上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

S800板第二次实验

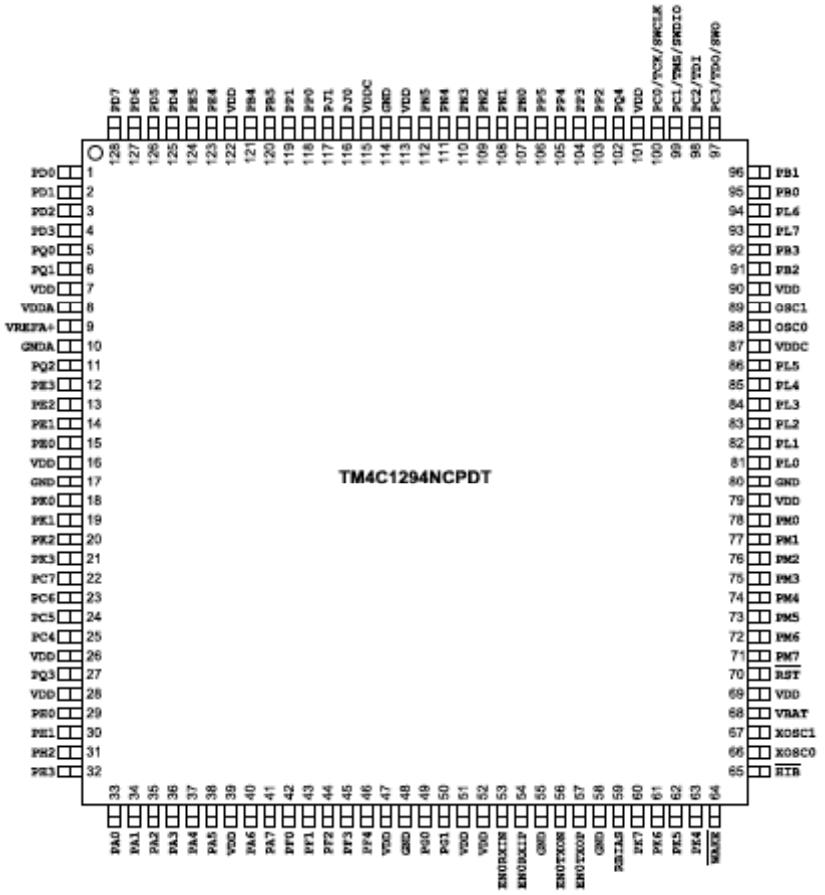
准备知识的学习

- 系统控制
- I2C

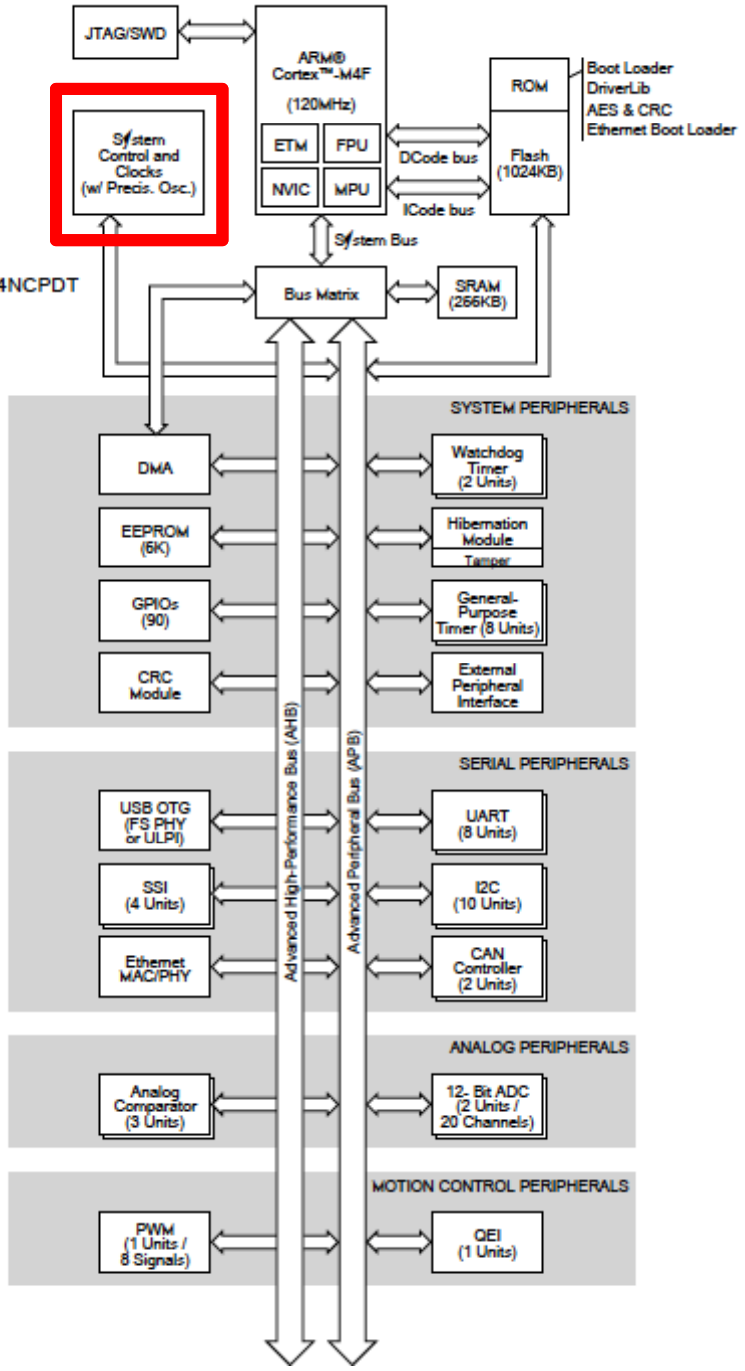
系统控制

System Control

High-Level Block Diagram



TM4C1294NCPDT



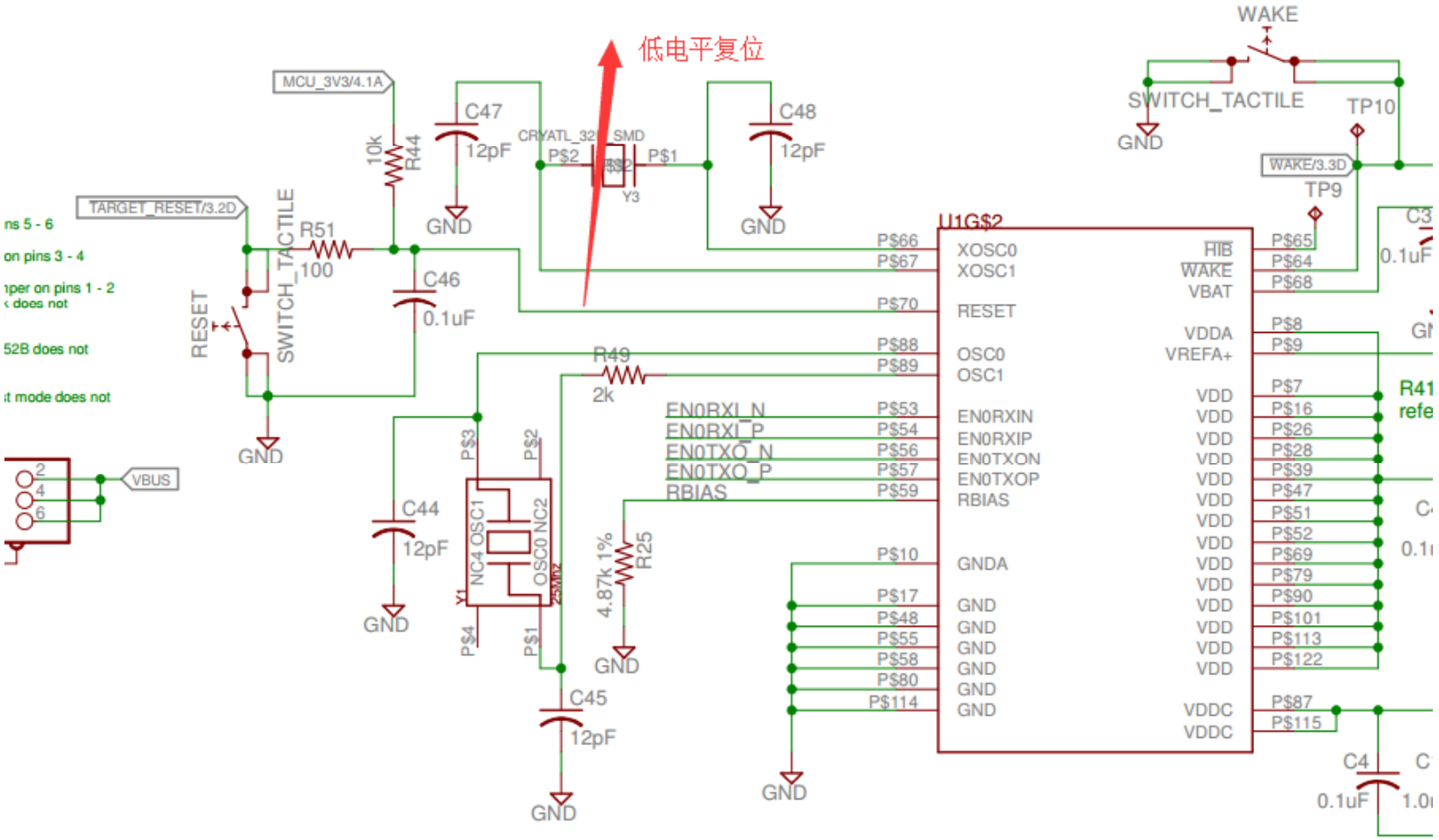
System Control

- System control configures the overall operation of the device and provides information about the device.
  - **reset control**
  - NMI operation
  - power control
  - **clock control**
  - low-power modes

Reset Control

- 复位源 Reset Sources
  - 上电复位Power On Reset (POR)
  - 外部管脚复位 (RST Pin)
  - Vdda或VDD掉电复位
  - 软件复位
  - 看门狗 (Watchdog)复位
  - 冬眠模组事件 (Hibernation Module)
  - 主振荡失败复位 (MOSC Failure)

S800板上的复位电路



Clock Control

- 系统时钟源(**SYSCLK**)
  - **Precision Internal Oscillator (PIOSC)**
    - CPU内部提供的16M振荡源在POR时使用，未校正时精度有限

Parameter	Parameter Name	Min	Nom	Max	Unit
F_{PIOSC}	Factory calibration, 0 to +105°C: Internal 16-MHz precision oscillator frequency variance across voltage and temperature range when factory calibration is used	-	-	±4.5%	-
	Factory calibration, -40°C to <0°C	-	-	±10%	-
	Recalibration: Internal 16-MHz precision oscillator frequency variance when recalibration is used at a specific temperature	-	-	±1%	-
T_{START}	PIOSC startup time[a]	-	-	1	µs

  - **Main Oscillator (MOSC)**
    - 外部主振荡源，外部4-25M的振荡信号输入，可采用石英晶体振荡器作为高精度时钟源使用

S800板上的时钟源

- CPU内部
  - 低精度的16M振荡器HSI（正负50%误差），可以用于不需要精确定时的场合，不能用于同步通讯如USART，CAN，ETHERNET等。
- 红板上共有两个外部晶体振荡器供选择：
  - 一个为外部25M高精度无源晶振HSE，为正常使用时提供外部时钟信号；
  - 一个为32.768K无源晶振LSE，主要用于低功耗或电池供电时提供外部时钟信号。
- 使用说明：
  - 如果不使用PLL倍频电路，则只能使用25M的HSE或16M的HSI作为时钟。
  - 如果使用PLL倍频电路，则可以升频到480M后再分频作为CPU时钟使用，最大可接受频率为120M时钟。
  - 频率越高，则速度越快，功耗越大。

时钟配置

- 系统时钟在不同工作模式下(RUN，SLEEP，DEEP SLEEP)使用不同的时钟源；
