AN1211

Interfacing DACs and ADCs to the NEURON IC

Control data often consists of voltage values from sensors or to controllers interfaced to a microprocessor. A system designer may be faced with the challenge of designing a network for monitoring and controlling large numbers of voltage inputs and outputs. The classical approach has been to design a central control unit which monitors and controls all inputs and outputs. Such a system can suffer size and reliability constraints, as well as a potentially drastic single point of failure. An alternative approach is to design a distributed network in which each sensor and controller in a system has processing power and can communicate on a common network medium. The MC143150 (NEURON IC) caters to such a solution. It is a distributed communication and control processor with an embedded firmware protocol tailored to the transmission of control data on a network (refer to the NEURON IC technical data for details — document # MC143150/D).

The concept of “smart nodes” on a network is illustrated in Figure 1 — a block diagram of DACs and ADCs on a distributed network. The converter ICs are interfaced to NEURON ICs which communicate on the network to another node referred to as the network controller NEURON IC. The DACs and ADCs may be monitored and controlled from a remote location using the network controller which could be interfaced to a user–friendly computer environment.

This application note will describe the interfaces between the MC143150 NEURON IC and two voltage converter ICs: the MC144110 — a 6–bit, 6–channel digital–to–analog converter (DAC); and the MC14443 — an 8–bit, 6–channel analog– to–digital converter (ADC). The first example will detail communication between the network controller NEURON IC and a DAC controller NEURON IC which serially controls a DAC through its I/O port. And the second example will detail the same network controller NEURON IC communicating to an ADC controller NEURON IC which monitors an ADC through its I/O port. Software for both chip controller NEURON ICs as well as the network controller NEURON IC is listed at the end of this document. Note that the software solutions are written in NEURON C (a NEURON C Programmers Guide should be used to reference unfamiliar function calls and events).

