET_WEIGHT;

    when SEND_WEIGHTS_HOME =>
      frame_type <= '1';
      busy <= '1'; clear <= '0'; -- reset counters to start from
beginning -- GBR changed value of clear
      data_ack <= '0';
      next_state <= ASSEMBLE_FRAME;
      controlunit_state <= GET_WEIGHT;

    when GET_WEIGHT =>
      cntr_increment <= '0'; -- having that happen here would be
really bad
      clear <= '1'; -- GBR changed value of clear
      if ack = '0' then
        save_bus_value <= '1';
        bus_cmd <= "00"; -- read weight
        save_to_ram <= '0';
      else
        save_bus_value <= '0';
        bus_cmd <= "10";
        controlunit_state <= next_state;
        next_state <= ALMOST_IDLE; -- just in case.
      end if;
```

```
when ASSEMBLE_FRAME =>
  if tb_full = '0' then
    send_frame <= '1';
    bus_cmd <= "10";
    controlunit_state <= INCR_COUNTER;
  end if;

when INCR_COUNTER =>
  cntr_increment <= '1';
  send_frame <= '0';
  if(last_weight = '1') then
    controlunit_state <= ALMOST_IDLE;
  else
    next_state <= ASSEMBLE_FRAME;
    controlunit_state <= GET_WEIGHT;
  end if;

when LOAD_WEIGHTS =>
  busy <= '1'; data_ack <= '0';
  assert_data <= '1';
  load_address <= '1';
  next_state <= ALMOST_IDLE;
controlunit_state <= SET_WEIGHT;

when EVALUATE_NN =>
  load_from_ram <= '0';
  busy <= '1';
  data_ack <= '0';
  if done = '0' then
    bus_cmd <= "11";
  else
    bus_cmd <= "10";
  controlunit_state <= next_state;
  next_state <= ALMOST_IDLE;
  end if;

when CALC_ERROR =>
  if ce_done = '1' then
    calc_err <= '0';
    if err_result = '0' then
      controlunit_state <= RESTORE;
    else
      controlunit_state <= DONE_TRAINING_RUN;
    end if;
  else
    calc_err <= '1';
  end if;

when RESTORE => --load_data <= '0';
  assert_stored <= '1';
  next_state <= DONE_TRAINING_RUN;
  controlunit_state <= SET_WEIGHT;
```

```
when SET_WEIGHT =>
  cntr_increment <= '0';
  load_from_ram <= '0';
  if ack = '0' then
    load_address <= '0';
    read_rand <= '0';
    bus_cmd <= "01";
  else
    bus_cmd <= "10";
  end if;
assert_rand <= '0'; assert_data <= '0'; assert_stored
<= '0';
controlunit_state <= next_state;
end if;

when E_NN =>
  load_from_ram <= '1';
  next_state <= IS_LAST;
controlunit_state <= EVALUATE_NN;

when IS_LAST =>
if current_datum >= datum_counter then
  controlunit_state <= CALC_ERROR;
  current_datum <= 1;
else
  controlunit_state <= E_NN;
  current_datum <= current_datum + 1;
end if;

when DONE_TRAINING_RUN =>
  if attempt_counter = 5 then
    cntr_increment <= '1';
    train_counter <= train_counter - 1;
    attempt_counter <= 0;
    if train_counter = 1 then
      controlunit_state <= ALMOST_IDLE;
    else
      next_state <= START_TRAINING_RUN;
      controlunit_state <= GET_WEIGHT;
    end if;
  else
    attempt_counter <= attempt_counter + 1;
    next_state <= START_TRAINING_RUN;
    controlunit_state <= GET_WEIGHT;
  end if;

when SEND_RESULTS =>
  if tb_full = '0' then
    send_frame <= '1';
    controlunit_state <= ALMOST_IDLE;
  end if;

when others =>
  controlunit_state <= ALMOST_IDLE;

end case;
end if;
end if;
end process statechange;
end architecture rtl;
```

# Appendix C8 – VHDL code for ENTIRE UNIT

## C.8.1 Neural Network Trainer