- 一般可以选择PIOSC或MOSC作为系统时钟来源，在本系统中因为硬件固定，只能为16M PIOSC及25M MOSC；
- 可以通过PLL倍频模式将系统时钟调整到不同系统时钟，系统频率最大120M
  - 在16M或25M模式下，可以将PLL频率配置为320M或480M均可
  - 然后从此PLL频率经过（1-1024）分频再到想要的系统频率SYSCLK
- 示例程序（对TM4C1290系列只能使用SysCtlClockFreqSet函数）
  - *SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |SYSCTL_OSC_MAIN |SYSCTL_USE_PLL |SYSCTL_CFG_VCO_480), 20000000);*
    //倍频到480M，再分频到20M，分频必须为整数倍
  - *SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |SYSCTL_OSC_MAIN |SYSCTL_USE_OSC), 25000000);*

系统时钟设置函数SysCtlClockFreqSet

⌘ 驱动库手册：P487

功能	配置系统时钟。
原型	uint32_t **SysCtlClockFreqSet**(uint32_t **ui32Config**, uint32_t **ui32SysClock**)
参数	**ui32Config**为设备时钟所需要的配置。 **ui32SysClock**为所需要的处理器频率。
说明	该函数配置设备的主系统时钟，如：输入频率，振荡器源，是否使能PLL以及系统分频。参数ui32Config是几个不同值的逻辑“或”，它们分别都被分成多个组，其中在每个组中只能选择一个值。 外部晶振频率的选择为如下参数之一：**SYSCTL_XTAL_25MHz**..... 振荡器源的选择为如下参数之一：**SYSCTL_OSC_MAIN** 使用外部晶振或振荡器，**SYSCTL_OSC_INT** 使用内部16MHz振荡器，..... 系统时钟源选择：**SYSCTL_USE_PLL**是使用PLL输出作为系统时钟，**SYSCTL_USE_OSC** 是选择一个振荡器作为系统时钟。 PLL倍频选择：**SYSCTL_CFG_VCO_480**、**SYSCTL_CFG_VCO_320**

函数在实验二中的应用

- //use internal 16M oscillator, HSI

```
ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_16MHZ
|SYSCTL_OSC_INT |SYSCTL_USE_OSC), 16000000);
```

PLL时钟设置

Crystal Frequency (MHz)	MINT (Decimal Value)	MINT (Hexadecimal Value)	N	Reference Frequency (MHz)[b]	PLL Frequency (MHz)
5	64	0x40	0x0	5	320
6	160	0x35	0x2	2	320
8	40	0x28	0x0	8	320
10	32	0x20	0x0	10	320
12	80	0x50	0x2	4	320
16	20	0x14	0x0	16	320
18	160	0xA0	0x8	2	320
20	16	0x10	0x0	20	320
24	40	0x28	0x2	8	320
25	64	0x40	0x4	5	320
5	96	0x60	0x0	5	480
6	80	0x50	0x0	6	480
8	60	0x3C	0x0	8	480
10	48	0x30	0x0	10	480
12	40	0x28	0x0	12	480
16	30	0x1E	0x0	16	480
18	80	0x50	0x2	6	480
20	24	0x18	0x0	20	480
24	20	0x14	0x0	24	480
25	96	0x60	0x4	5	480