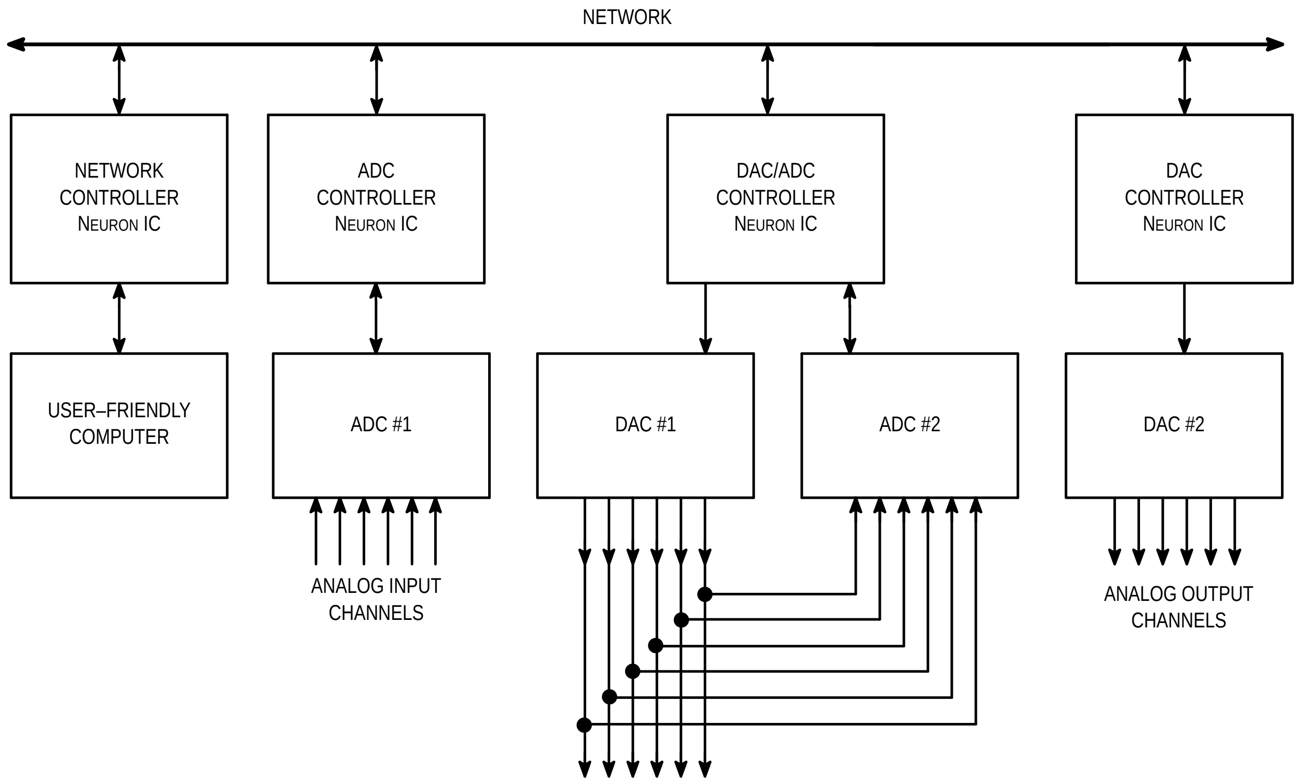

Figure 1. Block Diagram of ADC and DAC Chips Interfaced to NEURON ICs on a Network

NEURON IC INTERFACE TO DIGITAL–TO–ANALOG CHIP

INTRODUCTION

This section describes how the NEURON IC may be used to drive a Digital–to–Analog Converter (DAC).

The MC144110 is a low–cost, 6–bit DAC with a synchronous serial interface port to provide NEUROWIRE (identical to Motorola's SPI) communication with the MC143150. The chip contains six static D/A converters and operates between 4.5 and 15 V. The MC144110 can be cascaded if more than six outputs are required.

The following example describes one NEURON IC as a DAC controller and another NEURON IC as a network controller. The network controller will send a channel number and voltage value to the DAC controller which will command its DAC accordingly. See Figure 2 for a block diagram of the system.

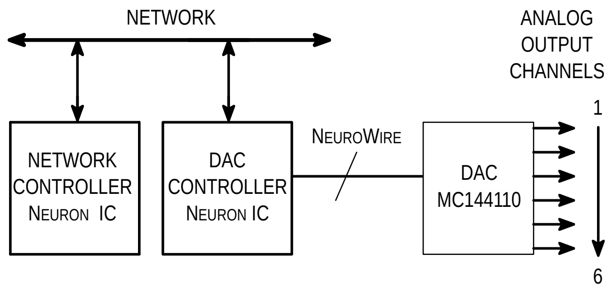

Figure 2. Block Diagram of NEURON IC — DAC Interface

CIRCUIT

The serial interface between the MC143150 and the MC144110 is defined as follows: IO_0 is a chip select for the DAC, IO_8 is a clock output, and IO_9 is a data output. See Figure 3 for details of the circuit used in this example.

SOFTWARE

Function

The DAC controller NEURON IC receives a channel number (1–6) and a voltage value (0–16383 mV) from the network controller; converts and formats the voltage value; and sends it to the DAC.

Network Variables

The network controller NEURON IC sends a structure called to_dac on the network containing a 'channel number' field (1 byte) and a 'voltage value' field (2 bytes). The DAC controller NEURON IC uses the nv_update_occurs event (a pre–defined NEURON C event) to receive this structure from the network controller.

Conversion

The 2–byte network input voltage received in millivolts must be scaled to a 6–bit number and placed in the most significant 6 bits of a byte memory location. The DAC controller uses the following scaling equation:

Vdac = Vnet / (Vdd/2^6) – 1

where Vnet = the 2–byte voltage value sent on the network
Vdd = the DAC supply voltage
and Vdac = the converted 6–bit DAC voltage

The conversion process completes by shifting Vdac two bits to the left.

Format

The DAC expects 36 bits in each transmission from the DAC controller NEURON IC (a 6–bit value for each channel). Hence, the DAC controller must format a 36–bit block (5 bytes) from a 6–byte block. In other words, the DAC controller stores a 6–bit analog value in a byte memory location for each DAC channel value and must compact a 6–byte array (called net_data) to a 5–byte array (called DAC_data). See Figure 4 for a graphic representation of RAM configuration in the DAC controller during the conversion and format functions. Note that a 6–bit DAC requires software overhead that would not be required for an 8–bit DAC (the format function would not be necessary with an 8–bit DAC).

