



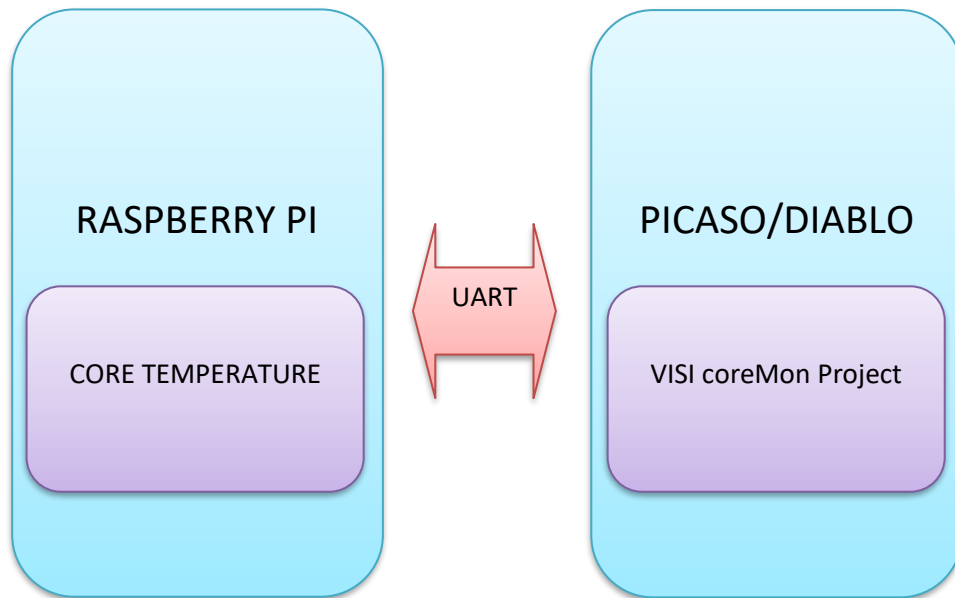
ViSi Simple Raspberry Pi Core Temperature Monitor

DOCUMENT DATE: **9th MAY 2020**
DOCUMENT REVISION: **1.1**



Description

UART/Serial communication with Raspberry Pi is one way to transport information from-and-to a Raspberry Pi and an external device. This application note demonstrates how to use the 4D Systems Intelligent display to show the core temperature.



Before getting started, the following are required:

- The target module can be a PICASO or a DIABLO16 display
Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.
- [4D Programming Cable](#) or [µUSB-PA5](#)
- [micro-SD \(µSD\)](#) memory card
- [Workshop 4 IDE](#) (installed according to the installation document)
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.
- [Raspberry Pi module](#)

Content

Description	2
Content.....	3
Application Overview	3
Setup Procedure	4
<i>Setup WiringPi on Raspberry Pi</i>	<i>4</i>
<i>Raspberry Pi – 4D Systems Intelligent Setup Test</i>	<i>4</i>
Create a New Project.....	4
Design of the Project	4
ViSi coreMon Project	5
Includes section	5
Initializing the micro-SD Card	5
Serial information monitor	5
Raspberry Pi Core Monitor	6
Building the C Project from terminal	8
Connect the Raspberry Pi to 4D Systems Intelligent Display	8
Run the ViSi Based Program	9
Proprietary Information	10
Disclaimer of Warranties & Limitation of Liability.....	10

Application Overview

Raspberry Pi is a credit-card sized single board computer that is capable of running a several Linux distribution images. The Linux images running on the Raspberry Pi is a stand-alone operating system that is capable of allowing internal processing and communication with external devices.

This application note demonstrates how to display the Raspberry Pi core temperature into a Workshop IDE-ViSi based project. The core temperature is retrieved using system calls and then sent via UART to the display module. This application note is limited to retrieving the core temperature to simplify the demonstration.



Setup Procedure

Setup WiringPi on Raspberry Pi

The procedure included in this section is referenced from the official [WiringPi Download and Install](#) procedure. The content of this section intends to consolidate setup procedures necessary to establish serial communication between Raspberry Pi and 4D Systems intelligent displays using the aforementioned library.

[ViSi Setting Up the Raspberry Pi for ViSi Projects](#)

Raspberry Pi – 4D Systems Intelligent Setup Test

After setup of the Raspberry Pi is made, it is necessary to test the functionality of the inter-connectivity. To test the UART communication please refer to the link below.