缩写解释

- HSI：高速内部时钟
- HSE：高速外部时钟
- LSE：低速外部时钟
- PLL： Phase Locked Loop，锁相环

I2C接口

Inter-Integrated Circuit Interface

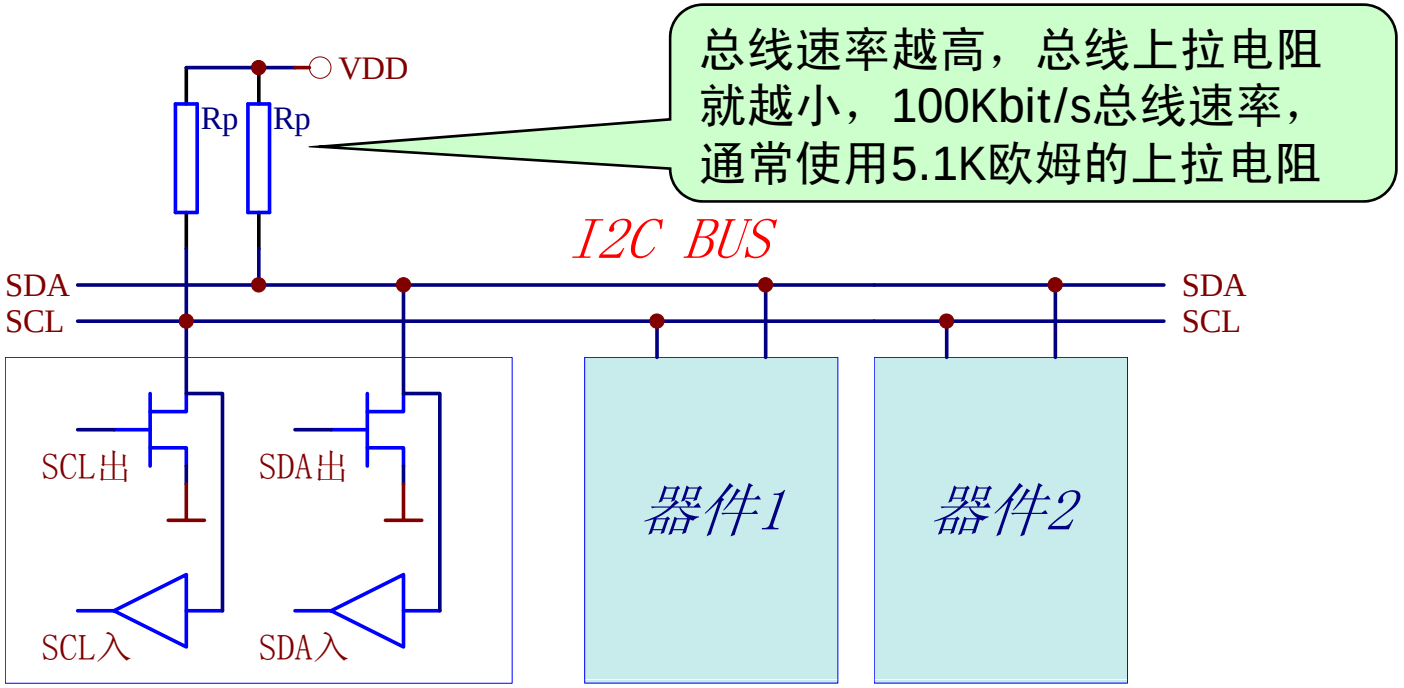
内部集成电路

I2C简介

- I2C是Philips公司开发的一种串行总线，用于IC器件之间的通信。
- I2C是一种简单的、低速的、两线式同步串行总线。
  - I2C总线采用两根线的设计(一根串行数据线SDA 和一根串行时钟线SCL)来提供双向的数据传输。
  - 数据线上的信号与时钟同步，并通过软件寻址识别每个器件，而不需要片选线。
  - 多个符合I^2C总线标准的器件都可以通过同一条I^2C总线进行通信，而不需要额外的地址译码器。
  - I^2C接口的标准传输速率为100Kbit/s，最高传输速率可达400Kbit/s。

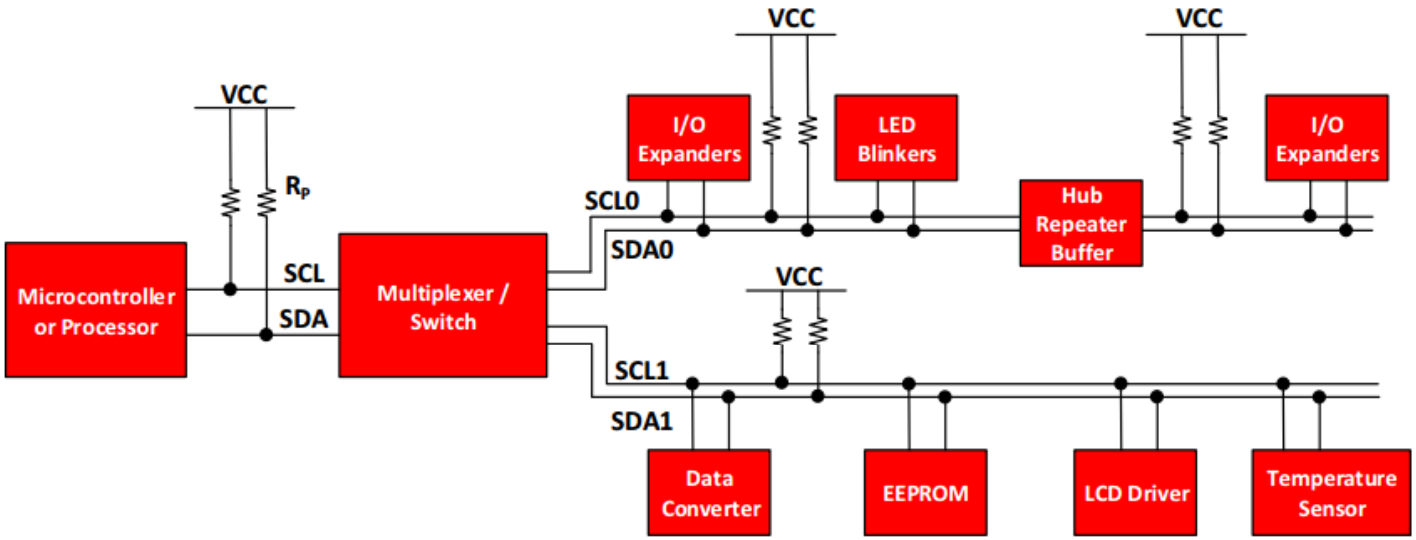
I2C电气连接

- I²C总线接口均为开漏或开集电极输出，因此需要为总线增加上拉电阻Rp。



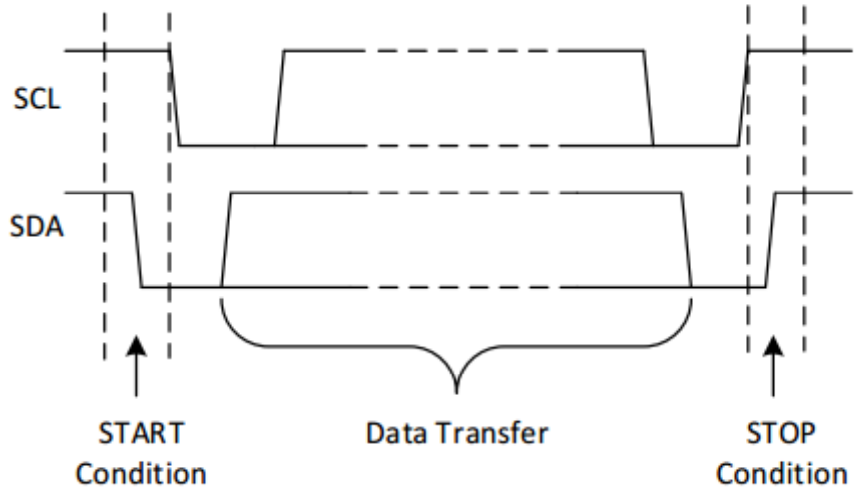
I2C连接方式

- I2C总线仅使用两个信号：SDA和SCL。SDA是双向串行数据线，SCL是双向串行时钟线。
- 系统中所有的微控制器和外围器件都将数据线SDA与时钟线SCL的同名引脚连在一起，总线上的所有节点都由器件引脚给定地址。



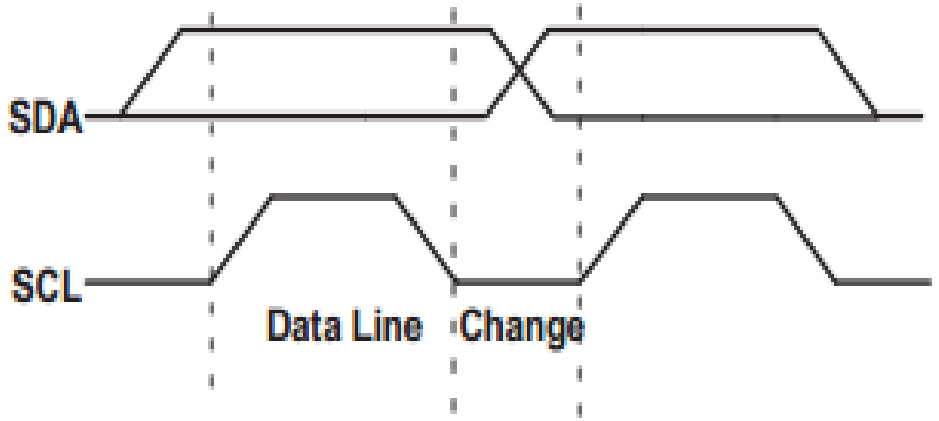
起止和停止条件

- I2C总线的协议定义了两种状态：起始和停止。
- 当SCL为高电平时，在SDA线上从高到低的跳变被定义为起始条件。
- 当SCL为高电平时，在SDA线上从低到高的跳变则被定义为停止条件。
- 总线在起始条件之后被看作为忙状态。
- 总线在停止条件之后被看作为空闲。