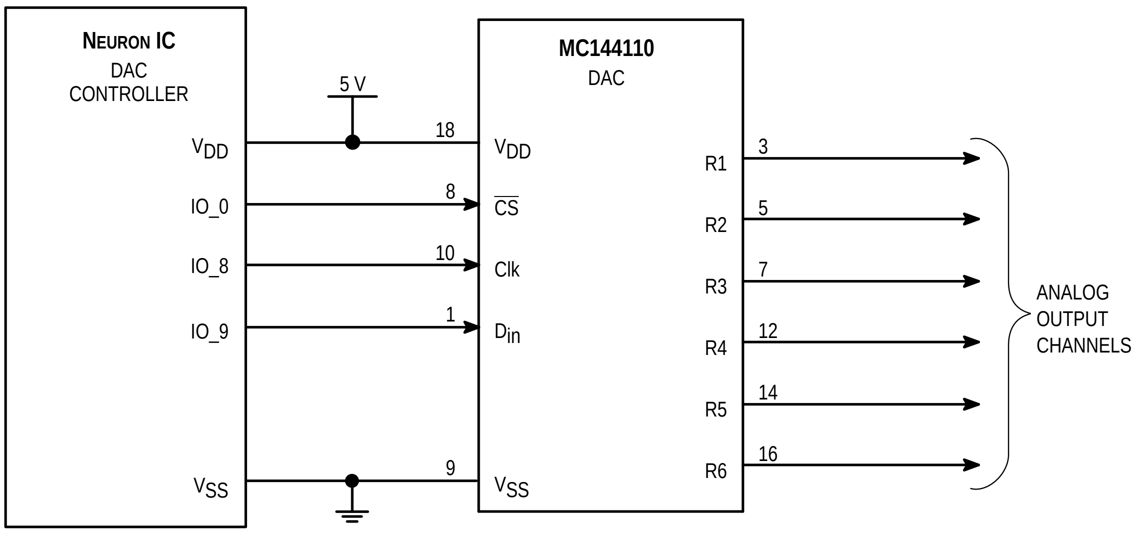

Figure 3. NEURON IC to MC144110 Interface

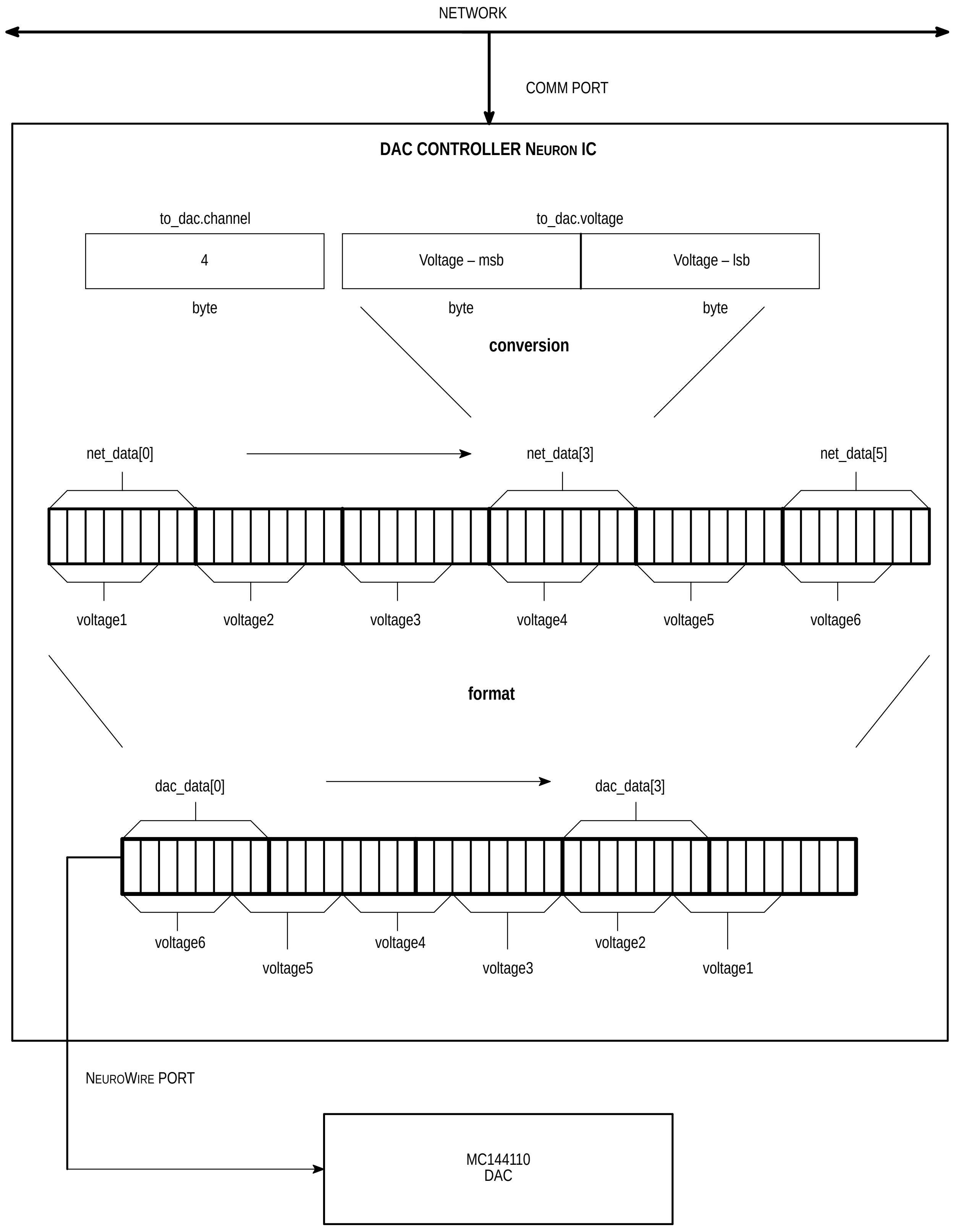

Figure 4. RAM Configuration of DAC Controller

DAC Addressing

The DAC controller transmits the formatted array (DAC_data) to the DAC in a packet of 36 bits (msb first). The first 6 bits received by the DAC control channel 6, the next 6 control channel 5, and so on.

The NEURON C software for the DAC controller NEURON IC is presented in Print Out 1. Note that the software can easily be modified to accommodate the 4–channel MC144111 (change the constant called NUM_CHANNELS to 4). The network controller NEURON IC is programmed to periodically send voltage values between 0 and Vdd to DAC channels in a sequential fashion (see Print Out 3).

NEURON IC INTERFACE TO ANALOG–TO–DIGITAL CHIP

INTRODUCTION

This section describes how the NEURON IC may be used to monitor an Analog–to–Digital Converter (ADC).

