NP4185 TFT Display Controller Datasheet

Version	Date	Remark
00	2018.12.21	First Issue
01	2020.1.9	Add backlight PWM Control

CONTENTS

1.OVERVIEW

P series product is with high-performance, low-cost display-specific controller, and it has different sizes for full color TFT display and has developed corresponding different control chips, so that the products can achieve the best performance and make the products have high competitiveness in the market. The display control is simple, user-friendly, high anti-interference ability, and has been widely used in industrial, smart new energy, smart home, elevator, consumer products and other fields.

2. FEATURE

2-1、NP4185 Controller

Core and Memory

- ➢ High-performance hardware processing core, the main frequency can reach 250MHz
- ➢ Internal 256K flash memory, 400K static random-access memory

Peripheral Interface

- ➢ Two high-speed QSPI interfaces, which can be externally connect to SPI FLASH or SPI PSRAM
- ➢ One RGB interface, support RGB 5-6-5 bit
- ➢ One LED PWM interface with 255-level brightness adjustable
- ➢ A standard 4-wire SPI user interface
- ➢ One user interface mode switching interface

Power Supplier

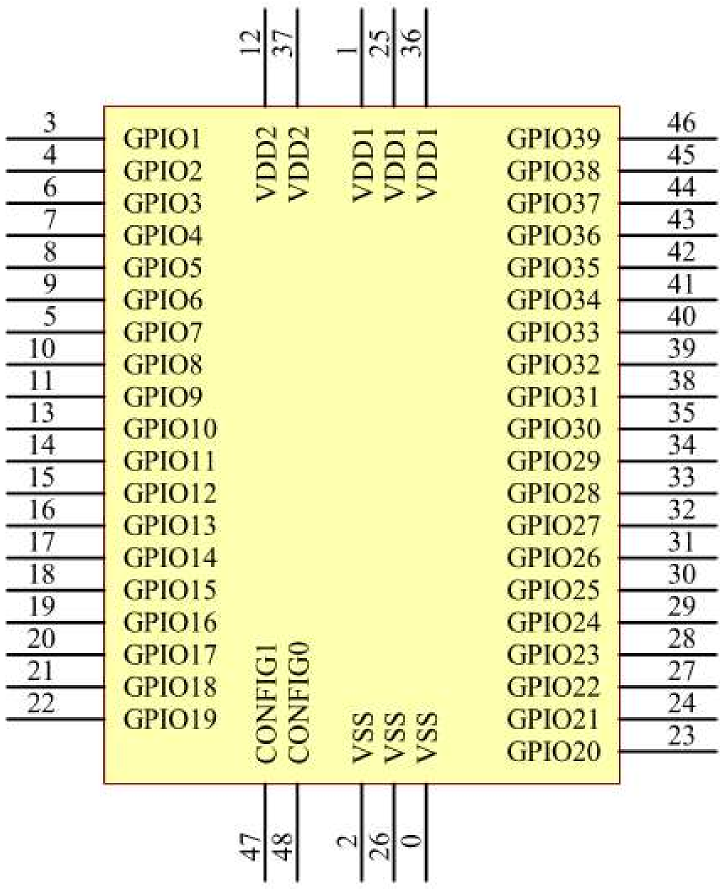
- ➢ VDD1 is 2.8V to 3.5V
- ➢ VDD2 is 1.1V to 1.3V

Package and Operating temperaturae

- ➢ QFN48
- ➢ Operating temperature：-20℃ to +70℃
- ➢ Storage temperature: -30℃ to +80℃

3. PIN DESCRIPTION

NP4185



NP4185 PIN Diagram

NRK3301 pin description:

PIN NO.	Name	I/O Type	Function	Other Function
1/25/26	VDD1	P	Power	Power Supply1
12/37	VDD2	P	Power	Power Supply2
2/26	VSS	P	Power	Power Ground
47/48	CONFIG0/1	I/O	CHIP CONFIG	
3~11/13~24/27~46	GPIO	I/O	GPIO	