数据有效性

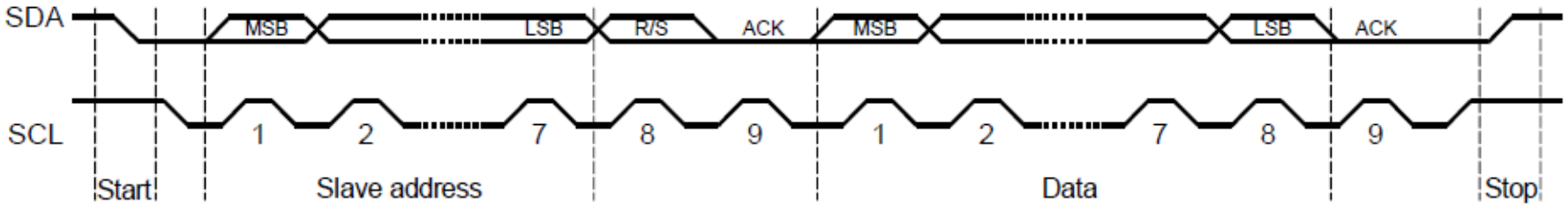
- 当SDA和SCL线为高电平时，总线为空闲状态。
- 在时钟SCL的高电平期间，SDA线上的数据必须保持稳定。
- SDA仅可在时钟SCL为低电平时改变。



数据传输

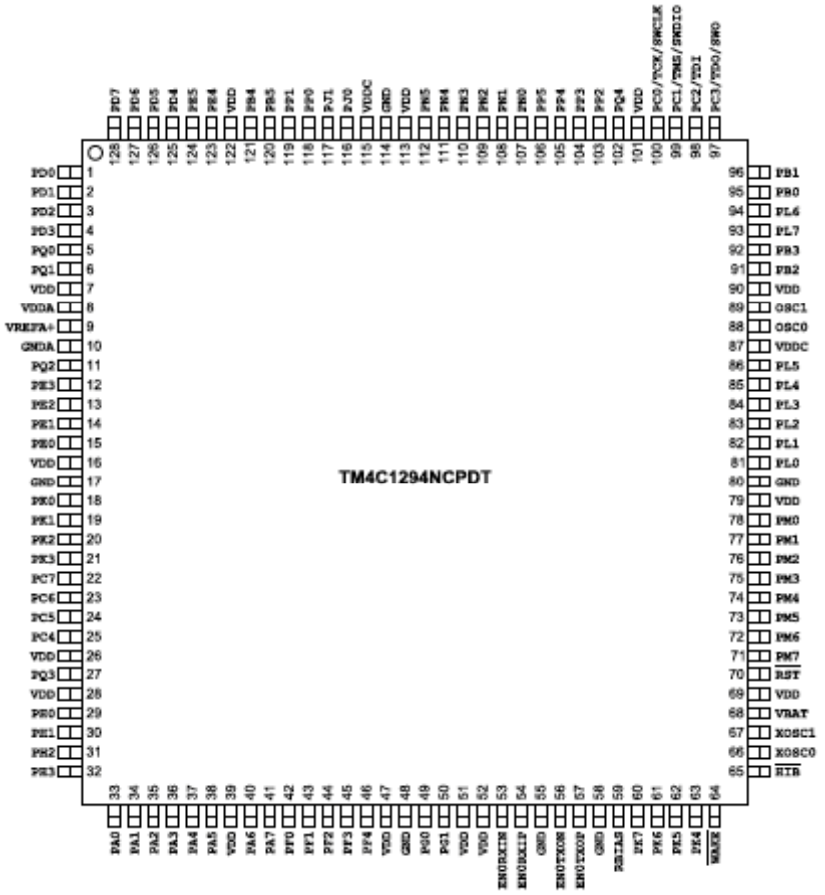
- I2C总线每次传输的数据长度为9位，包括8位数据位和1位应答位。
  - 每次SDA线上的每个字节必须为8位长，每个字节后面必须带有一个应答位。
  - 不限制每次传输的字节数。
  - 数据传输是MSB在前。
- 当接收器不能接受另一个完整的字节时，它可以将时钟线SCL拉到低电平，以迫使发送器进入等待状态。当接收器释放时钟SCL时继续进行数据传输。
- I2C总线可以构成多主数据传送系统，但只有带CPU的器件可以成为主器件。主器件发送时钟、启动位、数据工作方式，从器件则接收时钟及数据工作方式。
  - TM4C的每个I2C模块由主机和从机两个功能组成，并由唯一地址进行标识。

带有7位地址的数据格式

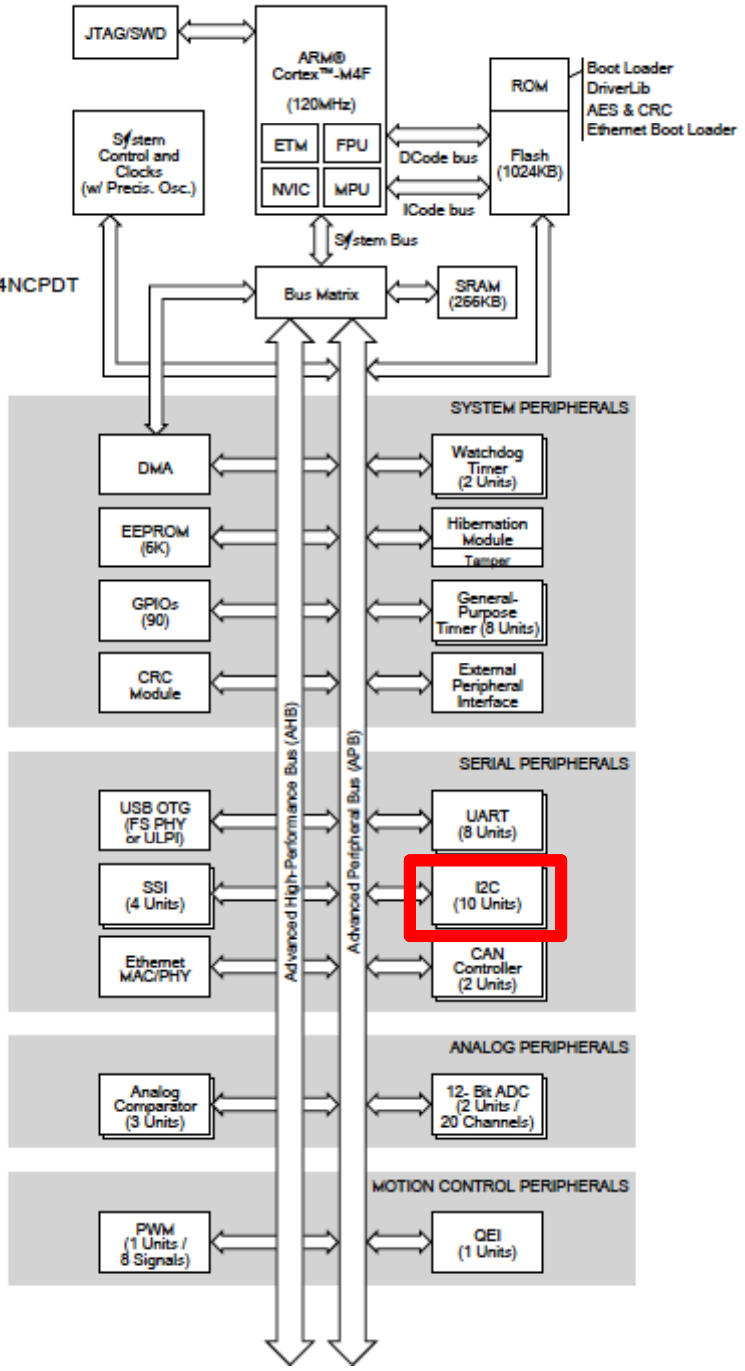


- 在达到开始条件之后，从机地址将被发送。
- 从机地址共7位，紧跟着的第8位是数据传输方向位（R/S位）。这个数据方向位决定了下一个操作是接收（高电平）还是发送（低电平），0表示传输（发送），1表示请求数据（接收）。
- 数据传输始终由主机产生的停止条件来终止。然而，通过产生重复的起始条件和寻址另一个从机（而无需先产生停止条件），主机仍然可以在总线上通信。因此，在这种传输过程中可能会有接收/发送格式的不同组合。
- 带有I2C总线的器件除了有从机地址（Slave Address）外，还有数据地址（也称子地址）。从机地址是指该器件在I2C总线上被主机寻址的地址，而数据地址是指该器件内部不同部件和存储单元的编址。

