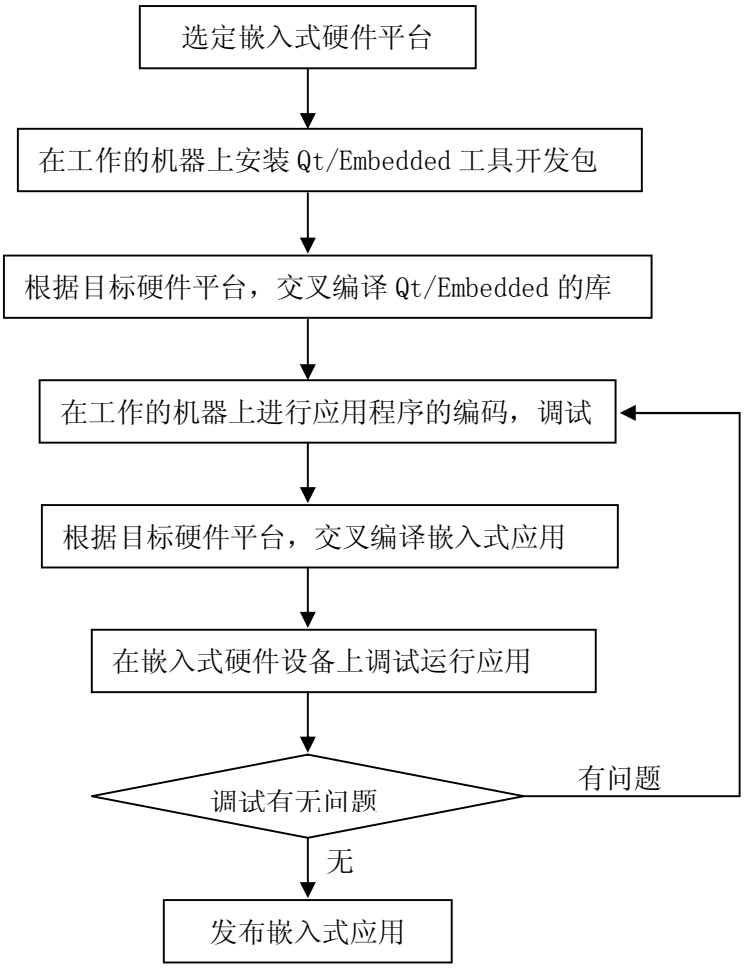
Qt 嵌入式图形开发（实战篇）

前面我们详细介绍了 Qt 嵌入式工具开发包的安装和使用方法，但是这个介绍对于要真正进行一次商业的嵌入式应用开发来说并不足够。嵌入式应用的开发工作基本上是在工作站或是 PC 机上完成的，我们在工作的机器上调试运行嵌入式应用，并将输出结果显示在一个仿真小型设备显示终端的模拟器上。在开发的后期，我们要根据我们选择的嵌入式硬件平台，将嵌入式应用编译链接成适合在这个硬件平台上运行的二进制目标代码，另外由于应用使用到了 Qt/Embedded 的库，所以我们还要将 Qt/Embedded 库的源代码编译链接成为适合在这个硬件平台上使用的二进制目标代码库。当一个 Qt/Embedded 应用被部署到小型设备上，并可靠的运行，这样一个开发过程才宣告结束。

使用 Qt/Embedded 开发一个嵌入式应用的过程大体可用下面的流程图表示：



图一　Qt/Embedded 应用开发的一般流程

在以下的篇幅，我们将按照上述的流程完整的介绍使用 Qt/Embedded 开发一个嵌入式应用例子的过程。

一、嵌入式硬件开发平台的选择

嵌入式系统的核心部件是各种类型的嵌入式处理器，目前据不完全统计，全世界嵌入式处理器的品种总量已经超过 1000 多种，流行体系结构有 30 几个系列，如何在种类纷繁的嵌入式处理器中选择适合应用需求的处理器呢？在应用的需求分析过程中，有几项因素决定了应该选择什么样的嵌入式处理器：①嵌入式处理器能否在技术上实现应用②嵌入式处理器的成本是否符合应用要求。对于嵌入式处理器能否在技术上实现应用需要考虑到应用在运行时的特点，比如应用对实时性的要求，对计算量和计算速度的要求，对外围接口电路的要求，对图形用户接口的要求等。当我们选定了一个嵌入式处理器之后，我们还要考虑选用什么样的操作系统，在选择操作系统时我们也要注意嵌入式处理器能否被所选的操作系统支持，有没有合适的编译器能够生成支持这种操作系统和嵌入式处理器的二进制目标代码。