4、ELETRICAL CHARACTERISTICS

Absolute Maximum Ratings

parameter	descrpation	min	max	unit
Tamb	Ambient Temperature	-30	+80	°C
Tstg	Storage temperature	-35	+85	°C
VDD1	Supply Voltage	-0.3	3.6	V
VDD2	Charger Voltage	-0.3	1.4	V
V_{GPIO}	3.3V IO Input Voltage	-0.3	3.6	V

PMU Characteristics

symbol		min	Typ.	max	unit	Test condition
VDD1	Voltage Input	3.0	3.3	3.6	V	
VDD2	Voltage Input	1.1	1.2	1.3	V	

IO Input/Output Electrical Logical Characteristics

IO input characteristics						
symbol		min	Typ.	max	unit	Test condition
V_{IL}	Low-Level Input Voltage	-0.3		0.3* VDDIO	V	VDDIO = 3.3V
V_{IH}	High-Level Input Voltage	0.7* VDDIO		VDDIO+0.3	V	VDDIO = 3.3V
IO output characteristics						
V_{OL}	Low-Level output Voltage			0.33	V	VDDIO = 3.3V
V_{OH}	High-Level output Voltage	3.0			V	VDDIO = 3.3V

5．SPI INTERFACE COMMUNICATION COMMAND PROTOCOL

// SPI timing request 1：when FLASH_SW=1，flash is used by customer’s MCU，SPI is 4-line timing, and the timing is the standard 4-line timing. At this time, you can use SPI FLASH directly，the SPI flash is readable and writable.

// SPI timing requirement 2: When FLASH_SW=0, the SPI timing is also a standard 4-wire timing but has no readback function, and the busy flag must be judged after the command/data is sent (output by the SPI_MISO pin). The sending command process is as follows :

// 1. First pull SPI_CS to low voltage

// 2. Send the command (12 configuration parameters), if you need to follow the display data, continue to send the display data

// 3. After sending the command, it is displayed that the controller needs to be processed. For this purpose, the busy flag status must be judged (SPI_MISO=1 is busy)

// When judging the busy flag, an exit mechanism is required to prevent the controller from waiting when the set parameters do not correspond to the written parameters.

// The exit mechanism is an appropriate delay. It is recommended that if the delay reaches 120ms and it is still busy judgment, then forced to exit. Please refer to SPI_Check_Busy()

// 4. Finally pull SPI_CS to high voltage

// Note 1: The FLASH_SW pin of the display controller is pulled down by default. If you don't need to operate the SPI FLASH, you can not process this pin

// Note 2: #define SPI_MISO_GET() in SPI.h GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_14) //For the user board, configure MISO as the busy flag input detection pin

//******************Important Notes on Sending Display Data to the Controller via the SPI Interface*****************//

// When the user's SPI interface directly send data to display controller (command 0x01 or 0x08), the maximum buffer of the controller's receiving area is 8192 bytes and the processing speed is very fast, but still need to pay attention to the following:

// 1. The max speed of the SPI clock is 10MHz, and the display data can be continuously written in bursts at or below 10MHz

// 2. Use the function of drawing points as little as possible. After all, it is slow to send data point by point with coordinate parameters, but there is a trick: if it is a continuous area, you can make the displayed data into a display area block in advance. One-time burst sending, very fast.