High-Level Block Diagram



TM4C1294NCPDT



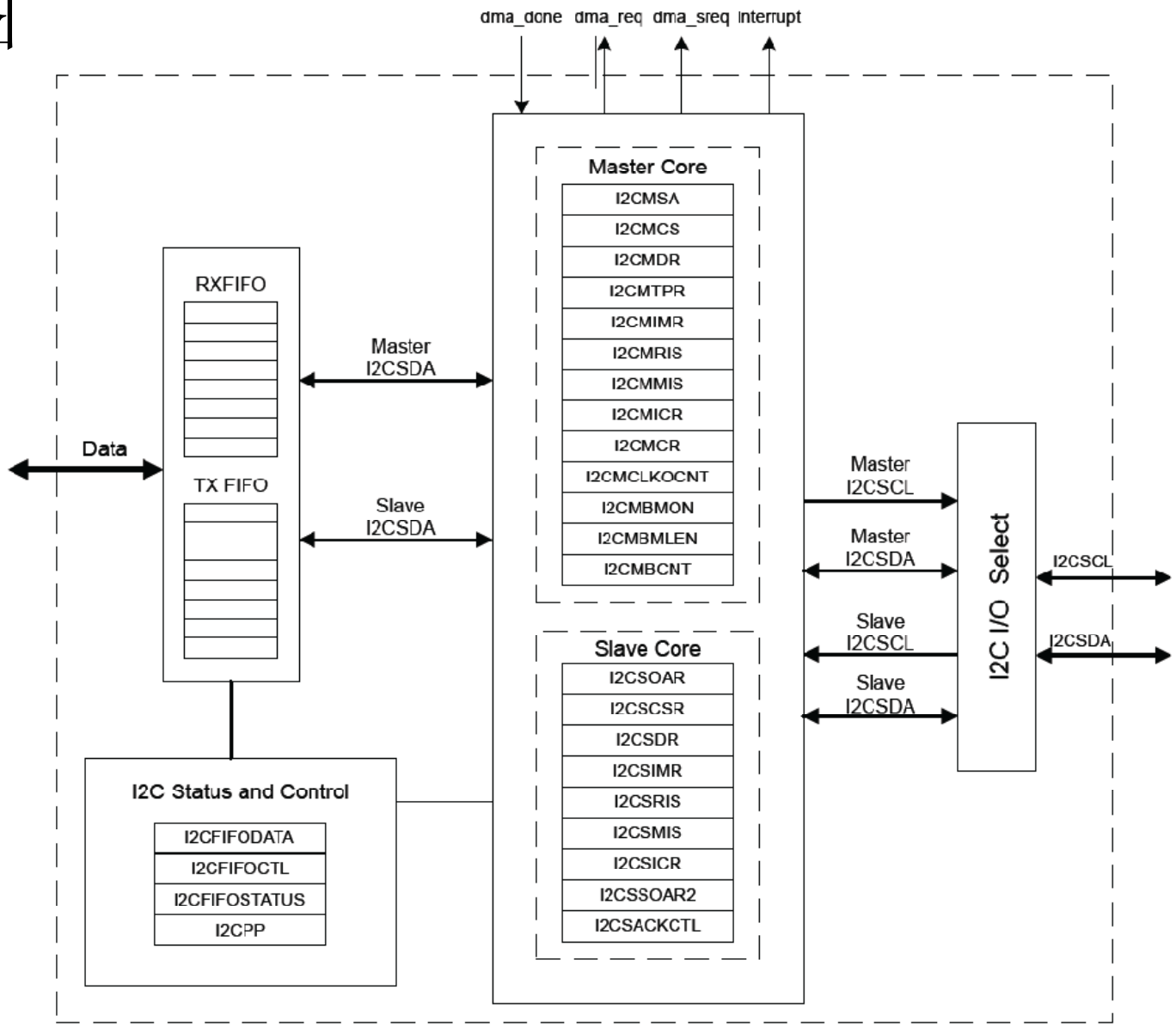
TM4C1294NCPDT微控制器上的I2C

- 微控制器上共配备了10个I2C模块，且微控制器具有与其他I2C总线上的设备交互（发送和接收）的能力。
- 每个I2C模块具有以下特点：
  - I2C总线上的设备可被配置为主机或从机；
    - 支持一个主机或从机发送和接收数据；
    - 同时支持主机和从机操作。
  - 4个I2C模式
    - 主机发送模式
    - 主机接收模式
    - 从机发送模式
    - 从机接收模式

TM4C1294NCPDT微控制器上的I2C

- 4种传输速度
  - 标准模式（100 Kbps）
  - 快速模式（400 Kbps）
  - 超快速模式（1 Mbps）
  - 高速模式（3.33 Mbps）
- 主机和从机中断的产生
  - 当主机发送或接收操作完成时（或因错误终止时），产生中断;
  - 当从机发送数据，或主机需要数据，或检测到起始或停止条件时，产生中断。
- 主机有仲裁和时钟同步，支持多主机，以及7位寻址模式。

I2C结构图

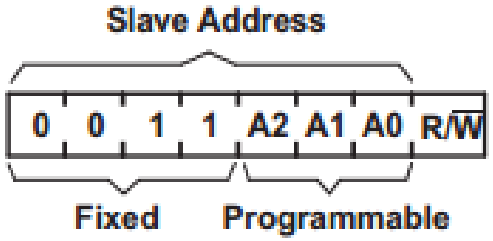

dma_done
dma_req
dma_sreq
interrupt
Master Core
I2CMSA
I2CMCS
I2CMDR
I2CMTPR
I2CMIMR
I2CMRIS
I2CMMIS
I2CMICR
I2CMCR
I2CMCLKOCNT
I2CMBMON
I2CMBMLEN
I2CMBCNT
RXFIFO
TX FIFO
Data
Master
I2CSDA
Slave
I2CSDA
Slave Core
I2CSOAR
I2CSCSR
I2CSDR
I2CSIMR
I2CSRIS
I2CSMIS
I2CSICR
I2CSSOAR2
I2CSACKCTL
I2C Status and Control
I2CFIFODATA
I2CFIFOCTL
I2CFIFOSTATUS
I2CPP
Master
I2CSCL
Master
I2CSDA
Slave
I2CSCL
Slave
I2CSDA
I2C I/O Select
I2CSCL
I2CSDA