The MC14443 is a low cost, single–slope, 6–channel, 8–10–bit ADC linear subsystem for microprocessor–based control. It contains a ramp start circuit and a comparator which will pulse out for a time proportional to the voltage on one of the ADC channels. Its comparator output driver is an open drain N–channel which provides a sinking current. The analog input range on a channel is between 0 and Vdd – 2.0 V.

The following example describes one NEURON IC as an ADC controller and another NEURON IC as a network controller. The network controller will send a channel number to the ADC controller which will poll its ADC for a voltage value to send back to the controller. See Figure 5 below for a block diagram of the system.

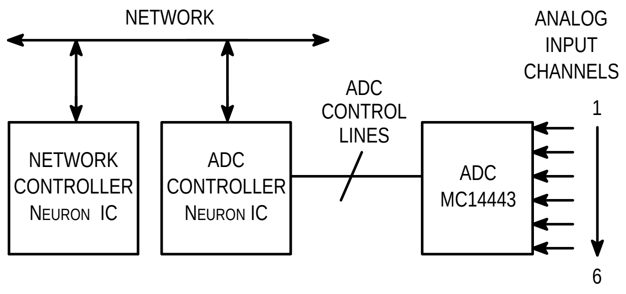

Figure 5. Block Diagram of NEURON IC — ADC Interface

CIRCUIT

The interface between the MC143150 and the MC14443 is defined as follows: IO_0–IO_3 use a nibble output object to address channels and control the ramp capacitor on the ADC; IO_4 is an on–time input from the comparator on the ADC. See Figure 6 for details of the circuit used in this example.

