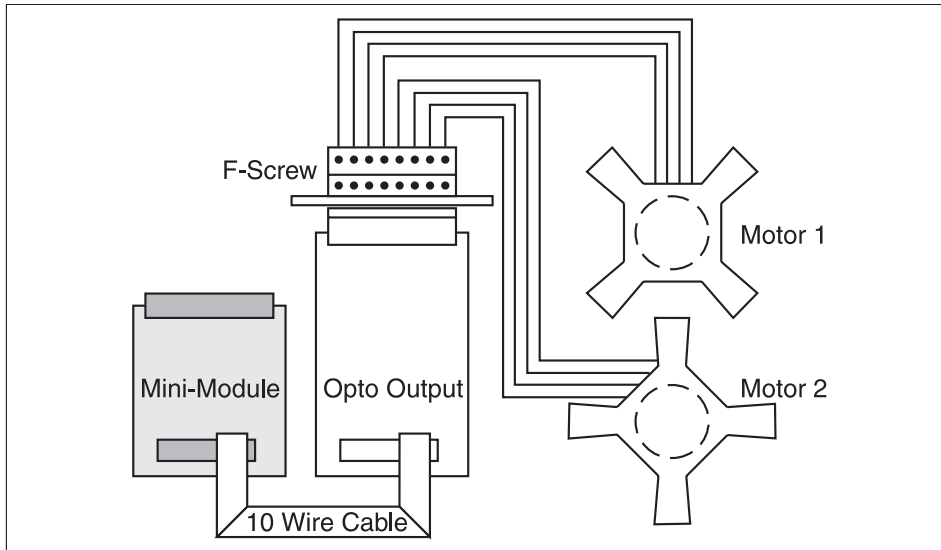




Application Examples

Some simple examples to show how easy it is to use the CMS range of Module products

Stepper Motor Application Note



This example uses the following equipment

- 1 K-100 Mini-Module
- 1 D-161 3 Amp, 8 Channel Opto Isolated Output
- 1 F-010 Type F to Screw Terminal Board
- 2 Stepper Motors
(Radio Spares 332-082)

A method of mounting the two motors so that they interlock.

This example shows how the Mini-Module can be used to drive stepper motors very easily. The two stepper motors are fitted with plates shaped as shown in the drawing above. The two plates are set up so that they interlock. The code that is listed below will drive the two stepper motors in such a way that the plates do not hit each other.

Connect the motors up as follows. The white wire and the black wire from both motors should be connected together and connected to the Z row on the F-Screw board. All of the connections in the Z row should be connected together.

Taking one of the motors, the red wire should be connected to D30 on the F-Screw (note the screw numbers start at the top with number 2 and increment downwards by 2 each time). The green

wire to D26, the red/white wire to D22 and the green/white wire to D18.

The second motor should be connected as follows. The red wire to D14, the green wire to D10, the red/white wire to D6 and the green/white wire to D2.

On the opto isolated output board the lower set of links should be fitted connecting the card to the bottom eight digital channels. When the motors are wired up in this fashion the first motor's red wire is connected to channel 0, green wire to channel 1 etc. The second motor's red wire is connected to channel 4, its green wire to channel 5 etc.

```

MODULE Servo;
FROM ADIO IMPORT OutPort,
WritePort;
FROM System IMPORT Delay;

CONST stspeed = 41;
      finishspeed = 1;

VARY y : INTEGER;

PROCEDURE Step(data,speed : INTEGER);
(*
Writes the value data to the motors
causing the motor to move. Speed is
used to determine how fast the motor
moves round.
*)
BEGIN
WritePort(0,data);
Delay(speed)
END Step;

PROCEDURE Forback(x : INTEGER);
(*
This procedure drives the first motor
clockwise and the second anti-clockwise.
Variable x determines how many times

