4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Exclusively available from element14

4DPi-24-HAT

2.4" HAT Primary Display for the Raspberry Pi

Quick Start Guide

What's Included

- ☐ 4DPi-24-HAT Display Module
- ☐ Quick Start Guide

Supporting Documents

Product brief, data sheet, CAD step model, simplified Step model, 3D-PDF model, schematic diagram and application note links are available at www.4dsystems.com.au

NOTE: Please refer to the 4D Systems website for the latest copy of 4DPi-24-HAT Data Sheet

Specifications

- Universal 2.4" Primary Display for the Raspberry Pi.
- Compatible with Raspberry Pi A+, B+ and Raspberry Pi 2.
- 240x320 pixel Resolution, RGB 65K true to life colours, TFT Screen with integrated 4-wire Resistive Touch Panel.
- Display full GUI output / primary output, just like a monitor connected to the Raspberry Pi
- High Speed 48MHz SPI connection to the Raspberry Pi, featuring SPI compression technology.
- Typical frame rate of 17 Frames per second (FPS), higher if image can be compressed further by the kernel. Lower if no compression is possible.
- Powered directly off the Raspberry Pi, no external power supply is required.
- On/Off or PWM controlled backlight, selectable by on board jumper.
- Module dimensions: 56.5 x 65 x 14.4mm weighing ~ 30g.
- Display Viewing Area: 36.72 x 74.75mm
- 4x mounting holes with 2.6mm diameter for mechanical mounting.
- RoHS and CE Compliant.

Let's get Started

Hardware Connection

The 4DPi-24-HAT is easily connected to a Raspberry Pi, by simply aligning the Female 40 way header with the Raspberry PI's Male 40 way header, and connecting them together – ensuring the aligning is correct and all pins are seated fully and correctly.

NOTE: *The 4DPi-24-HAT is supported only by the 40 way header, and therefore pressing on the touch screen may result in the 4DPi-24-HAT moving towards the Raspberry Pi, and therefore the circuitry touching the Raspberry Pi. This could result in damage to either product if a short circuit were to occur.* ***It is therefore highly encouraged to mount the display using standoffs (not included).***

Software Download / Installation

4D Systems has prepared a custom DMA enabled kernel for use with the Raspbian OperatingSystem, which is available for download as a single Debian Package. This can be installedover your existing Raspbian installation, or it can be applied over a fresh image.

This Debian package will only work for the Rasbian operating system, and will not work for other variants.

If you are starting from scratch, start from Step 1, else skip to step 3 if you already have a Raspbian Image and which to apply this kernel to that. Please note, it is impossible for us to know what you have done to your Raspbian image, if you are not installing from scratch – so if you encounter issues, please try and use a fresh image to determine if possible modifications are conflicting with our kernel release.

1. Download the latest Raspbian Image from the Raspberry Pi website:

   http://downloads.raspberrypi.org/raspbian_latest

2. Load the Raspberry Pi image onto a SD card, using the instructions provided on the Raspberry Pi website for Linux, Mac or PC:

   http://www.raspberrypi.org/documentation/installation/installing-images/README.md

3. Insert the SD card into the Raspberry PI. **Do not connect the 4DPi-24-hat yet.** You will need an external monitor / keyboard / network connection, else simply a network connection to the Pi and the rest can be done over an SSH connection. Start up the Pi with at minimum an Ethernet connection connected.

4. Either log into the Raspberry Pi from your keyboard/monitor using the standard 'pi' and 'raspberry' credentials, else SSH into your raspberry PI and log in via your SSH session.

5. Once logged into your Raspberry Pi, you will need to download and install the kernel which supports the 4DPi-24-HAT. The following step requires 'root' access. To gain super user access, type the following command on terminal/shell/SSH.

```
# sudo su
```

6. To download and install files, enter the following commands in terminal/shell /SSH to download the kernel from the 4D Systems Server:

```
# wget http://www.4dsystems.com.au/downloads/4DPi/
      4DPi-24-HAT/4DPi-24-HAT_kernel_R_1_0.tar.gz

# tar -xzvf 4DPi-24-HAT_kernel_R_1_0.tar.gz -C /
```

   By default the package is configured for Raspberry Pi2. To use the kernel on Raspberry Pi1, modify the content of /boot/config.txt.

```
# nano /boot/config.txt
```

