CircleOS Tutorial

for STM32-Primer2

Application Note AN0054

Document version

2009-01-13

Contents

1. Introduction

This document will guide you through the construction of your first STM32-Primer2 application based on CircleOS. You will also learn how to take advantage of your STM32-Primer2 features such as the LCD display or the embedded 3D MEMS accelerometer.

The Ride7 software toolset, along with its RKit-ARM plug-in, provides everything needed for programming your STM32-Primer2 and debugging applications, including:

1. USB host connection for in-circuit programming and debugging
2. Ride7 integrated development environment for code editing, device programming and application debugging (debug up to 32K of code, with included version. For information about upgrade to an unlimited version of Ride, visit http://www.stm32circle.com/resources).
3. GNU C /C++ compiler (unlimited compiling)

CircleOS is an Operating System that provides components and services specific to your STM32-Primer32. It will help you write applications without having to understand the deep details of the hardware implementation.

This document assumes that Ride7 and its RKit-ARM plug-in are already installed on your PC. If this is not the case, please head to the http://www.stm32circle.com web site and download them.

1.1 Conventions Used in this Manual

File | New	Refers to the menu item “New” on the File menu.
While(1);	(bold, monospaced type) User input
filename	Replace the italicized text with the item it represents
[ ]	Items inside [and] are optional.
[…]	Represents a list of optional items that are the same as the preceding item.
`while(1);`	(monospaced type) Code, Directives and software generated output
0000H	A Hexadecimal value (assembly format)
0x0000	A Hexadecimal value (C language format)

The Ride installation directory will be referred to as “%ridedir%” in this document. For instance, if you installed Ride in the default “C:\Program Files\Raisonance\Ride” directory, “%ridedir%\lib” will refer to the actual “C:\Program Files\Raisonance\Ride\lib” directory.