近年来，随着 EDI 的推广和 VLSI 设计的普及化，及半导体工艺的迅速发展，在一个硅片上实现一个更为复杂的系统的时代已来临，这就是 System On Chip(SOC)。各种通用处理器内核将作为 SOC 设计公司的标准库，和许多其它嵌入式系统外设一样，成为 VLSI 设计中一种标准的器件，用标准的 VHDL 等语言描述，存储在器件库中。用户只需定义出其整个应用系统，仿真通过后就可以将设计图交给半导体工厂制作样品。这样除个别无法集成的器件以外，整个嵌入式系统大部分均可集成到一块或几块芯片中去，应用系统电路板将变得很简洁，对于减小体积和功耗、提高可靠性非常有利。

SOC 可以分为通用和专用两类。通用系列包括 Infineon 的 TriCore，Motorola 的 M-Core，某些 ARM 系列器件，Echelon 和 Motorola 联合研制的 Neuron 芯片等。专用 SOC 一般专用于某个或某类系统中，不为一般用户所知。一个有代表性的产品是 Philips 的 Smart XA，它将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的 CCU 单元制作在一块硅片上，形成一个可加载 JAVA 或 C 语言的专用的 SOC，可用于公众互联网如 Internet 安全方面。

由于 SOC 芯片集成了处理器和许多常用的外围接口芯片，使得它的功能变得很强大，能够应用的领域和场合变得很广，所以嵌入式开发板的厂商选择 SOC 芯片作为开发板的核心芯片。三星的 S3C2410 就是这样一款带有 ARM920T 处理器内核的 SOC 芯片，由于它主频高，内置 LCD 和触摸屏控制器，以及声音控制器等外围电路，因而用在对图形用户接口有较高要求的场合是非常合适的。因为支持 ARM9 处理器的 linux 编译器早已发布，所以 S3C2410 可以很好的支持 linux 和 Qt/Embedded 的运行。

深圳优龙科技有限公司设计开发的 FS2410 嵌入式开发板是一个成熟稳定的嵌入式开发硬件平台，它使用了三星的 S3C2410 作为核心芯片。以下是 FS2410 的一些资料（更详细的资料可参考优龙公司的《FS2410 开发板使用手册》）。在本文提及的 Qt/Embedded 应用例子都是居于 FS2410 嵌入式硬件开发平台的。

1．1 优龙 FS2410 嵌入式开发板介绍

1.1.1 FS2410 开发板资源

技术参数：

S3C2410：16-/32-bit ARM920T 内核

系统时钟：使用外部12MHz 晶体由CPU 内部PLL 备频至200MHz
BOOT ROM：SST39VF160
NAND FLASH：K9F5608/ K9F1208
SDRAM：64Mbyte（32Mbyte×2）
TFT/STN LCD 和触摸屏控制器
2 通道UART
2 个USB 主机控制器（其中一个可配置为USB 设备控制器）
SD 卡/MMC 卡主机控制器
Embedded-ICE 调试接口
RTC 实时时钟（具备后备锂电池）
IIC 总线接口（驱动AT24C04-SC27）
ADC 模数转换接口
SPI 接口
IIS 数字音频输入/输出接口，音频输出采用UDA1341 Dac
EINT 外部中断接口
IrDA 红外线收发器
16 板上轻触键
CS8900，10M 以太网接口
发光二极管指示灯

1.1.2 FS2410 开发套件构成

FS2410 套件包括:
1）一块已测试好的FS2410 开发板（包括FS2410 核心板和设备板）
2）LCD 板一块，包含三星3.5 寸256K 色TFT 真彩屏加驱动电路
3）一个SuperJtag 调试头,该调试头可用来烧写2410 的boot 程序
4）一条两母直连串口线
5）一条交叉线网线
6）一条USB 线
7）一条并口线
8）一个+7.5~12V 直流电源
9）两张FS2410 光盘
10）一个包装盒
光盘内所附软件：
1.DEMO 程序-开发板测试程序源代码
1）NANDBOOT
NAND FLASH 启动引导程序， 通过bios 或jtag 烧入nand flash 0 地址,d9~d12 闪烁
2） u2410mon
nor flash 启动，通过usb 下载,通过bios 或jtag 烧入nor flash 0 地址
3）S3C2410_TEST
三星测试程序，包括各种测试程序（IIC，sd），仿真器usb 下运行载，
4）2410app
自编的测试程序，SD，按键，

2 flash 烧写程序
3 FS2410 LINUX for S3C2410 源代码及开发工具包，含CS8900 网口驱动，串口驱动，USB 驱动
4 FS2410 三星官方测试代码。
开发系统文档：
1) FS2410 使用手册
2) FS2410 原理图
3) 开发板元器件手册

二、安装 Qt/Embedded 工具开发包

我们要进行 Qt/Embedded 开发，就需要在我们工作的机器上安装 Qt/Embedded 工具开发包，这一步已在《Qt 嵌入式开发（入门篇）》的第一节中谈及，在此不在赘述。

三、交叉编译 Qt/Embedded 的库

开发居于 Qt/Embedded 的应用程序要使用到 Qt/Embedded 的库，我们编写的 Qt 嵌入式应用程序最终是在 FS2410 开发板上运行的，因此在把 Qt 嵌入式应用程序编译成支持 FS2410 的目标代码之前，我们需要两样东西，一个是 arm9 的 linux 编译器，另一个是经 arm9 的 linux 编译器编译过的 Qt/Embedded 的库。