   Find and modify the following line:

```
kernel=kernel7_hat.img
```

   change to:

```
kernel=kernel_hat.img
```

   Save the changes made to **/boot/config.txt.**

7. Booting the Raspberry Pi directly to Desktop GUI will require modification of the **/etc/rc.local** file.

```
# sudo nano /etc/rc.local
```

   Add the following command line, just before the 'exit 0' line.

```
sudo -u pi FRAMEBUFFER=/dev/fb1 startx &
```

8. Shutdown the Raspberry Pi safely, and remove the power.

9. Connect the 4DPi-24-HAT to the Raspberry Pi, and reapply power. The terminal should begin to show on the 4DPi-24-HAT, and will be ready to use once the Raspberry Pi has booted.

NOTE: *Please refer to the Datasheet on the 4D Systems website for the latest instructions as the information listed in the QSG is subject to change.*

NOTES

www.4dsystems.com.au

Sales : sales@4dsystems.com.au
Tech Support : www.4dsystems.com.au/support

1. Appendix 1 – Code Examples – Push Buttons

1.1. Example for communicating to Push Buttons, for C:

```
// test program to read state of buttons on 4D Systems 4DPi displays

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>

#define LCD4DPI_GET_KEYS _IOR('K', 1, unsigned char *)

void print_keys(int fd)
{
    unsigned char keys;

    if (ioctl(fd, LCD4DPI_GET_KEYS, &keys) == -1)
    {
        perror("_apps ioctl get");
    }
    else
    {
        printf("Keys : %2x\n", keys);
    }
}

int main(int argc, char *argv[])
{
    char *file_name = "/dev/fb1";
    int fd;

    fd = open(file_name, O_RDWR);
    if (fd == -1)
    {
        perror("_apps open");
        return 2;
    }

    print_keys(fd);
    printf("Ioctl Number: (dec)%d  (hex)%x\n", LCD4DPI_GET_KEYS, LCD4DPI_GET_KEYS);

    close (fd);
    return 0;
}
```

1.2. Example for communicating to Push Buttons, for Python:

```
#!/usr/bin/python
import array, fcntl
from time import sleep
# test program to read state of buttons on 4D Systems 4DPi displays

#LCD4DPI_GET_KEYS = -2147202303

_IOC_NRBITS   =  8
_IOC_TYPEBITS =  8
_IOC_SIZEBITS = 14
_IOC_DIRBITS  =  2
_IOC_DIRMASK    = (1 << _IOC_DIRBITS) - 1
_IOC_NRMASK     = (1 << _IOC_NRBITS) - 1
_IOC_TYPEMASK   = (1 << _IOC_TYPEBITS ) - 1
_IOC_NRSHIFT = 0
_IOC_TYPESHIFT = _IOC_NRSHIFT+_IOC_NRBITS
_IOC_SIZESHIFT = _IOC_TYPESHIFT+_IOC_TYPEBITS
_IOC_DIRSHIFT  = _IOC_SIZESHIFT+_IOC_SIZEBITS
_IOC_NONE = 0
_IOC_WRITE = 1
_IOC_READ = 2

def _IOC(dir, type, nr, size):
#  print 'dirshift {}, typeshift {}, nrshift {}, sizeshift
{}'.format(_IOC_DIRSHIFT, _IOC_TYPESHIFT, _IOC_NRSHIFT, _IOC_SIZESHIFT)
  ioc = (dir << _IOC_DIRSHIFT ) | (type << _IOC_TYPESHIFT ) | (nr << _IOC_NRSHIFT )
| (size << _IOC_SIZESHIFT)
  if ioc > 2147483647: ioc -= 4294967296
  return ioc
#def _IO(type, nr):
#  return _IOC(_IOC_NONE,  type, nr, 0)
def _IOR(type,nr,size):
  return _IOC(_IOC_READ,  type, nr, size)
#def _IOW(type,nr,size):
#  return _IOC(_IOC_WRITE, type, nr, sizeof(size))

LCD4DPI_GET_KEYS = _IOR(ord('K'), 1, 4)
buf = array.array('h',[0])

print 'Press Top & Bottom buttons simultaneously to exit'

with open('/dev/fb1', 'rw') as fd:

  while True:
    fcntl.ioctl(fd, LCD4DPI_GET_KEYS, buf, 1) # execute ioctl call to read the keys
    keys = buf[0]

    if not keys & 0b00001:
      print "KEY1" ,
    if not keys & 0b00010:
      print "KEY2" ,
    if not keys & 0b00100:
      print "KEY3" ,
    if not keys & 0b01000:
      print "KEY4" ,
    if not keys & 0b10000:
      print "KEY5" ,

    if keys != 0b11111:
      print
    if keys == 0b01110: # exit if top and bottom pressed
      break

    sleep(0.1)
```