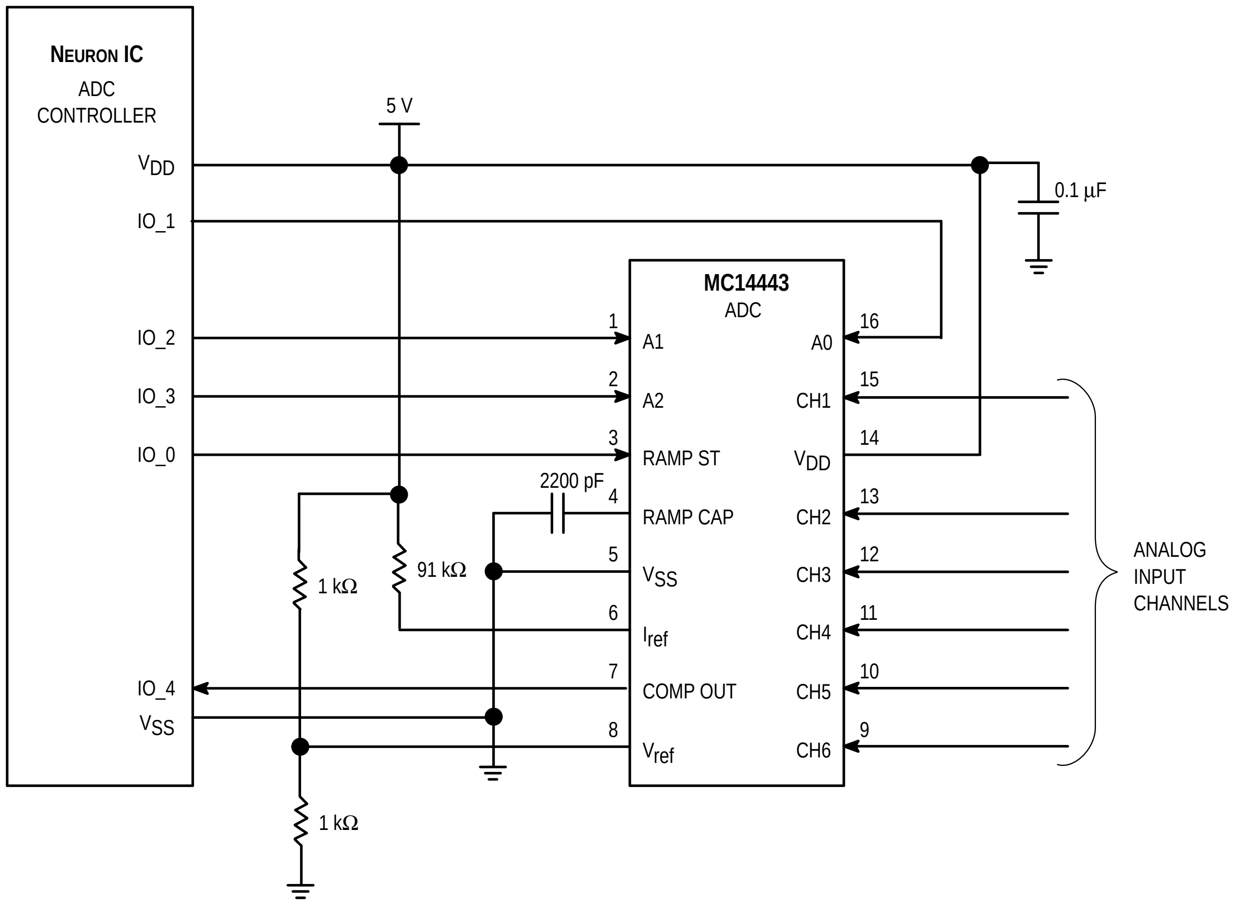

Figure 6. NEURON IC to MC14443 Interface

SOFTWARE

Function

Upon reset the ADC controller NEURON IC must compute a time reference based on the levels of Vref and Vss from the ADC. During normal operation the ADC controller receives a channel (1–6) from the network controller; addresses the ADC; measures its on–time input; converts a time value to a voltage value; and transmits the voltage value (0 – 16383 mV) and channel number (1–6) back to the network controller.

Network Variable Input

The network controller NEURON IC sends a ‘channel number’ variable called address (1 byte) on the network. The ADC controller NEURON uses the nv_update_occurs event (a pre–defined NEURON C event) to receive this structure from the network.

ADC Addressing

The ADC controller selects the appropriate channel via its nibble control lines and begins to record time after starting the ADC ramp capacitor.

Conversion

The discharge time of the ADC ramp capacitor must be converted from 200 ns units to mV. The ADC controller uses the following conversion formula:

Vadc = Vref * (tchannel / tref)

where Vref = known voltage to which unknown voltage is compared

tchannel = on–time of channel voltage

and tref = on–time of Vref – on–time of Vss

The conversion formula may need to be adjusted for maximum resolution.

Network Variable Output

After the conversion, the ADC controller transmits a structure called from_adc to the network controller. The structure contains a channel number (1 byte) and a voltage value (2 bytes) which the network controller stores in an array called adc_array.

See the NEURON C software for the ADC controller NEURON IC in Print Out 2. Note that this software will also accommodate the MC14447. The network controller NEURON IC is programmed to periodically poll channels in a sequential fashion (see Print Out 3).