①安装交叉编译工具

我们需要arm9的linux编译器去编译工程并产生arm9处理器的目标代码，而我们却是在一台PC机或者工作站上（这个机器有可能是x86的处理器）使用这个编译器，这个过程被称为交叉编译。交叉编译的实际定义是在一个处理器平台上编译产生一个工程代码的另一个处理器的目标代码。关于如何安装一个arm9的linux编译器请参考《FS2410开发板使用手册》第七章“如何编译linux”，但是需要特别提醒读者注意，使用toolchain做为交叉编译工具时，最好使用cross-3.3.2及其以后的版本，这样才能对qt/embeded有良好支持。

②交叉编译Qt/Embedded库

当我们有了arm9的linux编译器之后，我们就可以使用这个编译器交叉编译Qt/Embedded库的源代码，从而产生一个以ARM9为目标代码的Qt/Embedded库。具体过程如下

1、 解包 Qt/Embedded（以 Qt/Embedded2.3.7 为例）

解包这个 Qt/Embedded2.3.7 压缩包时，应把它解压缩到不同于你的机器的处理器使用的 Qt/Embedded2.3.7 的安装路径。

在 Linux 命令模式下运行以下命令：

```
tar xfz qt-embedded-2.3.7.tar.gz
```

2、配置 Qt/Embedded2.3.7 的安装

Qt/Embedded 的安装选项有很多个，你可以在命令行下直接输入“./configure”来运行配置，这时安装程序会一步一步提示你输入安装选项。你也可以在“./configure”后输入多个安装选项直接完成安装的配置。在这些选项中有一个选项决定了编译 Qt/Embedded 库的范围，即可以指定以最小，小，中，大，完全 5 种方式编译 Qt/Embedded 库。另外 Qt/Embedded 的安装选项还允许我们自己定制一个配置文件，来有选择的编译 Qt/Embedded 库，这个安装选项是“-qconfig local”；当我们指定这个选项时，Qt/Embedded 库在安装过程中会寻找 qt-2.3.7/src/tools/qconfig-local.h 这个文件，如找到这个文件，就会以该文件里面定义的宏，来编译链接 Qt/Embedded 库。

因为我们使用了优龙公司的 FS2410 嵌入式开发板，所以为了支持触摸屏的显示，我们需要定义一些宏，为了避免初学者在一开始就直接接触到 Qt/Embedde 的复杂的宏编译选项，我们把这些宏定义到一个名为 qconfig-local.h 的安装配置文件中，当读者在安装 Qt/Embedded 的时候，需要把这个文件复制到 Qt/Embedded 的安装路径的/src/tools 子路径下。例如您安装的是 Qt/Embedded2.3.7，那么您应该把我们提供给您的 qconfig-local.h 安装配置文件复制到 qt-2.3.7/src/tools 路径下。

具体过程如下：

```
cd qt-2.3.7
export QTDIR=$PWD
export QTEDIR=$QTDIR
cp /配置文件所在路径/qconfig-local.h   ./src/tools
make clean
./configure -xplatform linux-arm-g++ -shared -debug （接下行）
-qconfig local -qvfb -depths 4,8,16,32
make
cd ..
```

四、一个 Hello,World 的例子

下面我们将通过编写一个跳动的“Hello,World”字符串，来讲解 Qt 嵌入式应用的开发过程。

① 生成一个工程文件（.pro 文件）

一个应用通常对应一个工程文件，生成一个工程文件，并对它做一些简单的编辑，然后使用一个专门的工具（例如 tmake）处理这个工程文件，就可以生成一个 Makefile 文件。

产生一个工程文件的其中一个方法是使用 progen 命令(progen 程序可在 tmake 的安装路径下找到)。下面我们使用 progen 产生一个名为 hello 的工程文件