1.3. Example for Shutdown and Reset buttons, for C

```
// test program to Shutdown or Restart Pi using buttons on 4D Systems 4DPi displays

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>

#define LCD4DPI_GET_KEYS _IOR('K', 1, unsigned char *)

int get_keys(int fd, unsigned char *keys)
{
    if (ioctl(fd, LCD4DPI_GET_KEYS, keys) == -1)
    {
        perror("_apps ioctl get");
       return 1;
    }
    *keys &= 0b11111;
    return 0;
}

int main(int argc, char *argv[])
{
    char *file_name = "/dev/fb1";
    int fd;
    unsigned char key_status;

    fd = open(file_name, O_RDWR);
    if (fd == -1)
    {
        perror("_apps open");
        return 2;
    }

    key_status = 0b11111;
    while(key_status & 0b00001) // press key 1 to exit
    {
        if(get_keys(fd, &key_status) != 0)
             break;

        // printf("key_status: %x\n", key_status);

        if(!(key_status & 0b10000))
        {
            system("sudo shutdown -h now");
            break;
        }

        if(!(key_status & 0b01000))
        {
            system("sudo reboot");
            break;
        }

        sleep(0.1);
    }

    close(fd);
    return 0;
}
```

1.4. Example for Shutdown and Reset buttons, for Python

```
#!/usr/bin/python
import array, fcntl, os
from time import sleep
# test program to Shutdown or Restart Pi using buttons on 4D Systems 4DPi displays

#LCD4DPI_GET_KEYS = -2147202303

_IOC_NRBITS   =  8
_IOC_TYPEBITS =  8
_IOC_SIZEBITS = 14
_IOC_DIRBITS  =  2

_IOC_DIRMASK    = (1 << _IOC_DIRBITS) - 1
_IOC_NRMASK     = (1 << _IOC_NRBITS) - 1
_IOC_TYPEMASK   = (1 << _IOC_TYPEBITS ) - 1

_IOC_NRSHIFT = 0
_IOC_TYPESHIFT = _IOC_NRSHIFT+_IOC_NRBITS
_IOC_SIZESHIFT = _IOC_TYPESHIFT+_IOC_TYPEBITS
_IOC_DIRSHIFT  = _IOC_SIZESHIFT+_IOC_SIZEBITS

_IOC_NONE = 0
_IOC_WRITE = 1
_IOC_READ = 2

def _IOC(dir, type, nr, size):
#  print 'dirshift {}, typeshift {}, nrshift {}, sizeshift
{}'.format(_IOC_DIRSHIFT, _IOC_TYPESHIFT, _IOC_NRSHIFT, _IOC_SIZESHIFT)
  ioc = (dir << _IOC_DIRSHIFT ) | (type << _IOC_TYPESHIFT ) | (nr << _IOC_NRSHIFT )
| (size << _IOC_SIZESHIFT)
  if ioc > 2147483647: ioc -= 4294967296
  return ioc
#def _IO(type, nr):
#  return _IOC(_IOC_NONE,  type, nr, 0)
def _IOR(type,nr,size):
  return _IOC(_IOC_READ,  type, nr, size)
#def _IOW(type,nr,size):
#  return _IOC(_IOC_WRITE, type, nr, sizeof(size))

LCD4DPI_GET_KEYS = _IOR(ord('K'), 1, 4)
#print 'ssd {} {:12} {:0>8x} {:0>32b}'.format(ssd1289, hex(ssd1289), ssd1289,
ssd1289)
buf = array.array('h',[0])

with open('/dev/fb1', 'rw') as fd:

  while True:
    fcntl.ioctl(fd, LCD4DPI_GET_KEYS, buf, 1) # execute ioctl call to read the keys
    keys = buf[0]

    if not keys & 0b00001:
      break

    if not keys & 0b10000:
      os.system("sudo shutdown -h now")
      break

    if not keys & 0b01000:
      os.system("sudo reboot")
      break;

    sleep(0.1)
```