1、接口使用：

82xx与83xx（及其以上）使用统一的读写接口，接口函数在i2cDrv.c里面,当然也可以调用各自平台的读写接口。

/\*\*

\* 读取i2c数据

\* @param busIndex 总线号

\* @param devAddr 设备地址

\* @param regAddrArray 读取过程是先写出寄存器地址，寄存器地址位数不定，因此传入的是一个字节流

\* @param regAddrLen 寄存器地址的长度，也就是写入的regAddrArray的长度，单位:byte

\* @param pData 读取数据buffer的指针

\* @param len 读取的长度，单位Byte

\* @param waitRtn 某些i2c设备在写完寄存器地址后需要等待固定的时间或是查询某个状态，waitRtn可以实现这种操作，如果不需要该函数，传入NULL

\*

\* @return OK:成功 <0:错误

\*/

int i2cRead(int busIndex, UINT8 devAddr, UINT8 \* regAddrArray, int regAddrLen, char \* pData, int len, FUNCPTR waitRtn)

{

#if CPU==PPC603

#ifdef MPC83XX

return fslAdvI2cRead(busIndex, devAddr, regAddrArray, regAddrLen, pData, len, waitRtn);

#else

return mpc82xxI2cRead( devAddr, regAddrArray, regAddrLen, pData, len);

#endif

#else

return fslAdvI2cRead(busIndex, devAddr, regAddrArray, regAddrLen, pData, len, waitRtn);

#endif

}

/\*\*

\* 写入i2c数据

\*

\* @param busIndex 总线号

\* @param devAddr 设备地址

\* @param regAddrArray 读取过程是先写出寄存器地址，寄存器地址位数不定，因此传入的是一个字节流

\* @param regAddrLen 寄存器地址的长度，也就是写入的regAddrArray的长度，单位:byte

\* @param pData 写入数据的指针

\* @param len 写入的数据长度，单位Byte

\* @param waitRtn 某些i2c设备在写完寄存器地址后需要等待固定的时间或是查询某个状态，waitRtn可以实现这种操作，如果不需要该函数，传入NULL

\*

\* @return OK:成功 <0:错误

\*/

int i2cWrite(int busIndex, UINT8 devAddr, UINT8 \* regAddrArray, int regAddrLen, char \* pData, int len, FUNCPTR waitRtn)

{

#if CPU==PPC603

#ifdef MPC83XX

return fslAdvI2cWrite(busIndex, devAddr, regAddrArray, regAddrLen, pData, len, waitRtn);

#else

return mpc82xxI2cWrite( devAddr, regAddrArray, regAddrLen, pData, len);

#endif

#else

return fslAdvI2cWrite(busIndex, devAddr, regAddrArray, regAddrLen, pData, len, waitRtn);

#endif

}

2、配置方式

Deviceconfig里面添加I2C配置信息 在你的工程里面include I2C组件

82xx

#ifdef INCLUDE\_ZMD\_I2C

struct hcfResource i2cx0Resources[] =

{

{ "irq", HCF\_RES\_INT, {(void \*)INUM\_I2C}},

{ "predivider", HCF\_RES\_INT, {(void \*)((UINT32)16)}},/\*I2C 控制器的预分频系数\*/

{ "workMode", HCF\_RES\_INT, {(void \*)((UINT32)1)}},/\*1为master模式 0 为slave模式\*/

#if 0 /\*可选项 不配置使用以下参数作为默认配置\*/

{ "i2cSpeed", HCF\_RES\_INT, {(void \*)((UINT32)100000)}},

{ "maxBufLen", HCF\_RES\_INT, {(void \*)((UINT32)16\*1)}},/\*取与BDxSpace相同大小即可\*/

{ "BDxSpace", HCF\_RES\_INT, {(void \*)((UINT32)16\*1)}},/\*16 \* n\*/

{ "BDxTxNum", HCF\_RES\_INT, {(void \*)((UINT32)2\*1)}},/\*1 至少两个2TXBD 接收数据需要两个txBD来完成 发送实际上只能使用一个BD\*/

{ "BDxRxNum", HCF\_RES\_INT, {(void \*)((UINT32)1\*1)}},/\*2 1Rx\*/

{ "BDxbufWidth", HCF\_RES\_INT, {(void \*)((UINT32)64\*4)}},/\*64 \* n\*/

#endif

};

#define i2cx0Num NELEMENTS(i2cx0Resources)

#endif

/\* 设备定义列表 \*/

struct hcfDevice hcfDeviceList[] =

{

。。。

#ifdef INCLUDE\_ZMD\_I2C

{ "i2c", 0, VXB\_DEVID\_I2C, 0, i2cx0Num, i2cx0Resources},

#endif

};

83xx及其以上平台

#ifdef INCLUDE\_ZMD\_I2C

struct hcfResource i2cx0Resources[] =

{

{ "cs", HCF\_RES\_INT, {(void \*)0} },/\*i2c控制器map\*/

{ "irq", HCF\_RES\_INT, {(void \*)INUM\_I2C1} },/\*中断号\*/

};

#define i2cx0Num NELEMENTS(i2cx0Resources)

struct hcfResource i2cx1Resources[] =

{

{ "cs", HCF\_RES\_INT, {(void \*)1} },/\*i2c控制器map 0 83xX只有一个\*/

{ "irq", HCF\_RES\_INT, {(void \*)INUM\_I2C2} },/\*中断号\*/

};

#define i2cx1Num NELEMENTS(i2cx1Resources)

#endif

struct hcfDevice hcfDeviceList[] =

{

。。。。。。

#ifdef INCLUDE\_ZMD\_I2C/\*注意: 83xx 的8315只有一个I2C控制器\*/

{ "i2c", 0, VXB\_DEVID\_I2C, 0,i2cx0Num,i2cx0Resources},

{ "i2c", 1, VXB\_DEVID\_I2C, 1,i2cx1Num,i2cx1Resources},

#endif

};

3、调用方式，在你需要使用读写接口的地方 extern 读写函数即可。