```

```

the motors step in this direction.
*)
BEGIN
REPEAT
Step(039h,y);
Step(06ch,y);
Step(0c6h,y);
Step(093h,y);
DEC (x)
UNTIL x = 0
END Forback;

PROCEDURE Stopfor(x : INTEGER);
(*
This procedure stops the first motor
and drives the second motor in a clock-
wise direction. The variable x deter-
mines how many steps the motor takes.
*)
BEGIN
REPEAT
Step(003h,y);
Step(006h,y);
Step(00ch,y);
Step(009h,y);
DEC (x)
UNTIL x = 0
END Stopfor;

PROCEDURE Backfor(x : INTEGER);
(*
This procedure drives the first motor
anti clockwise and the second motor
clockwise. The variable x determines
how many steps the motor takes in this
direction.
*)
BEGIN
REPEAT
Step(093h,y);
Step(0c6h,y);
Step(06ch,y);
Step(039h,y);
DEC (x)
UNTIL x = 0
END Backfor;

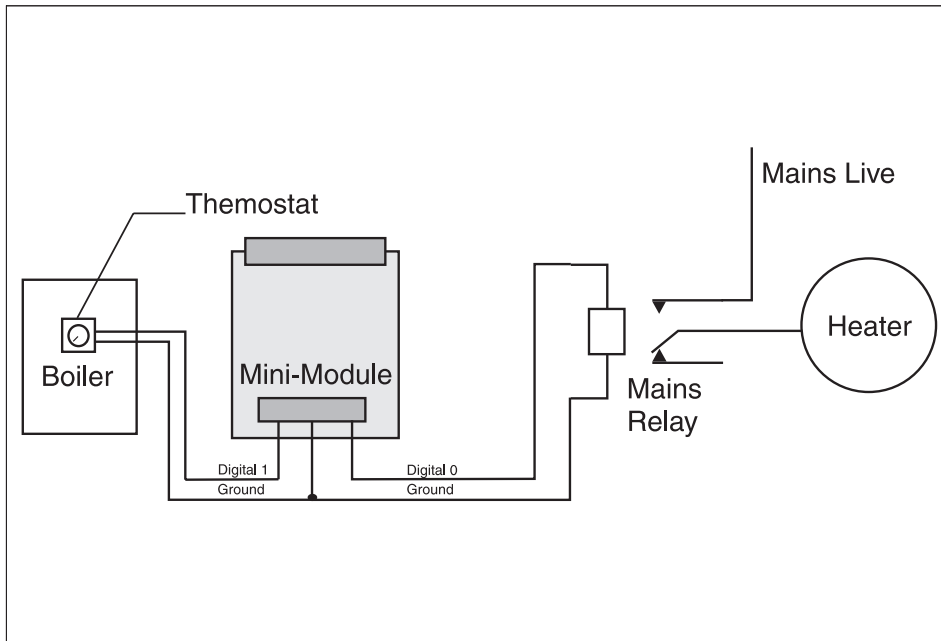
PROCEDURE Backstop(x : INTEGER);
(*
This procedure drives the first motor
anti clockwise and stops the second mo-
tor.
*)
BEGIN
REPEAT
Step(090h,y);
Step(0c0h,y);
Step(060h,y);
Step(030h,y);
DEC (x)
UNTIL x = 0
END Backstop;

BEGIN
OutPort(0);
LOOP
y := (2 * (stspeed
- finishspeed)) DIV 3;
Backfor(13);
y := (1 * (stspeed
- finishspeed)) DIV 4;
Forback(13);
y := finishspeed;
Stopfor(20);
Backstop(25);
Stopfor(36);
y := (3 * (stspeed
- finishspeed)) DIV 4;
Backstop(6);
y := finishspeed;
Forback(156);
Stopfor(25);
Delay(200)
END
END Servo.

```



Temperature Control of a Boiler using a Thermostat



The following equipment is required:

- 1 K-100 Mini-Module
- 1 Mains switching relay.

This example uses the Mini-Module to control a boiler with a thermostat controller attached to it. When the temperature in the boiler drops below the temperature set on the thermostat the heater is turned on. When the temperature in the boiler causes the thermostat to turn off it will turn the heater off.

This example uses just two of the thirty two digital lines available on the Mini-Module. One of the channels is used to monitor the state of the thermostat, the other to drive the relay. The following Module shows how easy it is to program the module to perform this function in Modula-2. Using this technique up to 16 boilers could be controlled.

The diode protected relay should be wired into the mains live wire if the heater is driven by the mains supply. The heater is connected to the middle contact and the live wire should be connected to the normally closed contact. The relay should have a 5V coil which is connected to the digital output from the Mini-Module. The other end of the coil should be connected to the GND pin on the Mini-Module. The relay will switch over when +5V is applied across the coil.