```
-----
-- Neural Network Trainer
-- Neural Network trainer using the Univariate random search algorithm
-- Author      : Guillermo Barreiro, Shyam Chadha, Reid Orsten, Andrew
--              Ling, Timmy Li, Darren Gonek
-- Date        : March 20, 2002
-- File Name   : NN_trainer.vhd
-- Architecture : Mixed
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.final_pack.all;

entity nn_trainer is
    port(clock, reset : in std_logic;
          cmpserialout, fifo_rx_full : out std_logic;
          cmpserialout : in std_logic; );
end entity ;

architecture mixed of nn_trainer is
    signal cmd_wires,addr_wires, sel_wires :std_logic_vector( 1 downto 0);
    signal fifo_read_ack, fifo_full, fifoempty_wire, done_calculations : std_logic;
    signal ack_nn, related_transmission : std_logic;
    signal result_nn, input1_wires: std_logic_vector ( 7 downto 0);
    signal input2_wires, general_data: std_logic_vector ( 7 downto 0);
    signal data_from_fifo: std_logic_vector (31 downto 0);
    signal clk5m, clk1m, clk100k, clk10k, clk1k, clk100, clk10, clk1 : std_logic;
    signal clock_wire : std_logic;

begin

Clock_divisor: clk_div
    port map(      clock_25Mhz => clock,
                  clock_5MHZ => clk5m,
                  clock_1MHz => clock_wire,
                  clock_100KHz => clk100k,
                  clock_10KHz => clk10k,
                  clock_1KHz => clk1k,
                  clock_100Hz => clk100,
                  clock_10Hz => clk10,
                  clock_1Hz => clk1
    );

component control_unit is
generic map (
    BUS_ADDR_WIDTH => 4,
    BUS_SEL_WIDTH => 4,
    BUS_DATA_WIDTH => 8,
    DATA_WIDTH => 8
)

```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
port (
    -- Trainer Bus Signals
    bus_addr => addr_wires,
    bus_sel  => sel_wires,
    bus_data => general_data,
    input1  => input1_wires,
    input2  => input2_wires,
    bus_cmd => cmd_wires,
    ack     => ack_nn,
    done    => done_calculations,
    results => result_nn,
    -- RS232 Interface Signals
    data_in  => data_from_fifo,
    fifo_empty => fifoempty_wire,
    tb_full  => fifo_full,
    data_acknowledge => fifo_read_ack,
    -- Miscellaneous
    reset    => reset,
    clock    => clock_wire
);

Serial_communication: serialtop
    generic map (
        InDivisor => 82,
        OutDivisor=> 1311
    )
    port map (
        clock => clock_wire,
        reset => reset,
        -----
        tx_fifo_write => related_transmission,
        tx_fifo_input :in std_logic_vector(15 downto 0);
                                --Data to send back to pc
        cmpserialout => cmpserialout,
        tx_fifo_full => fifo_full,
        -----
        rx_fifo_output => data_from_fifo,
        rx_fifo_read_ack => fifo_read_ack,
        cmpserialin cmpserialin,
        rx_full => fifo_rx_full,
        rx_empty => fifoempty_wire
    );

NN_datapath: nn_dp_3
    generic map (
        cmd_bitsize => 4,
        sel_bitsize => 4,
        data_bitsize => 8
    )
    port map (
        cmd => cmd_wires,
        sel => sel_wires,
        addr => addr_wires,
        databus => general_data,
        In1 => input1_wires,
        In2 => input2_wires,
        clock => clock_wire,
        reset => reset,
        result => result_nn,
        done => done_calculations,
        ack => ack_nn
    );

end mixed;
```