//***********************The following are several basic application functions**//

// Calculate the cache address according to the coordinate point

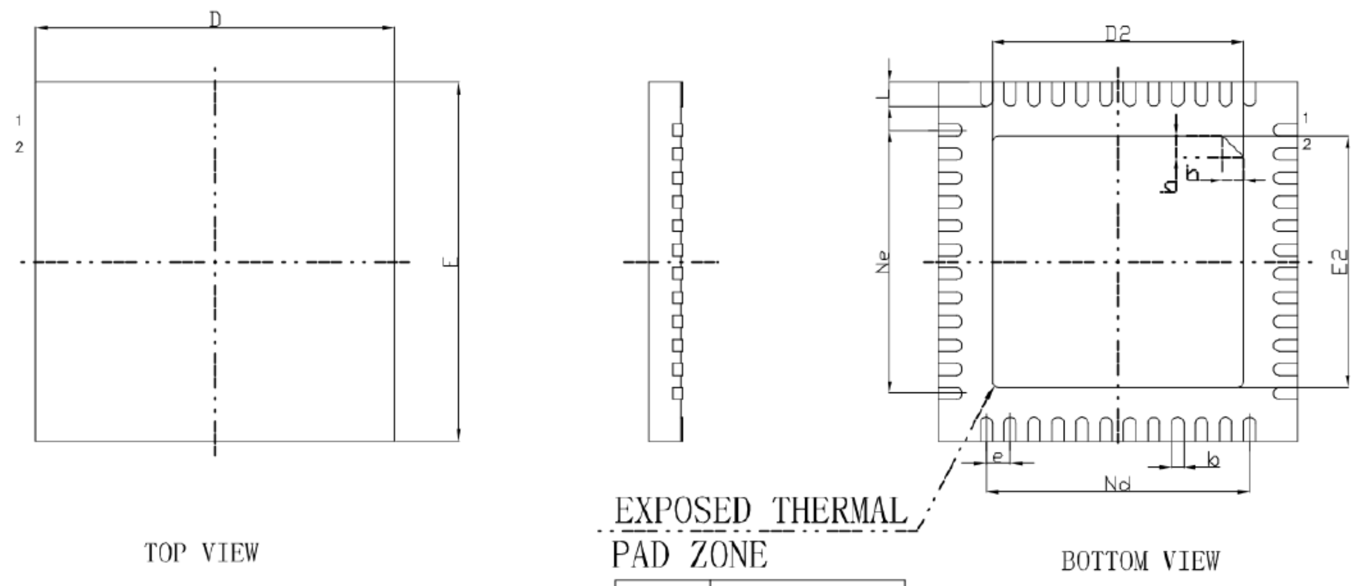
```
//uint32_t Get_XY2DramAddr(uint16_t X,uint16_t Y)
//{
//        uint32_t Dram_addr;
//        Dram_addr = ( X+Y*1536 )*2;     // Each line of bytes is fixed to 1536*2 bytes

//        return(Dram_addr);
//}
//-----------------------------------------------------------------------------------------------------------------------
```