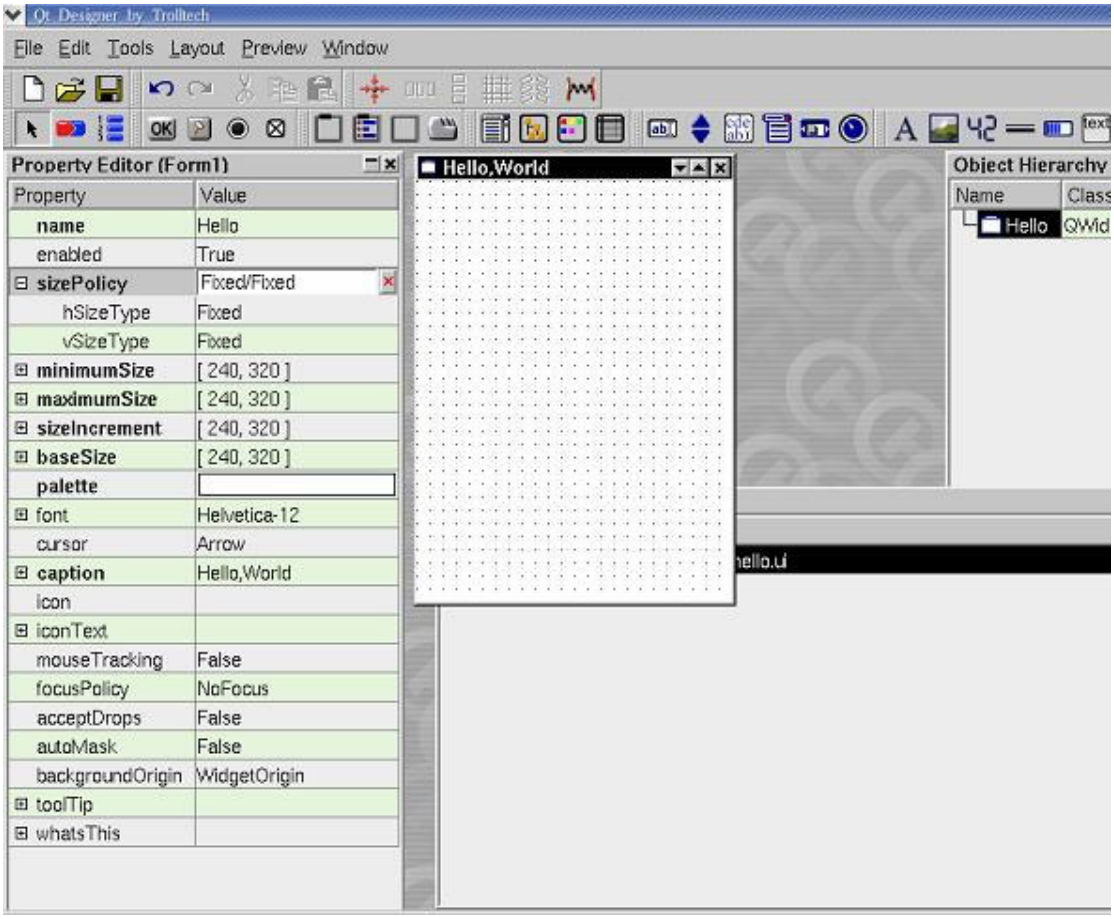
```
progen -t app.t -o hello.pro
```

产生的 hello.pro 工程文件并不完整，开发者还需手动往里添加工程所包含的头文件，源文

件等信息。

② 新建一个窗体

在 qt-2.3.x for x11 的安装路径的 bin 目录下运行“./designer”命令，就启动了一个 Qt 图形编辑器。点击编辑器的“new”菜单，弹出了一个“new Form”对话框，在这个对话框里我们选择“Widget”，然后点击“OK”按钮，这样我们就新建了一个窗体。接着我们对这个窗体的属性进行设置，注意把窗体的“name”属性设为“Hello”；窗体的各种尺寸设为宽“240”，高“320”，目的是使窗体大小和 FS2410 带的显示屏大小一致；窗体背景颜色设置为白色。具体设置可参见下图：



图二　Hello 窗体的属性设置

设置完成后，将其保存为 hello.ui 文件,这个文件就是 Hello 窗体的界面存储文件。

③ 生成 Hello 窗体类的头文件和实现文件

下面我们根据上述的界面文件 hello.ui 使用 uic 工具产生出 Hello 窗体类的头文件和实现文件，具体方法是：

```
cd qt-2.3.7/bin
uic  -o hello.h  hello.ui
uic  -o hello.cpp  -impl hello.h  hello.ui
```

这样我们就得到了 Hello 窗体类的头文件 hello.h 和实现文件 hello.cpp。下面我们就可以根据我们要实现的具体功能，在 hello.cpp 文件里添加相应的代码。