CONCLUSIONS

In conclusion, the NEURON IC can be a very practical processor in a large system of ADCs and DACs. In such a system, the network controller could serve as a network interface to a more powerful ‘‘number crunching’’ 32–bit microcontroller or the flexible environment of a PC. Also, a manual mode at the network controller could provide a human interface (i.e. a keyboard and display) for voltage monitoring and control. Finally, the ADC and DAC controller NEURON ICs could provide diagnostic information to the network controller from thousands of remote locations. In this manner, NEURON IC driven distributed systems provide capabilities and advantages that centralized control systems cannot offer.

```
////////////////// Print Out 1. Software for DAC Controller NEURON IC ////////////////
//
// This software enables a NEURON IC to drive an MC144110 DAC.
// Data are converted by the conversion function; the frame is
// created by the format function.
//
//////////////////////////////////////////////////////////////////////////////////////

#define VDD 5000
#define NUM_CHANNELS 6

typedef struct
{
       unsigned int channel_number;
       unsigned long Vnet;
} dac_control_struct;

network input dac_control_struct NV_DAC_struct;

IO_8 neurowire master select (IO_0) IO_DAC;
IO_0 output bit IO_DAC_select;

unsigned int net_data[6], DAC_data[5], Vdac;
unsigned int conversion (unsigned long Vnet, unsigned long Vdd)
{
      unsigned long temp1;
      unsigned int temp2;

      temp1 = Vdd >> 6;              //divide VDD by 2^6
      if (Vnet <= temp1) temp2 = 0;         //determine if voltage value is 0
      else temp2 = ((Vnet / temp1) – 1) << 2;     //calculate 6–bit voltage value
      return (temp2);
}            // end conversion()

void format (unsigned int *net_data, unsigned int *DAC_data)
{
      unsigned int temp3;

      // 6msb of net_data[0] goes into 4msb of DAC_data[4] and 2 lsb of DAC_data [3]
      temp3 = net_data[0] << 2;
      DAC_data[4] = temp3 & 0xf0;
      DAC_data[3] = net_data[0] >> 6;

      // 6msb of net_data[1] goes into 6msb of DAC_data[3]
      temp3 = net_data[1] & 0xfc;
      DAC_data[3] |= temp3;

      // 6msb of net_data[2] goes into 6lsb of DAC_data[2]
      DAC_data[2] = net_data[2] >> 2:

      // 6msb of net_data[3] goes int 2msb of DAC_data[2] and 4lsb
      // of DAC_data[1]
      temp3 = net_data[3] << 4;
      temp3 &= 0xc0;
      DAC_data[2] |= temp3;
      DAC_data[1] = net_data[3] >> 4;

      // 6msb of net_data[4] goes into 4msb of DAC_data[1] and 2lsb
      // of DAC_data[0]
      temp3 = net_data[4] << 2;
      temp3 &= 0xf0;
      DAC_data[1] |= temp3;
      DAC_data [0] = net_data[4] >> 6;

      // 6msb of net_data[5] goes into 6msb of DAC_data[0]
      temp3 = net_data[5] & 0xfc;
      DAC_data[0] |= temp3;
}            // end format()

when (reset)
{
      poll();
      net_data[0] = conversion(0,VDD);            //0 volts on channel 1
      net_data[1] = conversion(1000,VDD);         //1 volt on channel 2
      net_data[2] = conversion(2000,VDD);         //2 volts on channel 3
      net_data[3] = conversion(3000,VDD);         //3 volts on channel 4
      net_data[4] = conversion(4000,VDD);         //4 volts on channel 5
      net_data[5] = conversion(5000,VDD);         //5 volts on channel 6
      format(net_data,DAC_data);
      io_out (IO_DAC, &DAC_data, (NUM_CHANNELS * 6));    //serial xmit to DAC
}            // end when

when (nv_update_occurs(NV_DAC_struct))
{
      Vdac = conversion(NV_DAC_struct.Vnet, VDD);    //convert 2–bytes to 6–bits
      net_data[NV_DAC_struct.channel_number – 1] = Vdac;
      format (net_data, DAC_data);      //format serial message
      io_out (IO_DAC, &DAC_data, (NUM_CHANNELS * 6));    //serial xmit to DAC
}            // end when
```

```
///////////////// Print Out 2. Software for ADC controller NEURON IC //////////////////
//
// This software enables a NEURON IC to drive an MC14443 ADC.
// Selected channels are read from an ontime input on the NEURON.
// Converted voltage values are transmitted on the network.
//
/////////////////////////////////////////////////////////////////////////////////////

#include <control.h>
#define VREF 2500

IO_4 input ontime mux clock(0) comparator;
IO_0 output nibble control;

typedef struct {
      unsigned int channel;