[ViSi Setting Up the Raspberry Pi for ViSi Projects](#)

Create a New Project

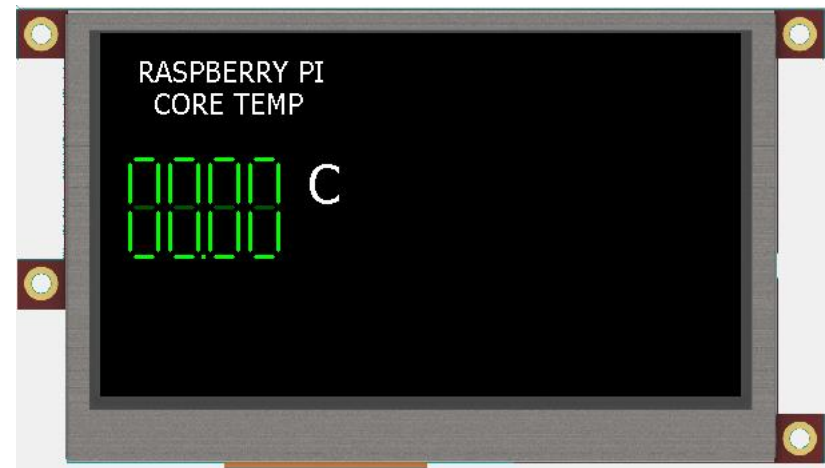
For instructions on how to create a new **Designer** project, please refer to the section “**Create a New Project**” of the application note

[Designer Getting Started - First Project](#)

For instructions on how to create a new **ViSi** project, please refer to the section “**Create a New Project**” of the application note.

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

Design of the Project



The demo project uses two types of widgets that are available in the Workshop IDE, namely - text labels and LED digit object. The LED digit will be used to display the core temperature.

The LED digit's properties can be changed to display only whole numbers. For this application note, the temperature of Raspberry Pi will be displayed only up to the first decimal point.

****Any changes in the placement of the LED Digits should reflect on the project code. Failure to update the placement position of the LED digits will result to object being shown at the wrong position.**

ViSi coreMon Project

This project uses the com0 UART port of the 4D Systems intelligent display module. The following set of statement shows the content of the project 'coreMon'. The project was made to be used with this application note for demonstration purposes.

Includes section

Workshop IDE-ViSi environment have statements that are automatically loaded when starting a project. The last line was manually included. The LEDDIGITSDISPLAY.inc was included to support the LED digits object. Without the include file, errors will be experienced during build of the project.

```
#platform "uLCD-43DT"
#inherit "4DGL_16bitColours.fnc"
#inherit "VisualConst.inc"
#inherit "coreMonConst.inc"
#inherit "LEDDIGITSDISPLAY.inc"
```

Initializing the micro-SD Card

Projects that make use of images and graphics will require a micro-SD for storage. The following set of statements initializes the micro-SD and load the images into a variable 'handl'.

```
putstr("Mounting...\n");
if (!disk:=file_mount())
while(!disk:=file_mount())
    putstr("Drive not mounted...\n");
    pause(200);
    gfx_cls();
    pause(200);
wend
endif
gfx_cls();
hdl := file_loadImageControl("coremon.dat", "coremon.gci", 1);
var numx[3]:= [0,0];
```

Serial information monitor

Receiving the information sent from the Raspberry Pi is fairly easy. The following set of statements constantly monitors and retrieves valid information from COM0 using the serin() function .

Each time the function finish executing the resulting value read is saved in a variable 'm'. Whenever the result of serin() is invalid, a value '-1' is received on variable m. Knowing the value of invalid results, we can now filter the incoming serial information.

```

com_Setbaud(CO0,11520);

img_Show(hnd1, ileddigits1); // show all digits at 0, only do this once
ledDigitsDisplay(0, ileddigits1+1, 20, 4, 3, 30, 0);
img_Show(hnd1, istatictext3);
img_Show(hnd1, istatictext1);
img_Show(hnd1, istatictext2);
var m, count;
repeat
  m := serin();
  if(m != -1)
    if(m == '&')
      count:=0;
      while(count < 2)
        m := serin();
        if(m != -1)
          numx[count++] := m;
        endif
      wend
      numx[2] := numx[0]*100 + numx[1]*10;

      img_Show(hnd1, ileddigits1); // show all digits at 0, only do this once
      ledDigitsDisplay(numx[2], ileddigits1+1, 20, 4, 3, 30, 0);

    endif
  endif
forever
endfunc

```

A user defined serial communication protocol is used in this application note. The following defines the protocol utilized.