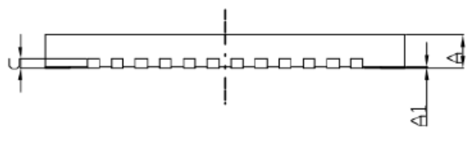
// command parameters are described as follows (12 parameters need to be sent at one time for each command, that is, sent in a function package): If you need to modify the screen brightness, you can modify it at the same time as the screen is refreshed

```
  //void  Set_NP4185_Comd(uint8_t  COMD,uint16_t  Dram_addr,uint16_t  Flash_addr,uint16_t  H_size,uint16_t  V_size,uint8_t
BL_PWM)
  //{
  //      SPI_SendByte( COMD );              // Command[0x01:User SPI data to Dram to display =
                                                          The user SPI directly writes data to display cache and refreshes the screen]
  //                                       //[0x04:Flash data to Dram to display = Read data from flash and write to display cache]
  //      SPI_SendByte( (uint8_t)(Dram_addr>>16) );       // Dram_address[19:16]   the user's SPI interface to start writing data
to a certain address in the display cache and directly refresh the screen

  //      SPI_SendByte( (uint8_t)(Dram_addr>>8) );        // Dram_address[15:8]   the user's SPI interface to start writing data
to a certain address in the display cache and directly refresh the screen
  //      SPI_SendByte( (uint8_t)(Dram_addr) );           // Dram_address[7:0]    the user's SPI interface to start writing data to
a certain address in the display cache and directly refresh the screen
  //      SPI_SendByte( (uint8_t)(Flash_addr>>16) );      // flash_address[19:16]   Specify the controller to read data from a
certain address of SPI FLASH for display. When the user SPI interface uses the display cache, this address is invalid (can be any
value)

  //      SPI_SendByte( (uint8_t)(Flash_addr>>8) );       // flash_address[15:8]    Specify the controller to read data from a
certain address of SPI FLASH for display. When the user SPI interface uses the display cache, this address is invalid (can be any
value)

  //      SPI_SendByte( (uint8_t)(Flash_addr) );          // flash_address[7:0]    Specify the controller to read data from a certain
address of SPI FLASH for display. When the user SPI interface uses the display cache, this address is invalid (can be any value)


  //      SPI_SendByte( (uint8_t)((V_size-1)>>8) );       // V_size-1[15:8]           Set the number of pixels for vertical operation
  //      SPI_SendByte( (uint8_t)(V_size-1) );            // V_size-1[7:0]            Set the number of pixels for vertical
operation
  //      SPI_SendByte( (uint8_t)((H_size-1)>>8) );       // H_size-1[15:8]           Set the number of pixels for vertical
operation
  //      SPI_SendByte( (uint8_t)(H_size-1) );            // H_size-1[7:0]            Set the number of pixels for vertical
operation
  //      SPI_SendByte( BL_PWM );                         // Backlight lightness[255~0] Set the backlight brightness
level, the brightest is 255, and the close is 0
  //}
  //------------------------------------------------------------------------------------------------------------------------
  // Quickly draw pot command

  //void Set_NP4185_DotComd(uint16_t X,uint16_t Y,uint16_t color)
  //{
  //      uint16_t Dram_addr;
  //      Dram_addr=Get_XY2DramAddr(X,Y);
  //      SPI_CS_LOW();
  //      SPI_SendByte( 0x08 );                           // Command[0x08:]
  //      SPI_SendByte( (uint8_t)(Dram_addr>>16) );       // Dram_address[19:16]   the user's SPI interface to start writing data
to a certain address in the display cache and directly refresh the screen

  //      SPI_SendByte( (uint8_t)(Dram_addr>>8) );        // Dram_address[15:8]    the user's SPI interface to start writing data
to a certain address in the display cache and directly refresh the screen
//	SPI_SendByte( (uint8_t)(Dram_addr) );		// Dram_address[7:0]	the user's SPI interface to start writing data to
a certain address in the display cache and directly refresh the screen


//	SPI_SendByte(color);	// Display font color
//	SPI_SendByte(color>>8);
//	SPI_Check_Busy();
//	SPI_CS_HIGH();
//}
//------------------------------------------------------------------------------------------------------------------------
// Busy flag

//uint8_t SPI_Check_Busy(void)
//{
//	uint8_t ucErrTime=0;
//	while(SPI_MISO_GET())
//	{
//		Delay_Xms(1);
//		ucErrTime++;
//		if(ucErrTime>250)	// suggest bigger than 120ms
//			return 1;
//	}
//	return 0;
//}
```

6. CHIP PACKAGE SIZE

封装尺寸 QN48Package Outline (6mm x 6mm)





SIDE VIEW

SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	0. 50	0.55	0. 60
A1	0	0.02	0. 05
b	0. 15	0.20	0. 25
c	0. 10	0.15	0. 20
D	5. 90	6. 00	6. 10
D2	4. 10	4.20	4. 30
e	0. 40BSC		
Ne	4. 40BSC		
Nd	4. 40BSC		
E	5. 90	6. 00	6. 10
E2	4. 10	4.20	4. 30
L	0. 35	0.40	0. 45
h	0. 30	0. 35	0. 40
L/F载体尺寸 (MIL)	177*177		