I2C芯片

PCA9557

I2C芯片PCA9557

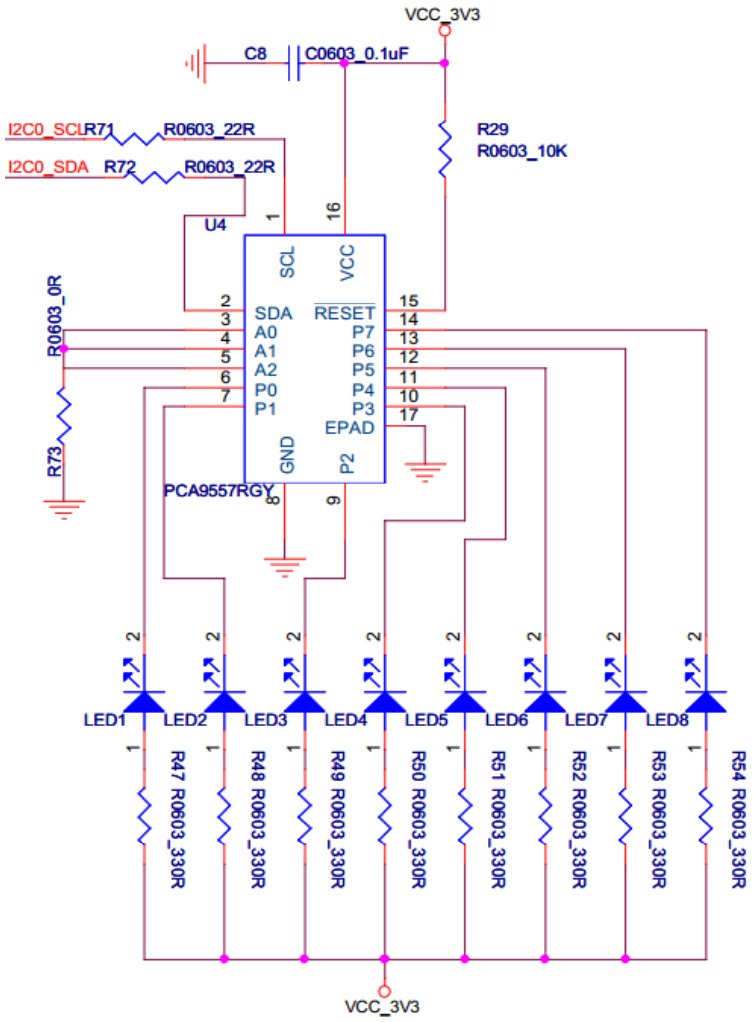
⌘地址定义



INPUTS			I^2C BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	24 (decimal), 18 (hexadecimal)
L	L	H	25 (decimal), 19 (hexadecimal)
L	H	L	26 (decimal), 1A (hexadecimal)
L	H	H	27 (decimal), 1B (hexadecimal)
H	L	L	28 (decimal), 1C (hexadecimal)
H	L	H	29 (decimal), 1D (hexadecimal)
H	H	L	30 (decimal), 1E (hexadecimal)
H	H	H	31 (decimal), 1F (hexadecimal)

详见PCA9557.pdf

S800蓝板上的PCA9557



- PCA9557为I2C转8位GPIO扩展芯片，在板上为LED驱动，低有效。即当管脚为低电平时，对应的LED亮。
- **PCA9557地址：0x18**

详见**S800**蓝板原理图

往PCA9557输出口寄存器写数据的时序

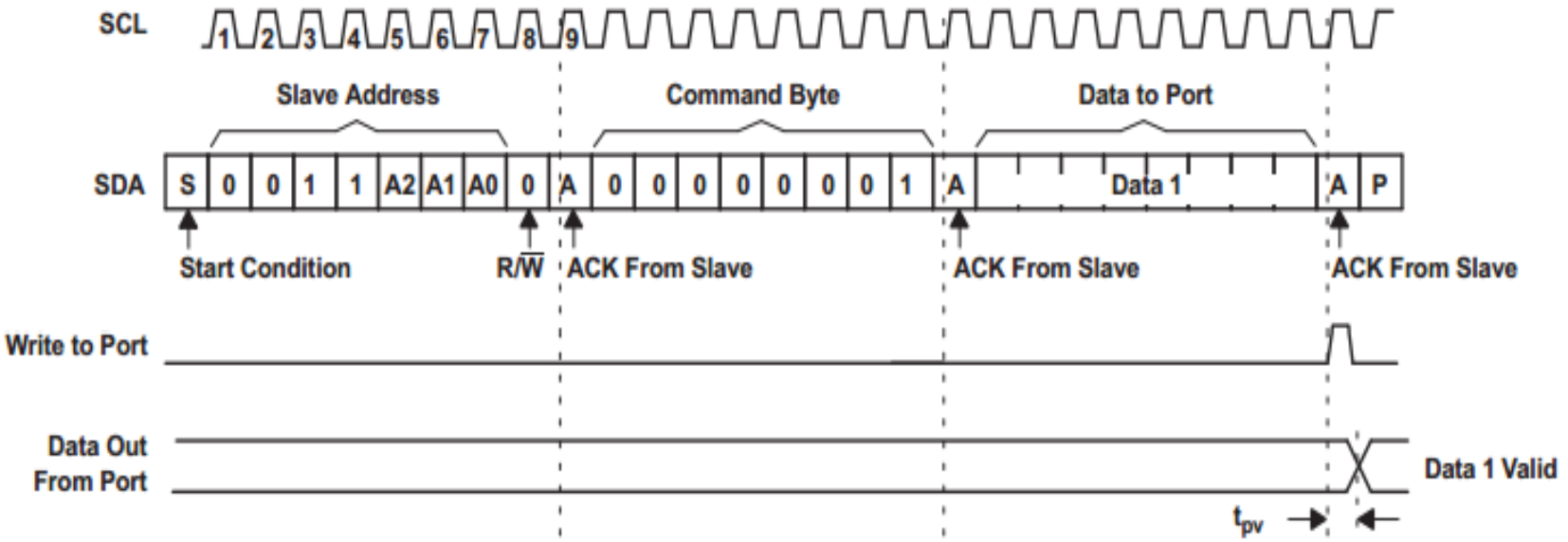


Figure 24. Write To Output Port Register

PCA9557编程

- PCA9557的使用分为两步：
  - PCA9557初始化，即将端口配置为输出。
  - 对PCA9557的输出端口赋值，例：赋0x0时，则8个LED全亮；当为0x0FF时，8个LED全灭。
- 固件库文件驱动方式从上图的对输出端口写数据可见，首先给I2C总线输出从设备即PCA9557地址，然后给出两字节数据，第一个字节为命令，第二个字节为数据。命令即输出端口01H，数据则为从0x00-0x0FF。
- 因此固件库使用如下：其中DevAddr为设备地址即0x18，RegAddr为输出端口地址即0x01，WriteData为写给输出端口的数据。

```
uint8_t I2C0_WriteByte(uint8_t DevAddr, uint8_t RegAddr, uint8_t
WriteData)
{
        uint8_t rop;
        while(I2CMasterBusy(I2C0_BASE)){};
        I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false);
        I2CMasterDataPut(I2C0_BASE, RegAddr);
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
        while(I2CMasterBusy(I2C0_BASE)){};
        rop = (uint8_t)I2CMasterErr(I2C0_BASE);
        I2CMasterDataPut(I2C0_BASE, WriteData);
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C0_BASE)){};
        rop = (uint8_t)I2CMasterErr(I2C0_BASE);
        return rop;
}
```

