mergePPC

# 工作目的

完成PPC目标机端代码的合并，使ppc版本：8569,8280,8640d，p2020，使这四个版本的ppc合并成一个目标机端源码，我把这个工作取名mergePPC。

# 版本依据

根据笑哥的给予svn路径，从以下取得：

**8280:**

http://192.168.16.4/svn/PPC/branches/task/022ppc8280针对(0.0.0.9)自测试修改后打包/output/code(没有工具库源码)

**8569:**

http://192.168.16.4/svn/722道系统PPC采购/branches/task/09ppc8569阶段交付打包/output/code

**P2020:**

http://192.168.16.4/svn/PPC/branches/task/023ppcP2020试用包打包\_723所

**8640D:**

http://192.168.16.4/svn/PPC/branches/task/017ppc8640d自测试前打包（0.0.0.8）/output/code

另外：

ppc8280到现在还没有找到目标机端源码，但获取以下信息：

1. 小倩告诉我目标机端源码就是和8569的一样，但是ts有改过,小倩给的svn路径http://192.168.16.4/svn/PPC/branches/task/016ppc8569自测试前打包 (0.0.0.7)/output/src/tool
2. 江哥告诉我8280没有目标机端代码，当时只是做了主机端硬件调试
3. 兵哥告诉我他当时只是接着小倩做了BDI的调试
4. 和江哥，笑哥沟通，确定8280就没有目标机端的代码，只需合并**8569,8640d，p2020**

# 对比依据

1. 对比.h文件和.c文件(include/src)
2. 对比主makefile
3. 对比ppcGnu4.12
4. 先对比简单的5个库，最后对比sysDebug库
5. 需要注意确定宏，然后这个宏在哪定义一定要确定好
6. 以8569为模板，把2020和8640d里面的差异加进去
7. 8569和2020用的是同一个型号的CPU，只不过8569是单核，2020是多核
8. 2020只需进行合并工作，但不需要验证，因为没有板子。
9. 8569，2020是软件单步，8640d是硬件单步。

# 对比工作

## 5库源码差异

3-27完成了hostShell，mdl，rse，tautils，wdb库的h和c文件的比较。

### hostshell

shellAcceptor(port, 1024\*40, 8000, 0); //8569

//shellAcceptor(port, 1460, 8000, 0); //p2020,8640d

### mdl

mdl库的LoadModule接口中：

moduleInfo.segment[1].address = dataAdd;

moduleInfo.segment[1].size = dataSize;

moduleInfo.segment[1].type = SEGMENT\_DATA;

moduleInfo.segment[1].flags = 0;

**修改原因：**

dataSize，不用说了吧？我认为就是手误吧。。。

flags参数需要改为0，如果参数为SEG\_FREE\_MEMORY，那么系统在释放内存时会再释放一次，然而系统在分配内存时是统一给seg、data、bss段分配内存的，所以只需要释放一次就行了。

### rse

rse的源码并没有改动，一毛一样

### tautils

#include "string.h"

#include <stdarg.h>

#include <stdio.h>

//ppc8569和ppc2020

//ppc8640d没有引用stdarg/stdio,只有#include <string.h>

### wdb

wdb的源码并没有改动，一毛一样

## 5库makefile

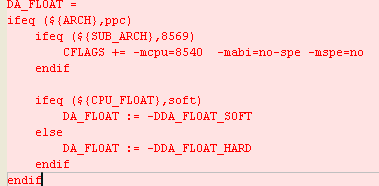
接下来准备对比主makefile

既然是makefile，那么我就需要理清楚该库的makefile流程

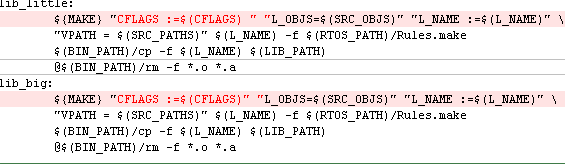
另外，了解一下makefile里面有什么？

### hostshell

统一修改为：如果是8569，则需要对CFLAGS新增参数，2020和8640d则没有执行该分支。



我很奇怪为何之前的8569会没有标红的CFLAGS := $(CFLAGS)呢？难道CFLAGS这个变量不需要传递给子make吗？？？2020和8640d本来都有的，现在统一都加上了。