### C.8.2 Top Level Package of Hardware Device

```
-----
-- Top Level Package of Hardware Device
-- Author      : Guillermo Barreiro, Shyam Chadha, Reid Orsten, Andrew
--              Ling, Timmy Li, Darren Gonek
-- Date       : March 20, 2002
-- File Name  : final_pack.vhd
-- Architecture : Mixed
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE final_pack IS

    -- Component Declaration
    component trainer_control is
        generic(
            CMD_WORD_SIZE: natural := 8;
            CMD_BUS_SIZE:  natural := 2;
            BUS_ADR_SIZE:  natural := 2;
            BUS_SEL_SIZE:  natural := 2;
            NUM_NEURONS:   natural := 3;
            NUM_INPUTS:    natural := 2
        );
    port(
        cmd_word: in std_logic_vector(7 downto 0);
        activedata: in std_logic_vector(15 downto 0);
        clock, reset, last_weight, ack: in std_logic;
        done, ce_done, tb_full: in std_logic;
        new_data, calc_err: inout std_logic;
        bus_cmd: out std_logic_vector(CMD_BUS_SIZE-1 downto 0);
        cntr_increment, go, busy, clear, read_rand: out std_logic;
        assert_rand, assert_data: out std_logic;
        save_to_ram, send_frame, assert_stored: out std_logic;
        load_address, mem_reset: out std_logic
    );
end component;

component serialtop is
    generic (
        InDivisor: positive := 82;    --7    115200bps --82 9600bps;
        OutDivisor: positive := 1311 --109 115200bps --1311 9600bps
    );
    port(
        clock, reset: in std_logic;
        -----
        tx_fifo_write: in std_logic;
        tx_fifo_input: in std_logic_vector(15 downto 0);
        cmpserialout : out std_logic;
        tx_fifo_full: out std_logic;
        -----
        rx_fifo_output : out std_logic_vector(31 downto 0);
        rx_fifo_read_ack: in std_logic;
        cmpserialin : in std_logic;
        rx_full : out std_logic;
        rx_empty : out std_logic
    );
end component;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
component nn_dp_3 is
    generic ( cmd_bitsize : positive := 2;
              sel_bitsize : positive := 2;
              data_bitsize: positive := 8);
    port ( cmd : in std_logic_vector ( cmd_bitsize -1 downto 0);
          sel : in std_logic_vector ( sel_bitsize-1 downto 0);
          addr : in std_logic_vector( 1 downto 0);
          databus : inout std_logic_vector ( data_bitsize -1 downto 0);
          In1, In2 : in std_logic_vector ( data_bitsize - 1 downto 0);
          clock, reset : in std_logic;
          result : out std_logic_vector ( data_bitsize -1 downto 0);
          done : out std_logic);
end component;

component busmaster is
    generic (
        datawidth : positive := 8;
        halfwidth : positive := 4
    );
    port (
        clock :          in std_logic;
        read_rand:       in std_logic;
        read_data:       in std_logic;
        assert_rand:     in std_logic;
        assert_data:     in std_logic;
        assert_saved :   in std_logic;
        load_addr:       in std_logic;
        clear:           in std_logic;
        in_sel:          in std_logic_vector(halfwidth - 1 downto 0);
        incr_sel :       in std_logic;
        load_sel :       in std_logic;
        data:            inout std_logic_vector(datawidth - 1 downto 0);
        addr:            out std_logic_vector(halfwidth - 1 downto 0);
        sel:             out std_logic_vector(halfwidth - 1 downto 0);
        in_rng:          in std_logic_vector(datawidth - 1 downto 0);
        in_data:         in std_logic_vector(datawidth - 1 downto 0);
        in_addr:         in std_logic_vector (halfwidth -1 downto 0);
        done:            out std_logic;
        stored_data:     out std_logic_vector(datawidth -1 downto 0)
    );
end component;

component prng is
    generic(Width: positive := 8);
    port (
        clock: in std_logic;
        reset : in std_logic;
        output: out std_logic_vector(Width-1 downto 0)
    );
end component;

END final_pack;
```