The serial port on the Mini-Module can be used to log the time that the heater was turned on for, how much energy was used and many other statistics

back to a P.C. or factory monitoring system.

In order to make the code more readable, the two digital I/O lines that are used to operate the relay and read the thermostat have been given names using the constant declaration in Modula-2. The code to drive the digital lines is contained in the standard library module ADIO that is contained on the distribution discs. The boolean variable is used to report the current state of the thermostat so that the heater is not turned on if it is already on or off it is already off.

The procedure Delay is imported from the standard Module-2 library System. This routine will cause the code to wait for 1 second before it reads the thermostat again. During this time this process is sleeping enabling other processes that are running to use its time slot, thus removing wasted time

The first part of the main program is setting up the digital channels and the variable OnOff to its initial value. The remainder of the code is an endless loop that is repeated until the power is turned off (or ESC is hit on the P.C. keyboard). If the value read back from the thermostat is true and the heater is currently off (onOff FALSE) then the heater has to be turned on. OnOff is set to TRUE to say that the heater is on. The heater will stay on until the thermostat returns a FALSE value. This will cause the procedure HeaterOff to be called to turn the heater off. The temperature that the boiler is kept at is set by the thermostat and so the Mini-Module is unable to tell how hot the boiler is at any time. The next example enables the

Mini-Module to read the temperature of the boiler at any time.

```
MODULE Boiler;
(*
This program will monitor the state of
a thermostat. When the thermostat falls
below a set temperature the heater will
be turned on. Once the temperature
rises above the temperature set by the
thermostat the heater is turned off.
```

```
The heater is connected to a relay
which is driven by channel 0 on the
Mini-Module. The thermostat on the
boiler is connected to digital channel
1 on the Mini-Module.
*)
```

```
FROM ADIO IMPORT OutChan, InChan,
ReadChan, WriteChan;
FROM System IMPORT Delay;
```

```
CONST heater      = 0;
thermostat        = 1;
```

```
VAR OnOff : BOOLEAN;
```

```
PROCEDURE HeaterOn;
(*
This procedure will turn the heater on
by closing the relay connected to chan-
nel 0.
*)
```

```
BEGIN
WriteChan(heater,TRUE)
END HeaterOn;
```

```
PROCEDURE HeaterOff;
(*
This procedure will turn the heater off
by opening the relay driven by channel
0.
*)
```

```
BEGIN
WriteChan(heater,FALSE)
END HeaterOff;
```

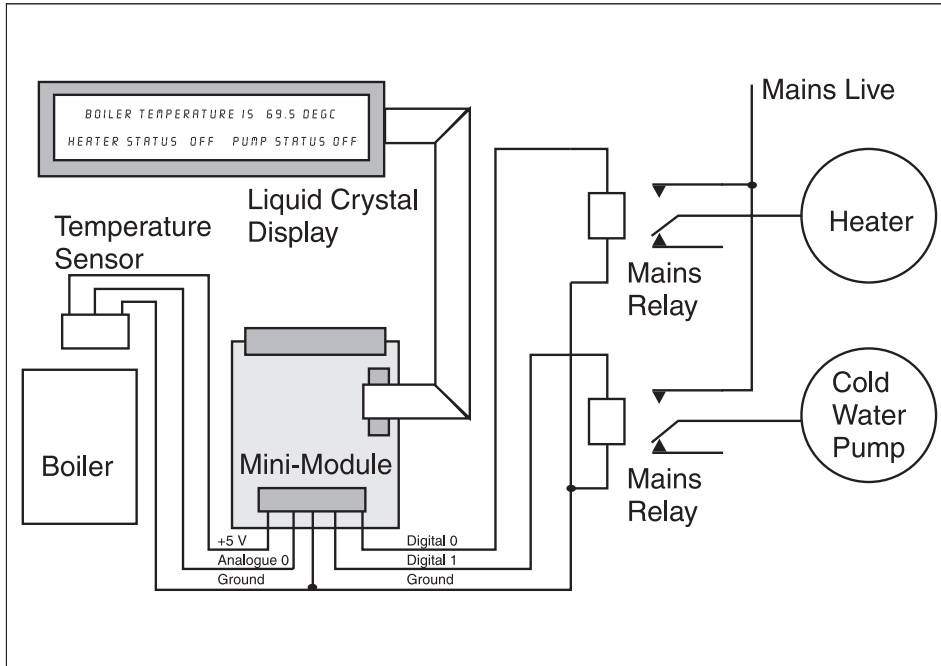
```
(*
Main program
*)
```

```
BEGIN
(* Set up directions of digital lines*)
OutChan(heater);
InChan(thermostat);
(* The heater starts off turned off *)
OnOff := FALSE;
(* The main loop of the program*)
```

```
LOOP
IF (ReadChan(thermostat)) AND
NOT(OnOff) THEN
HeaterOn;
OnOff := TRUE
ELSIF NOT(ReadChan(thermostat))
AND OnOff THEN
HeaterOff;
OnOff := FALSE
END;
Delay(100)
END
```

```
END Boiler.
```