Qualifying char	Whole number	Decimal number
'&'	Numx[0]	Numx[1]

For additional information on how create a user defined communication protocol for other application and use. Please refer to the following application note.

[Custom RS-485 Communication Protocol](#)

The content of the application note is not limited to rs-485 networks. The information on data packet generation is included with brief explanations for use and generation.

Raspberry Pi Core Monitor

In this section a simplified explanation on retrieving core information is presented. C programs in Raspberry Pi are fairly easy to create, build and run.

```

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

#include <wiringPi.h>

#include <wiringSerial.h>

int main ()
{
  int display ;

  char ch;

  FILE *data;

  int count = 1;

  int temp[3];

```

```
unsigned int nextTime ;

if ((display = serialOpen ("/dev/ttyAMA0", 115200)) < 0){
    fprintf(stderr, "Unable to open serial device: %s\n", strerror(errno)) ;
    return 1 ;
}

printf ("\nSuccessfully opened serial port") ;

if (wiringPiSetup () == -1) {
    fprintf(stdout, "Unable to start wiringPi: %s\n", strerror(errno)) ;
    return 1 ;
}

printf ("\nSuccessfully opened wiringPi") ;

nextTime = millis () + 300 ;

while(1){
    if (millis () > nextTime)  {
        system("vcgencmd measure_temp > data.txt");
        data = fopen("data.txt","r");
        while((ch = fgetc(data)) != '=');    // read file until '=' is reached
        while((ch = fgetc(data)) != '.'){
            temp[count++] = ch - 0x30;
            fflush (stdout) ;
        }
    }
}
```

```
    }

    temp[1] = (temp[0]*10) + temp[2];
    ch = fgetc(data);
    temp[2] = ch - 0x30;
    temp[0] = '&';
    count = 0;
    printf("\nCORE TEMPERATURE: %d.%d", temp[1], temp[2]);
    serialPutchar(display,temp[0]);
    serialPutchar(display,temp[1]);
    serialPutchar(display,temp[2]);

    fclose(data);
    nextTime += 300 ;
    delay(5000);
}

return 0 ;
}
```

Building the C Project from terminal

To build the c-project coreMon, use the command below. The command is in using the wiringPi format. After building the c-project, change the resulting coreMon project output file permission. To do this, use the commands below.

```
/home/pi$ gcc coreMon.c -o coreMon -O3 -Wall -I/usr/local/include -
Winline -pipe -L/usr/local/lib -lwiringPi -lwiringPiDev -lpthread -lm
```

```
/home/pi$ sudo chmod +x coreMon
```

Finally, to run the project, use the command below.

```
/home/pi$ sudo ./coreMon
```

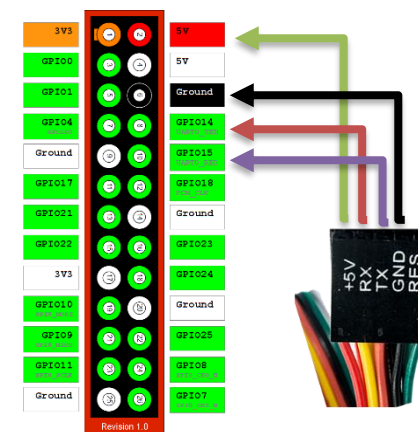
After successfully, building and compiling the simple program the image below will be seen on the raspberry monitor.

```
pi@raspberrypi ~/coreMon $ sudo ./coreMon
Successfully opened serial port
Successfully opened wiringPi
CORE TEMPERATURE: 3.6
CORE TEMPERATURE: 36.6
CORE TEMPERATURE: 36.6
CORE TEMPERATURE: 36.6
CORE TEMPERATURE: 36.6
```

Connect the Raspberry Pi to 4D Systems Intelligent Display



Connect the pins of the Raspberry Pi and 4D Systems Intelligent Display Module following the connection guide below.



Run the ViSi Based Program

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section “**Run the Program**” of the application note

[ViSi Getting Started - First Project for Picaso and Diablo16](#)

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expresses or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.