经过测试，即使不加CFLAGS，它的参数也会传递到子make中，这是为什么呢？但不管怎样，加上总是没有错的

### mdl

和hostshell情况一样。但8569和2020一样，对CFLAGS 新增了参数，以及没有传递CFLAGS给子make。8640d单一。

### rse

和hostshell情况一样。但8569和2020一样，对CFLAGS 新增了参数，以及没有传递CFLAGS给子make。8640d单一。

### tautils

和hostshell情况一样。但8569和2020一样，对CFLAGS 新增了参数，以及没有传递CFLAGS给子make。8640d单一。

### wdb

和hostshell情况一样。但8569和2020一样，对CFLAGS 新增了参数，以及没有传递CFLAGS给子make。8640d单一。

### Rules.make

3个版本是一样的

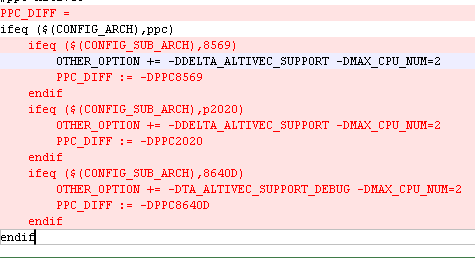
### base.make

修改了两个地方

1. 统一把ppc体系结构的配置，ifeq (${SUB\_ARCH},8569)和ifeq (${SUB\_ARCH},P2020)放到了一个base.make文件里面
2. VXWORK\_CFLAGS用宏区分开来，ppc8569与2020以及8640D不一样，我也很奇怪。

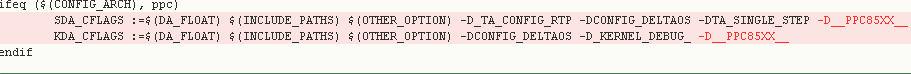
## sysDebug的makefile

修改了：



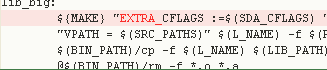
其实我也不晓得为啥子这么改，因为这几个参数我没去了解原因。

但是PPC\_DIFF是我用来在sysDebug源码中区分不同ppc版本的宏。



其中-D\_\_PPC85XX\_\_在base.make里面有定义啊，我不晓得ppc2020为啥还要重新用该宏。所以统一删掉了。

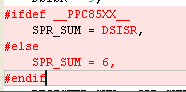




以上，添上CFLAGS，并把EXTRA\_CFLAGS改为CFLAGS。

## sysDebug源码

### dbContext.h



如上图，include的头文件就这么点更改。

### 8569和8640d对比区别

**taskExceptionLib.c**

**sdaExceptionLib.c**

**dbStandardRsp.c**

**dbAtom.c**

只有这四个文件有区别，并且都用宏\_\_PPC85XX\_\_来区分开来，她们都是软件单步和硬件单步的区别，所以统一用8569并且不用改。

### 8569和2020对比区别

发现改动不是很明白，想暂时先用8569的sysDebug，然后在看。后面因为没有p2020的板子就暂时没测。。。

## ta

对于ta，我看它们的差异全部集中在board下面。

所以我的做法是把它们都用各自的文件夹区分开来，就可以了。

所以把ppc2020的board，从PowerPC\_8569\_722s改为了PowerPC\_2020

所以目标板类型就需要在boardconfiguration.xml文件中新增。

。。。。。

以上作废，因为笑哥说2020没有ta，有ta的只有8569和8640d，对应的板子是PowerPC\_8569\_722s和PowerPC\_CPPCAB\_0C1，其他板子都删掉，并在boardconfiguration.xml中删除。

## gnu4.1.2

对于这个，需要在模板里面改动，怎么改？

platform\target\deltaos\prj，这个位置可以改动toolchain.mk，其他地方还不行。。。

## debug头文件

差异几乎没有，唯一差异在于：

EXC\_FLOATING\_POINT\_UNAVAILABLE

EXC\_FLOATING-POINT\_UNAVAILABLE

该宏定义的浮点不可用异常的偏移地址，当然下横线是对的。

# 合包工作

1. 合toolchain.mk，每个toolchain.mk模板对应一个ppc版本，该路径在platform\target\deltaos\prj\customize。确定了8569等都是使用spe协处理器来进行浮点的计算，所以软硬浮点都没关系啦啦啦，但是这里统一写了硬浮点。
2. 合用户自定义项目可以创建各ppc版本，修改位置如下template\deltaos\framework\customize。
3. 合base.make以及Rules.make
4. 合ta
5. 合库，并在IDE上面用用户自定义项目编出适合各版本的工具库。