Temperature Control of a Boiler using a Sensor



This example will also control the temperature of a boiler. Instead of using a thermostat to switch on the heater this example goes one step further. A solid state temperature sensor is used (LM35DZ) to allow the Mini-Module to measure the actual temperature of the boiler. Depending on the temperature the Mini-Module can turn on a cold water pump or a heater. The Mini-Module can also display the current temperature of the boiler on a standard alphanumeric liquid crystal display.

The temperature sensor that is used in this example is the LM35DZ. This is a three pin transistor type device. Two power signals are supplied to it, 0V and 5V. At 0 degC the voltage given out by the device is 0V. As the temperature increases the voltage generated will rise by 10 mV per degC rise in temperature. The device can be used between 0 and 100 degC. They are accurate to within 1 degC over the full range of temperatures.

The code listed below will keep the water temperature in the boiler to between 50 and 80 degC. The heater will be turned off when the temperature reaches 75 degC. If the temperature of the water continues to rise above 80 degC the water pump is turned on to let in some cold water until the temperature in the boiler falls to 70 degC.

The heater relay should be connected to digital channel 0 on the Mini-Module. The pump relay is connected to digital channel 1 and the temperature sensor output is connected to analogue input 0 on the Mini-Module. An alphanumeric LCD is connected to the LCD

port on the Mini-Module, this is used to display the current temperature, to one decimal place, in the boiler and the status of the heater and water pump. The temperature of the water is calculated in the function procedure CalcTemp. This takes into account the number of bits that the temperature is measured to and the analogue reference value on the Mini-Module.

```

MODULE Boiler;
(*
This program will keep the temperature of the water in a boiler to between 50 and 80 degC. The current temperature and the status of the heater and water pump are displayed on a 2 x 40 alphanumeric Liquid Crystal Display.
*)

FROM ADIO IMPORT OutChan, WriteChan, Actrl, Adc;
FROM InOut IMPORT Open, Close, WriteReal, WriteString, WriteLn;
FROM Video IMPORT Cls, Cursor, Tab, pathout;
FROM InOut IMPORT Open, Close;
FROM RealIO IMPORT points;
FROM System IMPORT Delay;

CONST
  heater = 0; (* heater on channel 0 *)
  pump = 1; (* pump on channel 1 *)
  sensor = 0; (* Sensor on Adc 0 *)
  Vref = 2.56.; (*2.56 V reference *)
  VdegC = 0.01; (*10 mV/degC *)

VAR temp : REAL;
  reading, path : INTEGER;
  heatOn, pumpOn : BOOLEAN;

PROCEDURE Heater(OnOff : BOOLEAN);
(*
This procedure is used to turn the heater on or off. If OnOff is FALSE then the heater is turned OFF. If OnOff is TRUE the heater is turned ON.
*)
BEGIN
  IF OnOff THEN
    WriteChan(heater, TRUE);