比如我们要在 Hello 的窗体上显示一个动态的字符串“Hello，World”，那么我们需要重新实现 paintEvent(QPaintEvent *)方法，同时我们还需要添加一个定时器 Qtimer 实例，以周期性刷新屏幕，从而得到动画得效果。下面是我们修改后的 hello.h 和 hello.cpp 文件。

```
/*****************************************************************************
**  以下是 hello.h 的代码
*****************************************************************************/
#ifndef HELLO_H
#define HELLO_H

#include <qvariant.h>
#include <qwidget.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;

class Hello : public QWidget
{
    Q_OBJECT

public:
    Hello( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~Hello();

//以下是手动添加的代码
signals:
    void clicked();
protected:
    void mouseReleaseEvent( QMouseEvent * );
    void paintEvent( QPaintEvent * );
private slots:
    void animate();
private:
    QString t;
    int     b;
```

```
};

#endif // HELLO_H



/*********************************************************************
**  以下是 hello.cpp 源代码
*********************************************************************/

#include "hello.h"

#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qpushbutton.h>
#include <qtimer.h>
#include <qpainter.h>
#include <qpixmap.h>

/*
 *  Constructs a Hello which is a child of 'parent', with the
 *  name 'name' and widget flags set to 'f'
 */
Hello::Hello( QWidget* parent,  const char* name, WFlags fl )
    : QWidget( parent, name, fl )
{
    if ( !name )
    setName( "Hello" );
    resize( 240, 320 );
    setMinimumSize( QSize( 240, 320 ) );
    setMaximumSize( QSize( 240, 320 ) );
    setSizeIncrement( QSize( 240, 320 ) );
    setBaseSize( QSize( 240, 320 ) );
    QPalette pal;
    QColorGroup cg;
    cg.setColor( QColorGroup::Foreground, black );
    cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
    cg.setColor( QColorGroup::Light, white );
    cg.setColor( QColorGroup::Midlight, QColor( 223, 223, 223) );
    cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
    cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
    cg.setColor( QColorGroup::Text, black );
```

```
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, black );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setActive( cg );
cg.setColor( QColorGroup::Foreground, black );
cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
cg.setColor( QColorGroup::Light, white );
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, black );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setInactive( cg );
cg.setColor( QColorGroup::Foreground, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
cg.setColor( QColorGroup::Light, white );
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setDisabled( cg );
setPalette( pal );
QFont f( font() );
f.setFamily( "adobe-helvetica" );
f.setPointSize( 29 );
f.setBold( TRUE );
setFont( f );
```

```
    setCaption( tr( "" ) );

//以下是手动添加的代码
    t = "Hello,World";
    b = 0;
    QTimer *timer = new QTimer(this);
    connect( timer, SIGNAL(timeout()), SLOT(animate()) );
    timer->start( 40 );
}

/*
 *  Destroys the object and frees any allocated resources
 */
Hello::~Hello()
{

}
/*
  This private slot is called each time the timer fires.
*/

//以下至结尾是手动添加的代码
void Hello::animate()
{
    b = (b + 1) & 15;
    repaint( FALSE );
}



/*
  Handles mouse button release events for the Hello widget.

  We emit the clicked() signal when the mouse is released inside
  the widget.
*/

void Hello::mouseReleaseEvent( QMouseEvent *e )
{
    if ( rect().contains( e->pos() ) )
        emit clicked();
}
```

```
/*  Handles paint events for the Hello widget.

  Flicker-free update. The text is first drawn in the pixmap and the
  pixmap is then blt'ed to the screen.
*/

void Hello::paintEvent( QPaintEvent * )
{
    static int sin_tbl[16] = {
        0, 38, 71, 92, 100, 92, 71, 38, 0, -38, -71, -92, -100, -92, -71, -38};

    if ( t.isEmpty() )
        return;

    // 1: Compute some sizes, positions etc.
    QFontMetrics fm = fontMetrics();
    int w = fm.width(t) + 20;
    int h = fm.height() * 2;
    int pmx = width()/2 - w/2;
    int pmy = height()/2 - h/2;

    // 2: Create the pixmap and fill it with the widget's background
    QPixmap pm( w, h );
    pm.fill( this, pmx, pmy );

    // 3: Paint the pixmap. Cool wave effect
    QPainter p;
    int x = 10;
    int y = h/2 + fm.descent();
    int i = 0;
    p.begin( &pm );
    p.setFont( font() );
    while ( !t[i].isNull() ) {
        int i16 = (b+i) & 15;
        p.setPen( QColor((15-i16)*16,255,255,QColor::Hsv) );
        p.drawText( x, y-sin_tbl[i16]*h/800, t.mid(i,1), 1 );
        x += fm.width( t[i] );
        i++;
    }
    p.end();

    // 4: Copy the pixmap to the Hello widget
    bitBlt( this, pmx, pmy, &pm );
}
```

④ 编写主函数 main()

一个 Qt/Embeded 应用程序应该包含一个主函数，主函数所在的文件名是 main.cpp。主函数是应用程序执行的入口点。以下是 Hello,World 例子的主函数文件 main.cpp 的实现代码。

```
/*****************************************************************************
**  以下是 main.cpp 源代码
*****************************************************************************/
#include "hello.h"
#include <qapplication.h>


/*
  The program starts here. It parses the command line and builds a message
  string to be displayed by the Hello widget.
*/
#define QT_NO_WIZARD

int main( int argc, char **argv )
{
    QApplication a(argc,argv);
    Hello dlg;
    QObject::connect( &dlg, SIGNAL(clicked()), &a, SLOT(quit()) );
    a.setMainWidget( &dlg );
    dlg.show();
    return a.exec();
}
```