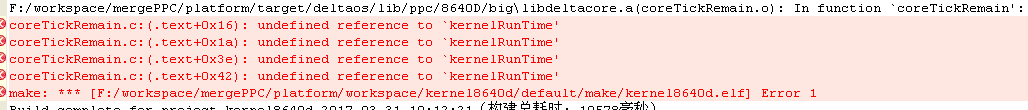
于此，合包完毕，接下来便是验证工作。

# 验证

## 问题1

VXWORK\_CFLAGS这个宏统一以8569的为准，不然2020和8640d编译不过。。

## 问题2



询问笑哥，得出这个是工具应该生成的一个全局变量，用于限制时间的，需要问下小倩要不要改IDE，并且在prjconfig.c里你自己定义一个就能编过。

于是，为了接着往下测试，临时在prjConfig.c文件中添加了全局变量char kernelRunTime[];

后面小倩改了插件。

## 问题3

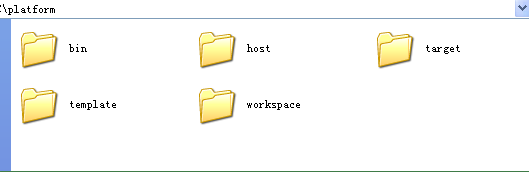
usrtools.c和taTaskIfLib.c文件忘记改了。因为这个原因折腾了一天，以后谨记。

和工具库相关的文件：库文件，base.make，Rules.make，然后usrtools.c和taTaskIfLib.c文件，以及include下的debug头文件。

这次的问题是：下面这行代码被屏蔽了，而在多核的情况下，这个ipi中断初始化一定是要打开的，另外，这也提醒我做事一定要仔细，遇到问题不要慌，要静下心分析。

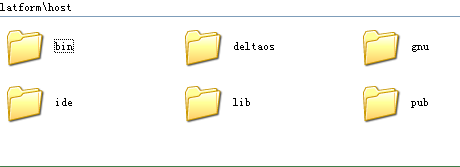
sdaIpiInit(intConnect\_debug,\_ipi\_emit\_debug,intEnable\_debug);

另外，OS编译出来，是需要哪些文件呢在？其实这得从platform的文件夹结构说起。



bin文件夹下面只有一个sh.exe

host下面：



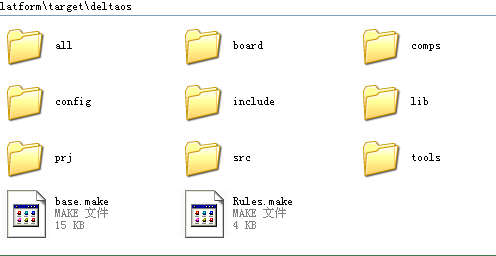
其中bin是放make等工具的。。

deltaos下面放的是loader文件，用来辅助loader.exe下载用的。

gnu下面放的是gdb和gnu的一系列工具

ide下面放的全是主机端相关的东西，其中LambdaPRO放的全是插件。

lib下面看不懂，也不用管，pub下面放着ts日志，ftp，虚拟机，loader.exe，ts.exe等等。



target下面就是deltaos和target\lib\_smp\h\config。

其中config下面就2个头文件，一个auto.conf，看不懂。。。

deltaos下面如上图。

all不管；

board里面放的就是bsp

comps放了很多.c这里需要着重注意，因为工具相关的usrtools.c和taTaskIfLib.c都在这里面。

include下面放的是头文件，工具相关的debug头文件夹也在这里面。

lib里面就是库啊库啊。

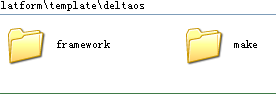
prj了里面放的就是toolchain.mk模板。

src下面不管他。。。是操作系统源码，但是给出去的包应该只有头文件。所以需要自己从svn上面下源码下来替换掉它们。

tools下面放的就是monitor，也就是我们的ta源码。

base.make和Rules.make是编译的时候都需要include的基本makefile。

template：



项目的模板，其中包括项目的框架模板，就是这个项目一开始的样子。

然后使项目的makefile模板，就是这个项目默认执行的makefile模板啦。