```

```

    heatOn := TRUE
  ELSE
    WriteChan(heater, FALSE);
    wheaten := FALSE
  END
END Heater;

PROCEDURE Pump(OnOff : BOOLEAN);
(*
This procedure is used to turn the pump on or off. If OnOff is FALSE then the pump is turned OFF. If OnOff is TRUE the pump is turned ON.
*)
BEGIN
  IF OnOff THEN
    WriteChan(pump, TRUE);
    pumpOn := TRUE
  ELSE
    WriteChan(pump, FALSE);
    pumpOn := FALSE
  END
END Pump;

PROCEDURE CalcTemp(value : INTEGER) : REAL;
(*
This procedure calculates the temperature in the boiler from the value returned from the sensor. Value is between 0 and 256.
*)
VAR temp : REAL;
BEGIN
  temp := FLOAT(value) * 256 / Vref;
  temp := FLOAT(temp) / VdegC;
  RETURN temp
END CalcTemp;

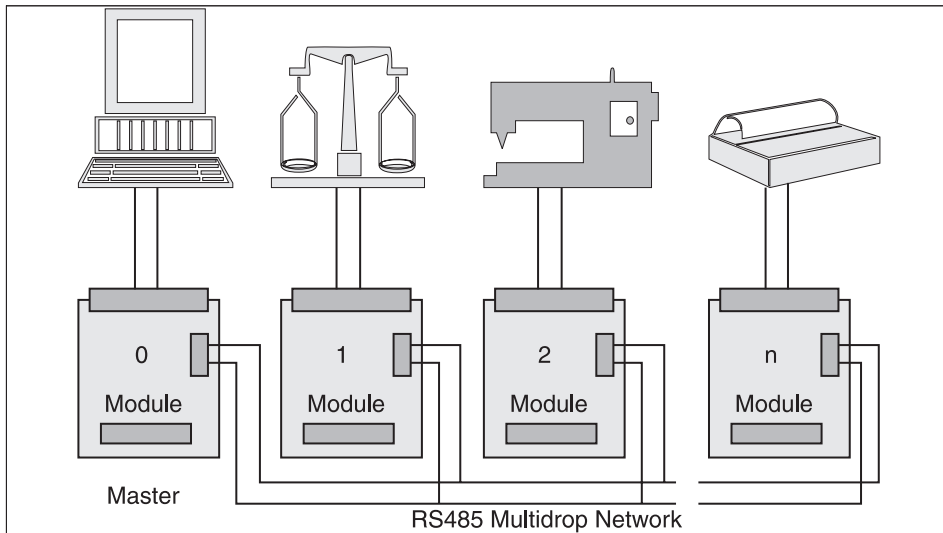
(* The main program *)

BEGIN
  (* Setup channels and variables *)
  OutChan(0);
  OutChan(1);
  Actrl(0); (* Channel 0 *)
  heatOn := FALSE;
  pumpOn := FALSE;
  path := Open("LCD");
  pathout := path;
  (* Setup the LCD display *)
  Cls; Cursor(FALSE);
  points := 1; (* 1 decimal place *)
  Tab(5,1);
  WriteString("Boiler temperature is ");
  Tab(32,1);
  WriteString("degC");
  Tab(3,2);
  WriteString("Heater Status");
  Tab(24,2);
  WriteString("Pump Status");

  LOOP
    reading := Adc();
    temp := CalcTemp(reading);
    IF (temp < 50.0) AND NOT(heatOn) THEN
      Heater(TRUE)
    END;
    IF (temp > 80.0) AND NOT(pumpOn) THEN
      Pump(TRUE)
    END;
    IF (temp > 75.0) AND heatOn THEN
      Heater(FALSE)
    END;
    IF (temp < 80.0) AND pumpOn THEN
      Pump(FALSE)
    END;
    Tab(28,1);
    WriteReal(temp,5);
    Tab(17,2);
    IF wheaten THEN
      WriteString("ON ")
    ELSE
      WriteString("OFF")
    END;
    Tab(36,2);
    IF pumpOn THEN
      WriteString("ON ")
    ELSE
      WriteString("OFF")
    END;
    Delay(100) (* Repeat every sec *)
  END;
  Close(path)
END Boiler.

```

Multi-drop RS485 Network



This example shows how easy it is to connect our module products together and use the RS485 network support to communicate between them. The network is a master slave arrangement, where there can only be one master and up to 252 slaves. Each of the slaves can be addressed individually or all together. As the network uses a differential protocol, it is ideal for use in noisy environments such as those commonly found in factories. All the features of the Module can be used at each station so all the other examples included in this could be used with the network if required. The line drivers that are provided can only drive up to 32 inputs so if more Modules are required on the network repeaters must be used.

The master will receive all data that is transferred on the network. A slave will only respond to data that carries its individual station number. If two slaves are given the same station number the network will generate errors. All slaves will receive a broadcast message but none of them will respond. If you want to have a terminal on the network that monitors all the activity you can program a second module as a master as long as you do not try to write on the network with the second master.