# Appendix D: Problem Specification and Software Experimentation

## APPENDIX D: INDEX

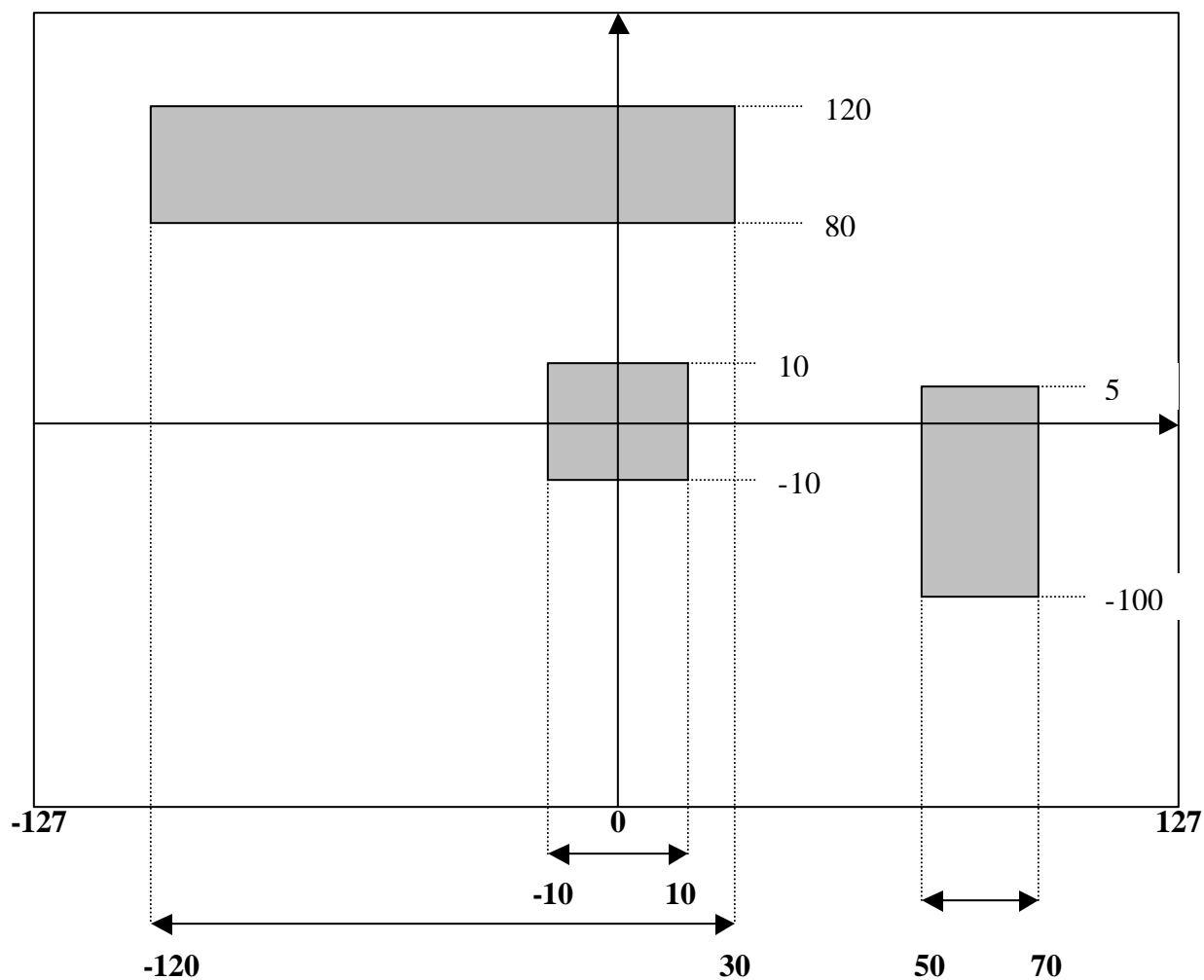
- D.1: Problem Description
- D.2: Training Input File
- D.3: Weights Input/Output File
- D.4: Matlab Code
- D.5: C Code
- D.6: Matlab Code to Visually Display the Solution

## D.1 Problem Description

**The Hardware Implementation is compared to the following experimentation done in software (Both in Matlab and in C)**

### **Problem to solve:**

**The Classification of Points within a Circle on the X-Y Plane.**



**FIGURE 12:** Classification Regions of the X-Y Plane

### D.2 Training Input File

#### Training Input File:

All values in the input file are required to be 8-bit Hexadecimal values. Hence, single digit values must consist of two characters.

The input file is organized as:

Number of inputs

Number of Hidden neurons

Number of outputs

Input X1 InputY1

Output1

Input X2 InputY2

Output2

Input Xn InputYn

Outputn

---

#### // Training Input File

```
02
02
01
00 00
00
00 02
00
00 05
00
01 01
00
01 02
01
01 04
00
02 00
00
02 01
01
02 02
01
02 03
01
02 04
01
02 05
00
03 02
01
04 01
00
04 02
00
04 05
00
05 00
00
05 02
00
05 05
00
```

### **D.3 Weights Input/Output File**

#### **Weights Input/Output File:**

This file is written to after a training session, or called from if the Neural Network's weights are to be loaded.

Each line below of the weights file represents 8-bit Hexadecimal values. Hence, single digit values must consist of two characters (have a zero in front), and the Neuron\_Number|Weight\_Number consists of two values (the first is in the range of 0 to 6, and the second either 1 or 2).