⑤ 编辑工程文件 hello.pro 文件

到目前为止，我们为 Hello,World 例子编写了一个头文件和两个源文件，这 3 个文件应该被包括在工程文件中，因此我们需要编辑 hello.pro 文件，加入这 hello.h,hello.cpp,main.cpp 这三个文件名。具体定义如下

```
/*****************************************************************************
**  以下是 hello.pro 文件的内容
*****************************************************************************/

TEMPLATE     = app
CONFIG       = qt warn_on release
HEADERS      = hello.h
SOURCES      = hello.cpp \
             main.cpp
INTERFACES   =
```

⑥ 生成 Makefile 文件

编译器是根据 Makefile 文件内容来进行编译的，所以我们需要生成 Makefile 文件。Qt 提供的 tmake 工具可以帮助我们从一个工程文件（.pro 文件）中产生 Makefile 文件。结合当前例子，我们要从 hello.pro 生成一个 Makefile 文件的做法是：首先查看环境变量 $TMAKEPATH 是否指向 arm 编译器的配置目录，在命令行下输入以下命令

```
echo $TMAKEPATH
```

如果返回的结果的末尾不是 。。。。/qws/linux-arm-g++ 的字符串，那你需要把环境变量 $TMAKEPATH 所指的目录设置为指向 arm 编译器的配置目录，过程如下，

```
export TMAKEPATH = /tmake 安装路径/qws/linux-arm-g++
```

同时，我们还要确保当前的 QTDIR 环境变量指向 Qt/Embedded 的安装路径,如果不是，则需要执行以下过程

```
export QTDIR = ……/qt-2.3.7
```

上述步骤完成后，我们就可以使用 tmake 生成 Makefile 文件，具体做法是在命令行输入以下命令：

```
tmake -o Makefile hello.pro
```

这样我们就会看到当前目录下新生成了一个名为 Makefile 的文件。下一步，我们需要打开这个文件，做一些小的修改。

（1）将 LINK = arm-linux-gcc 这句话改为

LINK = arm-linux-g++

这样做是因为要是用 arm-linux-g++进行链接

（2）将 LIBS = $(SUBLIBS) -L$(QTDIR)/lib -lm -lqte 这句话改为

LIBS = $(SUBLIBS) -L/usr/local/arm/2.95.3/lib -L$(QTDIR)/lib -lm -lqte

这是因为链接时要用到交叉编译工具 toolchain 的库。

⑦ 编译链接整个工程

最后我们就可以在命令行下输入 make 命令对整个工程进行编译链接了。

```
make
```

make 生成的二进制文件 hello 就是可以在 FS2410 上运行的可执行文件。下面我们将介绍如何将这个文件发布到 FS2410 上。

五、发布一个 Qt/Embeded 应用到 FS2410

在优龙公司的《FS2410 开发板使用手册》的第六章“Linux 的引导和烧写”介绍了如何把 Linux 的引导和内核程序的映象文件烧写到 FS2410 的 FLASH 上。一个 Qt/Embedded 应用的运行需要有 Linux 操作系统和 Qt/Embedded 库的支持。所以我们除了要烧写 Linux 到 FS2410 的 FLASH 存储空间之外，我们还要烧写 Qt/Embedded 的二进制库到 FS2410 的 FLASH。一般来说，我们先把 Qt/Embedded 的二进制库复制到某个目录下，然后再把这个目录制成某种类型的根文件系统，最后把这个根文件系统烧写到 FS2410 的 FLASH 上，这个过程可能需要一些制作根文件系统的工具，例如 mkcramfs。在优龙公司的 FS2410 开发套件中附有的一张光盘中，有一个叫 Linux 的文件夹，包含了一个名为 qtopia.cramfs 的根文件系统映象，这个根文件系统包含了 trolltech 公司使用 Qt/Embedded 开发的一个 PDA 应用环境（简称 QPE），当用户把这个根文件系统映象烧写到 FS2410 的 FLASH 存储空间后，在 FS2410 的根文件系统下就包含了一个 Qt/Embedded 2.3.7 版本的二进制库文件。

鉴于目前发行的套件中的 uClinux 均采用 ROMFS 作为其根文件系统，因此其目录大多是不可写的。只有/var，/tmp 是 RAM 盘可写，但板子一掉电里面的内容就丢失了，因此只能作临时文件保存，无法永久的保存数据，例如配置文件等。但是/var，/tmp 这些目录可作为我们调试程序的安装目录，我们可以把编译好的 Qt/Embedded 应用程序传送到这些目录，并运行。

下面我们简单介绍一下如何把一个 Qt/Embedded 应用程序传送到 uClinux 的根文件系统的/tmp 目录。

①用 FS2410 开发套间中的串口线连接好开发用的机器的串口和 FS2410 开发板的串口。

② 在 WINDOWS 操作系统的程序|附件|通讯中运行一个超级终端，并设定通过串口 x 与 FS2410 进行连接。具体设置过程见下列图示：

图三 输入连接名称

图四 输入连接所使用的串口的端口号

图五 设置串口通讯的的参数

③ 打开 FS2410 的电源，或者复位 FS2410，这时可在超级终端的窗口中看到 BIOS 引导信息，如下图

```
channel QPE/Application/quicklauncher added
channel QPE/QuickLauncher-43 added
Registered QPE/QuickLauncher-43
Cannot suspend - no APM support in kernel

Power on reset
**********************************
*                                *
*     FS2410 Board BIOS          *
*                                *
**********************************
Please visit Http://www.uCdragon.com for more details
NAND Flash Boot

Please select function :
0 : USB download file
1 : Uart download file
2 : Write Nand flash with download file
3 : Load Pragram from Nand flash and run
4 : Erase Nand flash regions
5 : Write NOR flash with download file
6 : Set boot params
7 : Test Power off
```