The Modules are connected together using two core cable with a characteristic impedance of about 100 Ohms. The positive signals are connected together and the negative signals are connected on each Module. The first and the last Module on the line must have terminating resistors fitted if the line is long to prevent signal reflections.

The code examples use the network library NetW to communicate between one master and two slaves. The procedure ErrChk is used to report the error message for any errors on the network.

It is included in both the master and slaves programs but we have only included it here once for conciseness.

The network library and the driver are not part of the standard Module starter packs, they are available from your distributor using the sales order code K-275, RS485 Network Support.

```

MODULE Master;
IMPORT Terminal, NetW;
VAR string : ARRAY[0..255] OF CHAR;

PROCEDURE ErrChk;
VAR str : ARRAY[0..63] OF CHAR;
BEGIN
  IF NetW.error=0 THEN RETURN END;
  CASE NetW.error OF
    1 : str := 'Driver or descriptor
              Missing' |
    2 : str := 'Driver not initialized' |
    3 : str := 'No response from SLAVE' |
    4 : str := 'SLAVE not allowed this
              call' |
    5 : str := 'Network timed out during
              receive' |
    6 : str := 'Bad checksum' |
    7 : str := 'Message too long'
  ELSE
    str := 'Undefined error'
  END;
  Terminal.WriteString(str);
  Terminal.WriteLine;
  NetW.DeInit;
  HALT
END ErrChk;

BEGIN
  (* Initialize the network *)
  NetW.Init; ErrChk;
  NetW.Flush; ErrChk;
  (* We are the master *)
  NetW.Station(0); ErrChk;
  LOOP
    (* Enable station 8 *)
    NetW.Select(8);
    NetW.WriteString("Hello station 8
                      how are you today?");
    ErrChk;
    (* Get slave 8's response *)
    NetW.ReadString(string); ErrChk;
    Terminal.WriteString(string);
    Terminal.WriteLine;

    (* Enable station 7 *)
    NetW.Select(7);
    NetW.WriteString("Hello station 7
                      how are you today?");

```

```

ErrChk;
(* Get slave 7's response *)
NetW.ReadString(string); ErrChk;
Terminal.WriteString(string);
Terminal.WriteLine
END;
(* Deselect all stations *)
NetW.Select(0);
NetW.DeInit
END Master.

MODULE Slave8;
IMPORT Terminal, NetW;
VAR string : ARRAY[0..255] OF CHAR;

PROCEDURE ErrChk;
(* See master program *)
END ErrChk;

BEGIN
  NetW.Init; ErrChk;
  NetW.Station(8); ErrChk;
  (* We are station 8 *)
  NetW.Flush; ErrChk;
  LOOP
    (* Wait for some action *)
    REPEAT UNTIL NetW.Ready();
    (* Read message from master *)
    NetW.ReadString(string); ErrChk;
    Terminal.WriteString(string);
    Terminal.WriteLine;
    (* We respond with a message for
    the master *)
    NetW.WriteString("Slave 8 is very
                      well thank you");
    ErrChk
  END;
  NetW.DeInit
END Slave8.

MODULE Slave7;
IMPORT Terminal, NetW;
VAR string : ARRAY[0..255] OF CHAR;

PROCEDURE ErrChk;
(* See master program *)
END ErrChk;

BEGIN
  NetW.Init; ErrChk;
  NetW.Station(7); ErrChk;
  (* We are station 7 *)
  NetW.Flush; ErrChk;
  LOOP
    (* wait for some action *)
    REPEAT UNTIL NetW.Ready();
    (* Read message from master *)
    NetW.ReadString(string); ErrChk;
    Terminal.WriteString(string);
    Terminal.WriteLine;
    (* We respond with a message for
    the master *)
    NetW.WriteString("Slave 7 is very
                      well thank you");
    ErrChk
  END;
  NetW.DeInit
END Slave7.

```



**Cambridge Microprocessor
Systems Limited.**

17-18 Zone 'D',
Chelmsford Road Ind. Est.,
Great Dunmow,
Essex. U.K. CM6 1XG
Tel 01 371 875644
Fax 01 371 876077
E-mail cms@dial.pipex.com