The weight value represents the numerator to be normalized as the weight.

The weights file is organized as:

```
Neuron_Number|Weight_Number
Weight
Neuron_Number|Weight_Number
Weight
Neuron_Number|Weight_Number
Weight
Neuron_Number|Weight_Number
Weight
Neuron_Number|Weight_Number
Weight
Neuron_Number|Weight_Number
Weight
```



## D.4 Matlab Code

# Matlab Code

### Matlab:

```
%-----
% NN training algorithm
% Random search
% Guillermo Barreiro March 03 - 2002
%-----
clear;
clc;
global Nin Nhid Nout;
%-----
% Itialization of variables
Nin = 2;
Nhid = 2;
Nout = 1;

Is = 5;
Imax = 10000;
b = -1.0;
Ehold = 10000000000;
Etreshhold = 0.001;

Errorgraphic = zeros(Imax-1,1);
temp = 0.5*Imax;
%-----
%Load Training data base
name = 'h:\Documents\EE552\Labs\Final Project\NN algorithm - Matlab\database.txt';
fid = fopen(name, 'r');
if fid == -1 ;
    fprintf(' The Database could not be opened');
else
    sizedb = str2num(fgetl(fid));
    In = zeros(sizedb, Nin);
    T = zeros( sizedb, Nout);
    for i=1:sizedb
        temp = str2num(fgetl(fid));
        for j=1:Nin
            In(i,j)= temp(j);
        end
        temp = str2num(fgetl(fid));
        for j=1:Nout
            T(i,j)= temp(j);
        end
    end
    fclose(fid);
end
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
%-----
%Create initial random weights
pause(0.001);
rand('state',sum(100*clock));

W = zeros(Nin,Nhid); % weights from input layer to hidden layer
U = zeros(Nhid, Nout); % inputs from hidden layer to output layer
W = rand(Nin, Nhid) - 0.5;
U = rand(Nhid, Nout) - 0.5;

Error = Update_NN(W,U,In,T,sizedb);
%-----
%Random placement of weights - Training
epochs = 1;
while ( epochs < Imax | Error < Etreshhold)
%-----
% Pivots over hidden layer
for m=1:Nhid
%-----
%Changes weights going from input layer to Hidden layer
for n=1:Nin
    for i=1:Is
        Whold = W(n,m);
        %pause(0.001);
        %rand('state',sum(100*clock));
        W(n,m) = 2*b*rand(1,1) - b;
        Ehold = Update_NN(W,U,In,T,sizedb);
        if Ehold < Error
            Error = Ehold;
        else
            W(n,m) = Whold;
        end
    end
end
end
%-----
%Changes weights going from Hidden layer to output layer
for j=1:Nout
    for i=1:Is
        Uhold = U(m,j);
        %pause(0.001);
        %rand('state',sum(100*clock));
        U(m,j) = 2*b*rand(1,1) - b;
        Ehold = Update_NN(W,U,In,T,sizedb);
        if Ehold < Error
            Error = Ehold;
        else
            U(m,j) = Uhold;
        end
    end
end
end

Errorgraphic(epochs) = Error;
if mod(epochs,100) == 0
    fprintf('Generation %d processed ! Error = %f\n', epochs, Error);
end
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
%-----
if epochs == temp
    Is = 12;
end
epochs = epochs + 1;
end

% graphs the TSSE
plot(Errorgraphic);
% Saves the NN weights
fid = fopen('Weights.txt', 'w');

fprintf(fid, '\n\n Weights from Input to hidden layer \n\n');

for n = 1:Nin
    for m = 1:Nhid
        fprintf(fid, '%f', W(n,m));
    end
    fprintf(fid, '\n');
end

fprintf(fid, '\n\n Weights from Hidden to Output layer \n\n');

for m = 1:Nhid
    for j = 1:Nout
        fprintf(fid, '%f', U(m,j));
    end
    fprintf(fid, '\n');
end

fclose(fid);
%-----
% Function that calculates the TSSE of the NN

function output = Update_NN( W, U, In, T, sizedb);
global Nhid Nout Nin;
Y = zeros( sizedb, Nhid);
Z = zeros(sizedb, Nout);
S = zeros(sizedb, 1);
% Evaluation of the first layer
for m = 1:Nhid
    S = In * W(:,m);
    for i = 1:sizedb;
        Y(i,m) = (1.0 / (1.0 + exp(-S(i))));
    end
end

S = zeros(sizedb, 1);
% Evaluation of the second layer
for j = 1:Nout
    S = Y * U(:,j);
    for i = 1:sizedb
        Z(i) = 1.0 / (1.0 + exp(-S(i)));
    end
end
output = (sum( (Z-T).^2 )) / Nout / sizedb;
```