④待 Linux 引导完成后，在当前文件系统的根目录下，运行“cd tmp”命令，进入到 tmp 目录，如下图

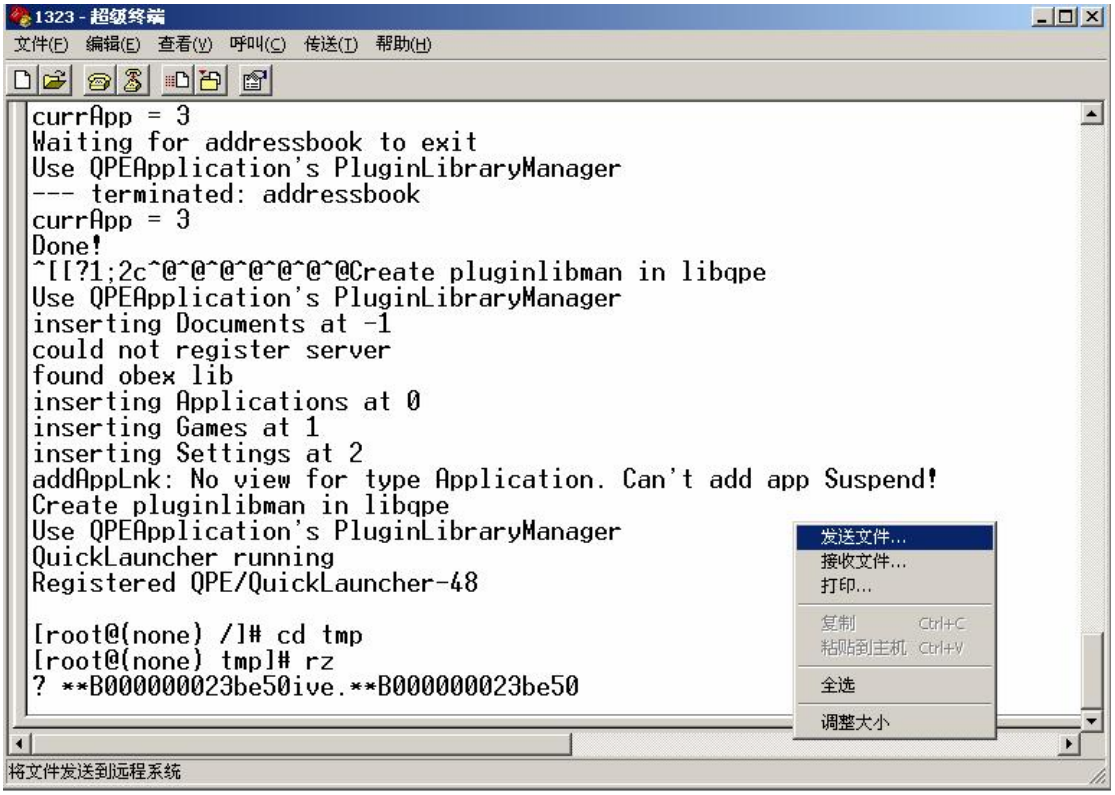
```
1323 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

Create pluginlibman in libqpe
currApp = 3
Waiting for addressbook to exit
Use QPEApplication's PluginLibraryManager
--- terminated: addressbook
currApp = 3
Done!
^[[?1;2c^@^@^@^@^@^@^@Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
inserting Documents at -1
could not register server
found obex lib
inserting Applications at 0
inserting Games at 1
inserting Settings at 2
addAppLnk: No view for type Application. Can't add app Suspend!
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
QuickLauncher running
Registered QPE/QuickLauncher-48

[root@(none) /]# cd tmp
[root@(none) tmp]#

连接的 0:01:40 VT100J 115200 8-N-1 SCROLL CAPS NUM 捕 打印
```

④ 在当前命令行下输入“rz”命令，开始一个文件传输过程；接着点击鼠标右键，在弹出的菜单中，点击“发送文件”子菜单，然后选择你要传送到 tmp 目录的 Qt/Embedded 应用程序。