I2C芯片

TCA6424

I2C芯片TCA6424

⌘地址定义

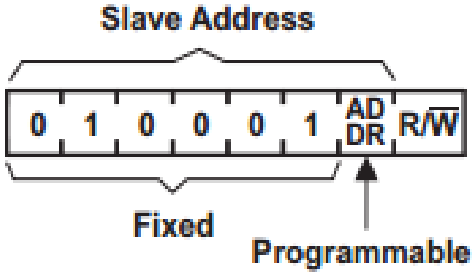


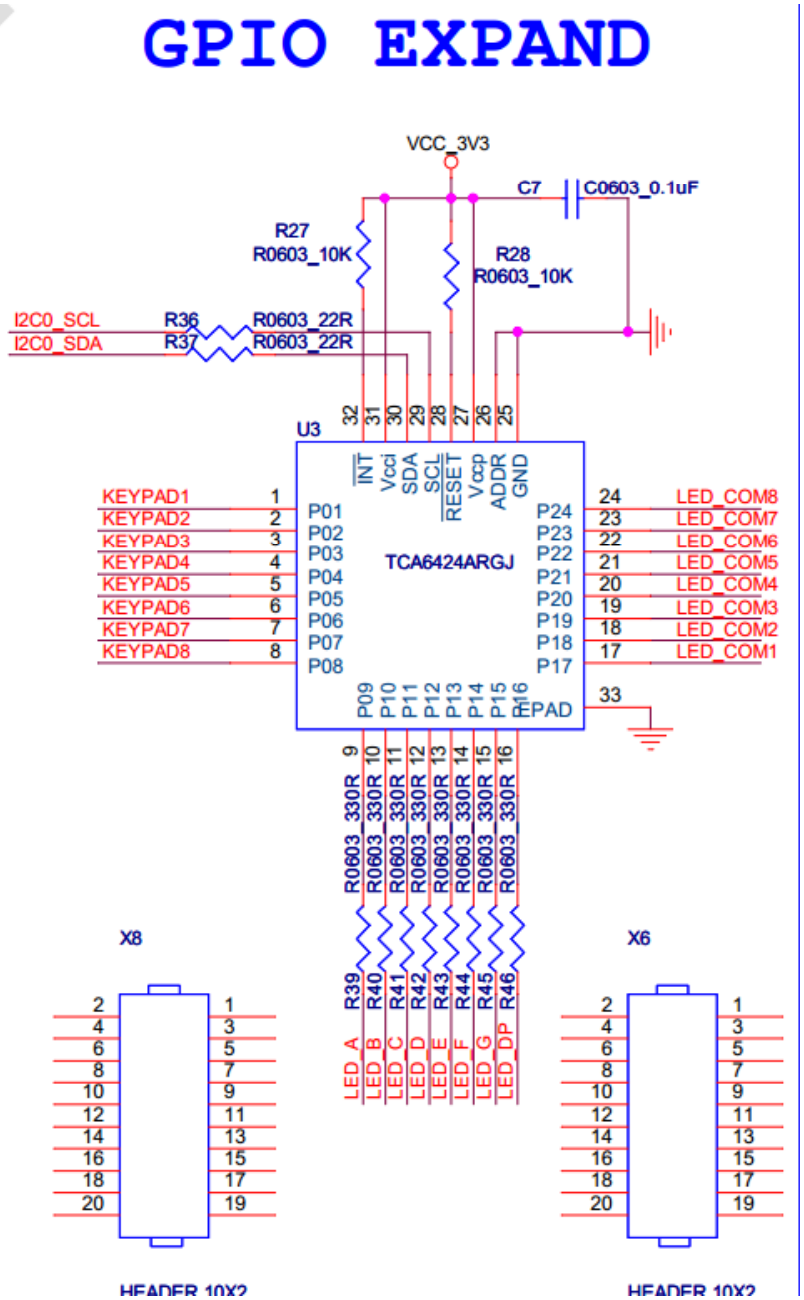
Table 3. Address Reference

ADDR	I²C BUS SLAVE ADDRESS
L	34 (decimal), 22 (hexadecimal)
H	35 (decimal), 23 (hexadecimal)

详见**TCA6424.pdf**

I2C芯片TCA6424

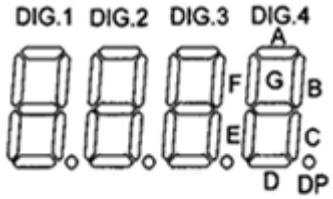
- **TCA6424地址：0x22**
- TCA6424为I2C转24位GPIO扩展芯片，分为3组，每组8位，在蓝板上：
  - P0为按键SW1-SW8；
  - P1为动态共阴数码管的脚位信号，高电平时点亮相应的笔划；
  - P2为动态共阴数码管的片选信号，当为高电平时，驱动对应的8050三极管导通，从而选通对应的位。



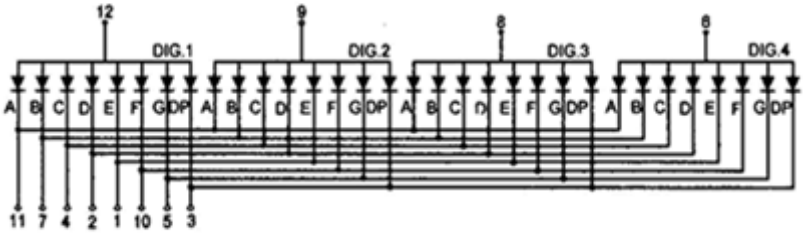
TCA6424编程

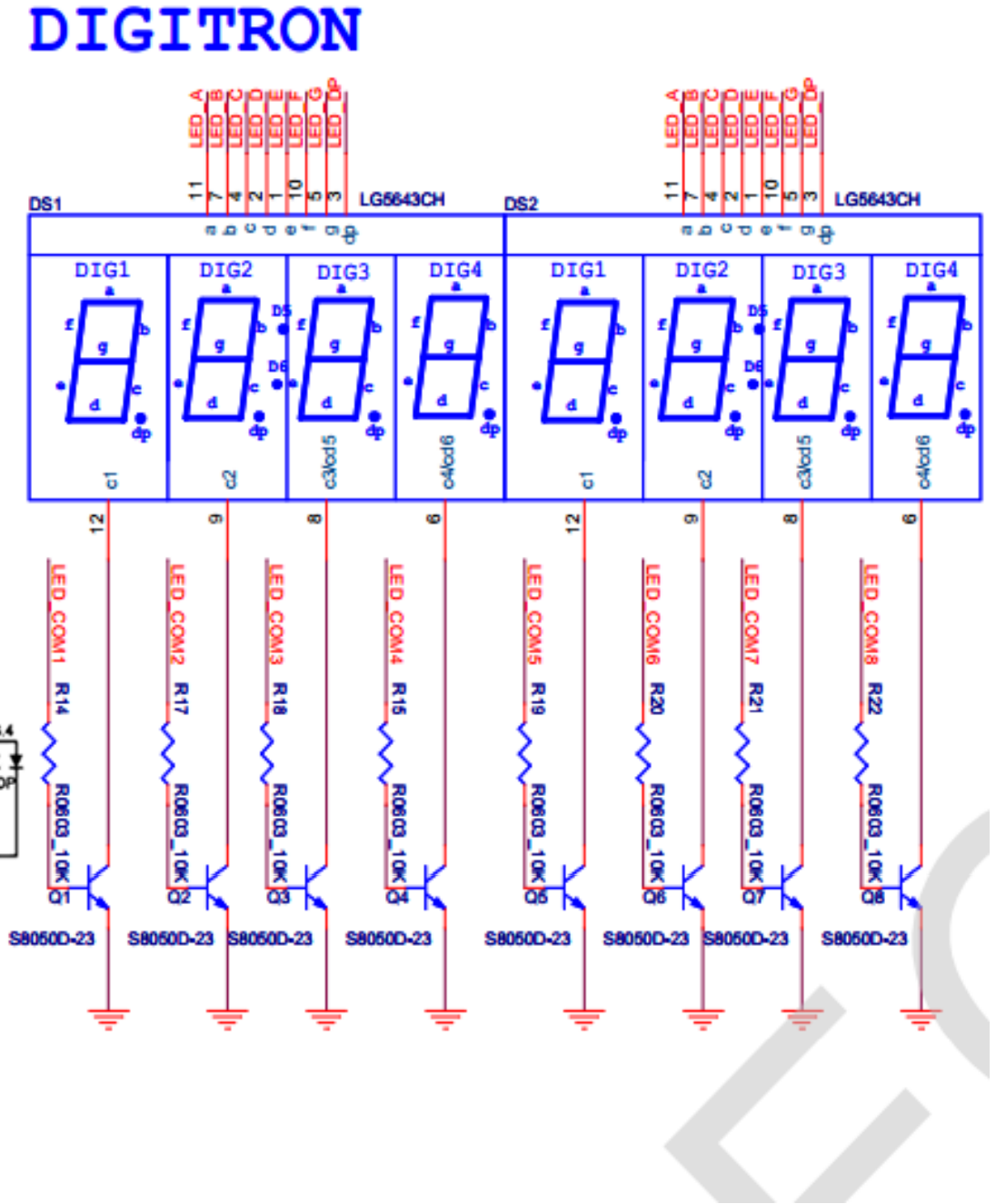
- TCA6424的使用分为两步：
  - TCA6424初始化，即将端口P0配置为输入，P1，P2配置为输出。
  - 对TCA6424的输出端口赋值，从而点亮动态数码管。

蓝板上8位数码管



PCB:D3641A/B





I2C编程

GPIO管脚复用

- I2C0 SCL-GPIO PB2
  I2C0 SDA-GPIO PB3

- *GPIOPinConfigure(GPIO_PB2_I2C0SCL);//配置PB2为I2C0SCL*
  GPIOPinConfigure(GPIO_PB3_I2C0SDA);//配置PB3为I2C0SDA
  GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
  //I2C将GPIO_PIN_2用作SCL
  GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
  //I2C将GPIO_PIN_3用作SDA

函数GPIOPinConfigure

⌘ 驱动库手册：P266

功能	Configures the alternate function of a GPIO pin.
原型	Void **GPIOPinConfigure**(uint32_t **ui32PinConfig**)
参数	**ui32PinConfig** is the pin configuration value, specified as only one of the GPIO_P??_??? values.
说明	This function configures the pin mux that selects the peripheral function associated with a particular GPIO pin. Only one peripheral function at a time can be associated with a GPIO pin, and each peripheral function should only be associated with a single GPIO pin at a time (despite the fact that many of them can be associated with more than one GPIO pin). To fully configure a pin, a GPIOPinType() function should also be called.