### **D.5 C Code**

#### **C – Programming Language:**

```
//-----  
// Main file  
//-----  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <time.h>  
#include <math.h>  
#include "NNT_lib.h"  
  
int  
main(void)  
{  
char file[30],Redname[30],Errname[30];  
double err,error_min=0.001, Whold, Uhold, Ehold;  
char tab=9;  
Redtype tred;  
Inputs XQ;  
Outputs TQ;  
int iq,Q;  
long int it,Imax;  
int n,m,j,i,Is, b;  
FILE *FicErr;  
  
Imax=10000;  
Is =5;  
b = -1;  
  
sprintf(file,"NNT");  
  
/* nombres de los archivos asociados */  
sprintf(Redname,"%s.txt",file);  
sprintf(Errname,"%s.err",file); /*Training error */  
  
/* Reads data From file */  
Readexamples(&tred, XQ, TQ, Redname, &Q);  
/* Generates random numbers to initialize the NN */  
Initweights(&tred);  
/* Calcualtes Initial Error */  
for(iq=0;iq<Q;iq++) Forward(&tred,&(XQ[iq]));  
/* Calcualtes the Network's error*/  
err=error_tot(&tred,&XQ,&TQ,Q);  
  
/* Open file to save error */  
FicErr=fopen(Errname,"wt");
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
/*Begins training algorithm*/
it=0;
while(it<=Imax && err > error_min)
{
    for (m=1; m<=tred.M; m++)
    {
        for(n=1; n<=tred.N; n++)
        {
            for(i=0; i<Is; i++)
            {
                Whold = tred.W[n][m];
                srand(time(NULL));
                tred.W[n][m] = 2*b*((double)(rand())/RAND_MAX) - b;
                Ehold = error_tot(&tred, &XQ, &TQ, Q);
                if (Ehold < err)
                    err = Ehold;
                else
                    tred.W[n][m] = Whold;
            }
        }

        for(j=1; j<=tred.J; j++)
        {
            for(i=0; i<Is; i++)
            {
                Uhold = tred.U[m][j];
                srand(time(NULL));
                tred.U[m][j] = 2*b*((double)(rand())/RAND_MAX) - b;
                Ehold = error_tot(&tred, &XQ, &TQ, Q);
                if (Ehold < err)
                    err = Ehold;
                else
                    tred.U[m][j] = Uhold;
            }
        }
    }

    if (it % 1000 == 0)
    {
        printf("Epochs = %d Error = %f \n", it, err);
    }

    if ( it%100 == 0) fprintf(FicErr,"Iteration: %d Error: %f\n",it, err);
    if ( it > 0.5*Imax) Is = 12;

    it++;
}
SaveWeights(&tred);
fclose(FicErr);
}
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
//-----  
// Library.C  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <time.h>  
#include <math.h>  
  
#include "NNT_lib.h"  
  
void Readexamples(Redtype *Red, Inputs inputs, Outputs outputs, char *name, int *Q){  
    int i, N, M, J, q=0;  
    float temp;  
    FILE *cfptr;  
  
    if((cfptr = fopen(name, "r")) == NULL)  
        printf("Database file could not be opened !");  
    else{  
        fscanf(cfptr, "%d %d %d ", &N, &M, &J);  
        i=0;  
        while(!feof(cfptr)){  
            for(i=1; i<=N; i++){  
                fscanf(cfptr, "%f", &temp);  
                inputs[q][i]=temp;  
            }  
            for(i=1; i<=J; i++){  
                fscanf(cfptr, "%f", &temp);  
                outputs[q][i]=temp;  
            }  
            inputs[q][0]=-1;  
            q++;  
        }  
        (*Red).N=N;  
        (*Red).M=M;  
        (*Red).J=J;  
        (*Q)=q-1;  
        fclose (cfptr);  
    }  
}  
//-----  
void Initweights(Redtype *neuro){  
    int n, m, j;  
    double x;  
    srand(time(NULL));  
    for(m=0; m<=(*neuro).M; m++){  
        for(j=1; j<=(*neuro).J; j++){  
            x=(double)(rand())/RAND_MAX;  
            (*neuro).U[m][j]=(x-0.5);  
        }  
        for(n=1; n<=(*neuro).N; n++){  
            x=(double)(rand())/RAND_MAX;  
            (*neuro).W[n][m]=(x-0.5);  
        }  
    }  
}
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
//-----
void Forward(Redtype *red, Vecin *Xin)
{
    int n, m, j ;
    double s;
    /* Copies inputs in Nets Structure */
    for (n=1;n<=red->N;n++) red->X[n]=(*Xin)[n];
    /* Input 0 is polarization */
    red->X[0]=-1.0;
    //-----
    /* First layer calculations */
    for(m=1;m<=red->M;m++){
        s=0;
        for(n=0;n<=red->N;n++)
            s=s+(red->W[n][m])*red->X[n];
        red->Y[m]=1.0/(1.0+exp(-s));
    }
    red->Y[0]=-1.0;
    //-----
    /* Second layer calculations */
    for(j=1;j<=red->J;j++){
        s=0;
        for(m=0; m<=red->M; m++)
            s=s+(red->U[m][j])*red->Y[m];
        red->Z[j]=1.0/(1.0+exp(-s));
    }
    //-----
    double error_tot (Redtype *Red, Inputs *Xin, Outputs *T, int Q)
    {
        int j,q;
        double Enew=0;
        for (q=0;q<Q;q++)
        {
            Forward(Red,&((*Xin)[q]));
            for (j=1;j<=Red->J;j++)
            {
                Enew=Enew+pow(((T)[q][j])-(Red->Z[j])),2);
            }
        }
        Enew=Enew/Red->J/Q;
        return Enew;
    }
    //-----
    void SaveWeights(Redtype *red)
    {
        FILE *fic;
        int n,m,j;
        char name[30];

        sprintf(name,"redes.txt");
        fic=fopen(name,"wt");

        /* Architecture Topology */
        fprintf(fic,"N  %d M %d J %d\n",(*red).N,(*red).M,(*red).J);
    }
}
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
/* Weights */
for (m=1;m<=(*red).M;m++)
{
    for (n=1;n<=(*red).N;n++) fprintf(fic,"W[%d][%d]=%f\n",n,m,(*red).W[n][m]);
}

for (j=1;j<=(*red).J;j++)
{
    for (m=1;m<=(*red).M;m++) fprintf(fic,"U[%d][%d]=%f\n",m,j,(*red).U[m][j]);
}

fclose(fic);
}
//-----
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

### //Library.H