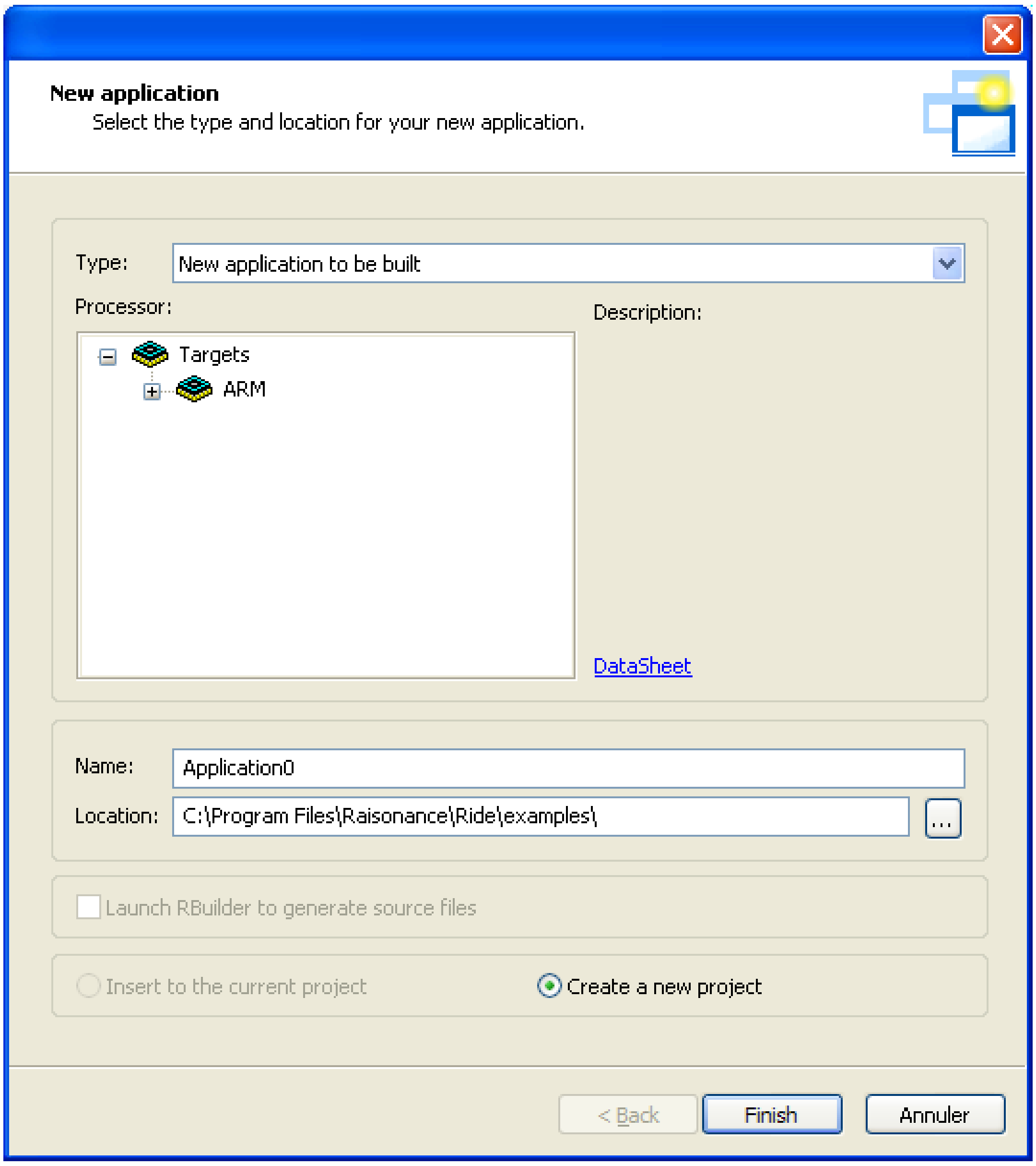
2. Creating a skeleton project

2.1 Launch Ride7

- From the start menu of your computer, select the “Programs > Raisonance Tools” group.
- Click on the Ride7 icon.

2.2 Create the project using the application Wizard

From the top menu in Ride7, click on the command "Project | New Project". The “New application dialog box will appear:



New application
Select the type and location for your new application.

Type: New application to be built

Processor:
- Targets
  - ARM

Description:

DataSheet

Name: Application0

Location: C:\Program Files\Raisonance\Ride\examples\

[ ] Launch RBuilder to generate source files

Insert to the current project [✓ Create a new project]

- Click on the “ARM” target family to expand the list of targets.
- Select the “STM32_Primer2_CircleOS derivative.
- Specify the name of the first application: “Cubic_Step0”
- Specify a new directory for "Location" (to be created by Ride7): “c:\tmp\Cubic_step0”,
- Click the "Finish" button.

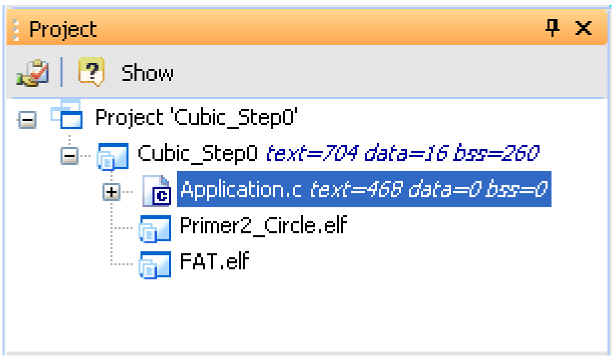
Ride7 will then automatically prepare the skeleton project for your application, which does not perform any specific action.

2.3 Build the project

- From the top menus in Ride7, click on the command "Project | Build Project".
- The “Build log” Ride7 window will display the compiler/linker results during the build operation.
- The "Build succeeded" message should then appear.

Congratulations: You have built your first CircleOS application!

Now let's personalize your application a little bit. From the list of files in the project window, open the “Application.c” file by double-clicking it in the Project window:



Project 'Cubic_Step0'
Cubic_Step0 *text=704 data=16 bss=260*
Application.c *text=468 data=0 bss=0*
Primer2_Circle.elf
FAT.elf

You can explore the contents of this file, which is the basic framework of the application.