      unsigned long voltage;
 } adc_struct;

unsigned long temp_time1;
unsigned long temp_time2;
unsigned long time_reference;
unsigned int local_channel;

network input unsigned int address;
network output adc_struct adc_data;

when (reset)
{
      // the first pulse is bogus since the first ontime input to the NEURON IC
      // after a reset is invalid
      io_out (control,0x0e);         // address Vref
      delay (40);                    // allow ramp cap to charge to Vref
      io_out (control,0x0f);         // begin discharge on ramp cap
      delay (40);                    // allow ramp cap to discharge
      io_out (control,0x0e);         // address Vref
      delay (40);                    // allow ramp cap to charge to Vref
      io_out (control,0x0f);         // begin discharge on ramp cap
      while (1)                      // gadfly
      {
            watchdog_update();       // tickle watchdog timer
            if (io_update_occurs (comparator))    //is ramp cap discharge is complete ?
            {
                  temp_time1 = input_value;           //record discharge time
                  goto jump1;    //exit gadfly loop
            }           // end if
      }           // end while
      jump1:
      io_out (control,0x00);         // address Vss
      delay (40);                    // allow ramp cap to charge to Vss
      io_out (control,0x01);         // begin discharge on ramp cap
      while (1)                      // gadfly
      {
            watchdog_update ();            // tickle watchdog timer
            if (io_update_occurs (comparator))    //is ramp cap discharge complete ?
            {
                  temp_time2 = input_value;        // record discharge time
                  goto jump2;          // exit gadfly loop
            }           // end if
      }           // end when
      jump2:
      time_reference = (temp_time1 – temp_time2) / 10;   //calculate time reference
}           // end when
```

```
when (nv_update_occurs (address))
{
      local_channel = address << 1;        //address = 2 * channel number
      io_out (control,local_channel);      // address channel
      delay (40);             // allow ramp cap to charge to unknown voltage
      io_out (control,local_channel+1);    // begin ramp cap discharge
      while (1)               // gadfly
      {
            if (io_update_occurs (comparator))   // is ramp discharge complete ?
            {
                 temp_time1 = input_value / 10;  //record discharge time
                 goto jump3;        // exit gadfly loop
            }          // end if
      }          // end while
      jump3:
      adc_data.voltage = ((VREF / time_reference) * temp_time1);   //calculate voltage
      adc_data.channel = local_channel >> 1;            // send channel number
}            //end when
```

```
/////////////////// Print Out 3. Software for Network Controller Neuron IC ///////////
//
// This software enables a Neuron IC to collect data from
// and send data to other Neurons on a network which control
// DACs and ADCs.
//
/////////////////////////////////////////////////////////////////////////////////////

#define NUM_DAC_CHANNELS 6
#define NUM_ADC_CHANNELS 6
#define MAX_DAC_VOLTAGE 5000

typedef struct {
      unsigned int channel;
      unsigned long voltage;
} converter_struct;

network output converter_struct to_dac;
network input converter_struct from_adc;
network output unsigned int adc_address = 1;

unsigned long adc_array[6];

mtimer adc_timer;
mtimer dac_timer;

when (reset) {
      adc_timer = 500;
      dac_timer = 750;
      to_dac.channel = 1;
      to_dac.voltage = 0;
}            //end when

// poll a different A/D channel every 2s
when (timer_expires (adc_timer)) {
      if (adc_address++ > NUM_ADC_CHANNELS) adc_address = 1;
      adc_timer = 2000;
}            //end when

// store new A/D values whenever they come in
when (nv_update_occurs (from_adc))
{
      adc_array[from_adc.channel – 1] = from_adc.voltage;
}            //end when

// transmit a new voltage value to a new channel every 2s
when (timer_expires (dac_timer))
{
      if (to_dac.channel++ >= NUM_DAC_CHANNELS) to_dac.channel = 1;
      if (to_dac.voltage >= MAX_DAC_VOLTAGE) to_dac.voltage = 0;
      else to_dac_voltage += 100;
      dac_timer = 2000;
}            //end when
```