```
#define Nmax 20
#define Mmax 20
#define Jmax 10
#define Qmax 50

typedef double Vecin [Nmax];
typedef double Vecinter [Mmax];
typedef double Vecout [Jmax];
typedef Vecinter layer1[Nmax];
typedef Vecout layer2[Mmax];

typedef struct {
    int N; /* Numero de entradas */
    int M; /* Numero de Neuronas ocultas */
    int J; /* Numero de Neuronas de salida */
    layer1 W; /* Pesos de la capa oculta */
    layer2 U; /* pesos de la capa de salida */
    Vecin X; /* entradas */
    Vecinter Y; /* salidas de la capa oculta */
    Vecout Z; /* salidas de la red */
} Redtype;

/* estructuras para la base de aprendizaje */
typedef Vecin Inputs [Qmax];
typedef Vecout Outputs [Qmax];

/* Rutinas de la libreria */
/* Reading of examples */
void Readexamples(Redtype *Red, Inputs inputs, Outputs outputs, char *name, int *Q);
/* Inicializacion de pesos */
void Initweights(Redtype *neuro);
/* NN Evaluation to one input */
void Forward(Redtype *red, Vecin *Xin);
/* Calculates the NN's error over all the training database */
double error_tot(Redtype *Red, Inputs *Xin, Outputs *T, int Q);
/* Saves the weights of the Network */
void SaveWeights(Redtype *red);
```