例如，下图是正在传送 Hello,World 例子的可执行文件到 tmp 目录

⑤ 最后我们就可以在 tmp 目录下运行传送过来的一个 Qt/Embedded 应用程序。

六、添加一个 Qt/Embedded 应用到 QPE

前面我们提到了优龙光盘里面有一个名为 qtopia.cramfs 的根文件系统映象，烧写这个根文件系统到 FS2410 的 FLASH，就会安装有 QT PDA 应用环境（简称 QPE）。我们可以在 QPE 里添加我们自己编写的应用，不过添加的过程需要重新交叉编译 QPE 的源文件。下面我们介绍如何在 QPE 里添加我们编写的 Hello，World 的例子。

① 在工作的机器上解包 qtopia

```
tar zxvf qtopia-free-1.7.x.tar.gz
cd qtopia-free-1.7.x
export QTDIR=$QTEDIR
export QPEDIR=$PWD
export PATH=$QPEDIR/bin:$PATH
```

注意在上面我们已经设定环境变量 QPEDIR 为 QPE 的安装（解包）路径。

② 建立 Hello，World 的例子程序的图标文件

方法是：制作一个 32 X 32 大小的 PNG 格式的图标文件，将该文件存放在 $QPEDIR/pics/inline 目录下，然后使用以下命令将$QPEDIR/pics/inline 目录下的所有图形文件转换成为一个 c 语言的头文件，这个头文件包含了该目录下的图形文件的 rgb 信息。

```
qembed --images $QPEDIR/pics/inline/*.*
        > $QPEDIR/src/libraries/qtopia/inlinepics_p.h
```

注意上述的 qembed 是在 qt3.x.x for x11 上才发布有的工具

③ 交叉编译 qtopia

在$QPEDIR 路径下，运行以下命令

```
cd src
./configure -platform linux-arm-g++
make
cd ..
```

④ 建立应用启动器（.desktop）文件

方法是：建立一个文本文件，在文件中添加以下的内容，这些内容指明了应用的名称，图标名等信息，然后将文件更名为 xxxx.desktop，保存在$QPEDIR/apps/applications 目录下。以下是我们例子程序的启动器文件（hello.desktop）：

```
[Desktop Entry]
Comment=A Hello Program
Exec=hello
Icon=Hello
Type=Application
Name=hello
```

⑥ **建立根文件系统**

在这里我们利用原有的 qtopia.cramfs 的根文件系统映象，把我们新建的应用的相关文件添加到这个根文件系统中。首先我们先要把 qtopia.cramfs 的根文件系统 mount 到我们工作机器上来，然后复制这个文件系统的内容到一个临时的 temp 目录，这时我们可以在 temp\Qtopia 目录下看到一个 qtopia-free-1.7.0 目录，这就是 qtopia.cramfs 的根文件系统里的 qpe 安装目录，接着把我们的新建的应用的相关文件（包括启动器文件，包含了图标的库文件 libqte.so.*,和应用程序的可执行文件)复制到 temp/Qtopia/ qtopia-free-1.7.0 的对应的目录。具体过程如下

```
mkdir /mnt/cram
mount -t cramfs qtopia.cramfs /mnt/cram -o loop
cp -ra /mnt/cram temp/
cp $QPEDIR/apps/Applications/hello.desktop （接下行）
          temp/Qtopia/qtopia-free-1.7.0/apps/Applications/
cp $QPEDIR/lib/libqpe.so.* （接下行）
          temp/Qtopia/qtopia-free-1.7.0/lib/
cp hello temp/Qtopia/qtopia-free-1.7.0/bin （拷贝可执行文件）
mkcramfs temp xxxxxx.cramfs          (生成新的根文件系统)
```

将生成的新的根文件系统烧写到 FS2410 的 FLASH 根文件系统区，复位，OK,就可以看到 QPE 里有我们编写的应用的图标了，点击这个图标，程序就成功运行了。当然为了使我们的应用和原先的 QPE 的应用具备统一的界面风格，我们在编写我们自己的应用的主函数文件（main.cpp）时，不妨使用 QPE 提供的宏，具体可参考 QPE 应用程序的源文件。下面是我们编写的 Hello,world 程序在 qvfb 的截屏图。

小记

开发 Qt/Embedded 应用对初学者可能是一项艰苦的工作过程，因为需要安装和设置很多的内容，有时候某一过程没有进行可能会导致一些莫名奇妙的出错提示。尽管我们的开发文档详细的介绍了嵌入式 Qt 的开发过程，然而我们还是不能保证初学者一步一步按照我们所描述的去做便可以在编译应用时万无一失，因为 linux 开发包之间有一定的依赖性，这些开发包又从属不同的开发商或组织。我们的建议是您在您的机器上安装 linux 并准备进行开发时，至少要安装一个工作站版的 linux。您需要注意您的机器上安装有 libjpeg, e2fsprogs, Freetypes 等开发包有否安装，因为上述我们谈到的 Qt 嵌入式的软件都用到了这些开发包。您不必为这些繁琐的工作感到沮丧，因为这些工作会让你有机会认识嵌入式软件工作的底层细节。您会把嵌入式软件的开发过程当作是在玩一个锻炼智力的玩具吗？您很快会有喜欢上这种玩具的感觉的！