函数GPIOPinTypeI2CSCL

- 驱动库手册：P275
- 引脚复用：所有的GPIO引脚都可以用作GPIO或是一种或多种的外设功能。

功能	配置管脚用作I2C的SCL
原型	Void **GPIOPinTypeI2CSCL**(uint32_t **ui32Port**, uint8_t **ui8Pins**)
参数	**ui32Port**是GPIO端口的基址。 **ui8Pins**是管脚的位组合（bit-packed）表示。
说明	The I2C pins must be properly configured for the I2C peripheral to function correctly. This function provides the proper configuration for the SCL pin. The pin is specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

I2C初始化

- *I2CMasterInitExpClk(I2C0_BASE,ui32SysClock, true);*
  //config I2C0 400k

- *I2CMasterEnable(I2C0_BASE);*

函数I2CMasterInitExpClk

⌘ 驱动库手册：P331

功能	初始化I2C的主机模块。
原型	Void **I2CMasterInitExpClk**(uint32_t **ui32Base**, uint32_t **ui32I2CClk**, bool **bFast**)
参数	**ui32Base**是I2C模块的基地址。 **ui32I2CClk**是为I2C模块提供的时钟频率。 **bFast**为建立快速数据传输。
说明	该函数通过为主机配置总线速度并使能I2C 主机模块，来初始化I2C主机模块的操作。 如果参数**bFast**为true, 则主机模块的传输速率为400 Kbps; 否则, 主机模块的传输速率为100 Kbps。 外设时钟与系统时钟相同。

函数I2CMasterEnable

⌘ 驱动库手册：P329

功能	使能I2C主机主机模块。
原型	Void **I2CMasterEnable**(uint32_t **ui32Base**)
参数	**ui32Base**为I2C模块的基地址。
说明	该函数使能I2C主机模块的操作。

I2C写操作

```
uint8_t I2C0_WriteByte(uint8_t DevAddr, uint8_t RegAddr, uint8_t WriteData)
{
uint8_t rop;
while(I2CMasterBusy(I2C0_BASE)){};//如果I2C0模块忙，等待
I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false);
//设置主机要放到总线上的从机地址。false表示主机写从机，true表示主机读从机

I2CMasterDataPut(I2C0_BASE, RegAddr);//主机写设备寄存器地址
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);/
/执行重复写入操作
while(I2CMasterBusy(I2C0_BASE)){};

rop = (uint8_t)I2CMasterErr(I2C0_BASE);//调试用
```

函数I2CMasterBusy

⌘ 驱动库手册：P327

功能	表示I2C主机是否忙。
原型	Bool **I2CMasterBusy**(uint32_t **ui32Base**)
参数	**ui32Base**为I2C模块的基地址。
说明	该函数返回主机是否忙于传输或接收数据的标识。
返回值	如果主机忙，则返回true；否则返回false。

函数I2CMasterSlaveAddrSet

⌘ 驱动库手册：P336

功能	设置I2C放到总线上的地址。
原型	Void **I2CMasterSlaveAddrSet**(uint32_t **ui32Base**, uint8_t **ui8SlaveAddr**, bool **bReceive**)
参数	**ui32Base**为I2C模块的基地址。 **ui8SlaveAddr**为7位从机地址。 **bReceive**表示与从机通信的类型。
说明	在开始通信时，该函数配置I2C放到总线上的地址。当bReceive参数被设置为true时，表示主机启动对从机的读操作；否则，表示主机启动对从机的写操作。

函数I2CMasterDataPut

⌘ 驱动库手册：P329

功能	从I2C主机传输一个字节。
原型	Void **I2CMasterDataPut**(uint32_t **ui32Base**, uint8_t **ui8Data**)
参数	**ui32Base**为I2C模块基地址。 **ui8Data**为主机传输的数据。
说明	该函数将提供的数据放入I2C主机数据寄存器。

函数I2CMasterDataGet

⌘ 驱动库手册：P328

功能	接收已经发给I2C主机的一个字节。
原型	uint32_t **I2CMasterDataGet**(uint32_t **ui32Base**)
参数	**ui32Base**为I2C模块的基地址。
说明	该函数从I2C主机寄存器读取数据的一个字节。
返回值	返回I2C主机接收到的字节，并强制转换为uint32_t类型。

函数I2CMasterControl

⌘ 驱动库手册：P327

功能	控制I2C主机模块的状态。
原型	Void **I2CMasterControl**(uint32_t **ui32Base**, uint32_t **ui32Cmd**)
参数	**ui32Base**为I2C模块的基地址。 **ui32Cmd**为向I2C主机下达的指令。
说明	该函数用于控制主机发送和接收操作的状态。参数ui8Cmd为以下值之一：**I2C_MASTER_CMD_SINGLE_SEND**、**I2C_MASTER_CMD_SINGLE_RECEIVE**、**I2C_MASTER_CMD_BURST_SEND_START**、**I2C_MASTER_CMD_BURST_SEND_CONT**、**I2C_MASTER_CMD_BURST_SEND_FINISH** ……

函数I2CMasterErr

⌘ 驱动库手册：P330

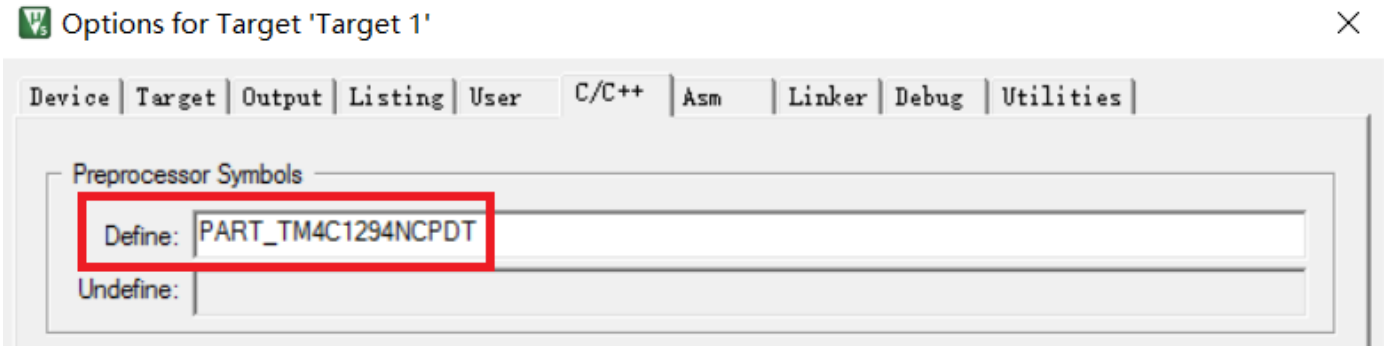
功能	获取I2C主机的错误状态。
原型	uint32_t **I2CMasterErr**(uint32_t **ui32Base**)
参数	**ui32Base**为I2C模块的基地址。
说明	该函数用于获得主机发送和接收操作的错误状态。
返回值	返回的错误状态为以下之一：**I2C_MASTER_ERR_NONE**、**I2C_MASTER_ERR_ADDR_ACK**、 **I2C_MASTER_ERR_DATA_ACK**,或**I2C_MASTER_ERR_ARB_LOST**。

第二次实验

Option for Target “Target 1”

- 新增：
  - C/C++

    Preprocessor Symbols-Define： PART_TM4C1294NCPDT
  - 原因：CPU型号预定义，因为driverlib中某些头文件需要根据CPU类型进行不同预定义
  - 第二次实验必需的



实验目的

- 了解I2C总线标准及在TM4C1294芯片的调用方法
- 掌握用I2C总线扩展GPIO芯片PCA9557及TCA6424的方法
- 能够通过扩展GPIO来输出点亮LED及动态数码管

实验2-1

- PCA9557点亮所有LED；TCA6424控制数码管显示0在第一位。
- 要求：阅读、理解程序。
- 思考题：
  - 人为修改内部时钟或外部时钟，如将内部时钟改为8M，或将外部时钟改为30M，会有什么结果？
  - 能否将PLL时钟调整到外部时钟的频率以下？如将25M外部时钟用PLL后调整为20M？
  - 将PLL后的时钟调整为最大值120M，LED闪烁会有什么变化？为什么？

实验2-2

- 进行LED的跑马灯实验，当LED在某位点亮时，同时在数码管的某位显示对应的LED管号。如LED跑马灯时，从左到右依次点亮LED1~LED8，此时在数码管上依次显示1~8。
- 要求：阅读、理解程序。

实验2-3

- 接程序2-2，当按键USR_SW1按下时，停止跑马灯，但LED及数码管显示维持不变；当按键松开后，继续跑马灯。