You will find the following elements:

- A string "Application_Name" that will be used in the “Applications” menu of your STM32-Primer2. This is the name of the application. You can change it, but ensure that you do not make it longer than 8 characters. For instance, let's change the application name to “Cubic-0”:

```
const char Application_Name[8+1] = {"Cubic-0"};
```

- The *Application_Ini()* and *Application_Handler()* functions that will be called by the main/Applications STM32-Primer2 menus. When a command is launched from this menu, the Application_Ini() function is called only once; the Application_Handler() function will be called repeatedly by the Systick interrupt handler.
  In the basic framework application, the Application_Ini() function just checks the CircleOS version. Application_Handler() is empty.

After you have done these changes, rebuild your application through the “Project | Make Project” menu. You will see the message "Build succeeded" in the Build log window.

2.4 Program your application on your STM32-Primer2

In order to program your application on your Primer, just follow these steps:

- Connect your Primer to an available USB port on your PC. On the Primer, there are two USB connectors named (moulded into the plastic on the rear) “STM32” and “DEBUG”. You must connect the supplied USB cable to the port named “DEBUG” which will connect the embedded RLink hardware programmer/debugger. The port named “STM32” is connected to the USB peripheral on the STM32 Microcontroller and can be used by your Primer applications.

- Power-on your Primer using its orange button.

Note: If your Primer is not powered on, you will not be able to program it:
In such a case, a “Target power failure” alert message will be displayed. You just have to power-on your Primer2 and restart the programming operation.

- Upon first connection, Windows will detect a new hardware. When requested, instruct Windows *not* to search for updates, then click on the automatic installation procedure so that your Primer2 is properly configured by your PC.

- You can launch the debug session of your project by running the "Debug | Start" command in the main menu. This action will erase the Flash on the Primer and re-load it with the necessary information to debug your current application.

- After loading and resetting, the program execution will stop at the "main()" function of the CircleOS.

- Run the program through the “Debug | Run” command. Your Primer is now in debug mode, under Ride7 control.

- On the Primer, press the orange button once. You will see that our new Cubic-0 application is present in the main menu. Highlight your “Cubic-0” application then press the orange button. This will launch your Cubic-0 application.

Note: As your application does not do anything yet, your Primer2 will “freeze” once you start your Cubic-0 application. To reset your Primer2 you just have to hold its orange button for few seconds, which will shut off the Primer. Once you start it again it will show its main menu again.

2.4.1 Information about the other files in the project

As you can see from Ride7's “Project” window, your project contains not only the "Application.c" source file, but also 2 other files:

- Circle.elf: this is the complete CircleOS binary. It contains many functions and utilities that can be called from the applications. Documentation for CircleOS and how to use its functions can be found on http://www.stm32circle.com.

- FAT.elf: the FAT contains the list of the available applications, and a pointer to each of them. It is used by CircleOS to find all the applications available on your Primer, and makes it possible to independently add/remove lots of different applications in ROM. The FAT is usually built by the "circle_mgr.exe" utility when loading a new application. However, for debugging purposes Ride7 will load a fixed, single-application FAT that contains only one pointer to one application located at the address 0x6000 in the FLASH.

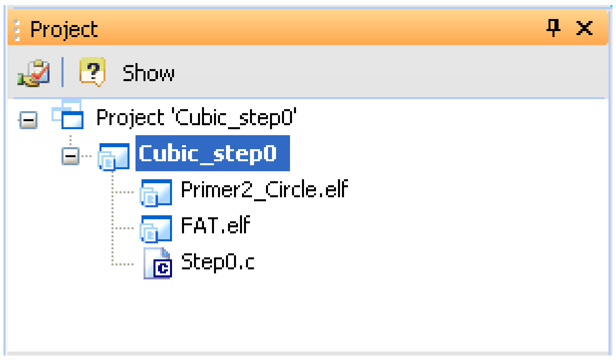
3. Drawing a square in your application

3.1 Creating a second application within the existing project

First of all, we can change the name of the existing "Application.c" file of our first step. We have to do that to avoid overwriting our first "application.c" file:

- Open the file “Application.c”
- Using the "File | Save As" command, rename this file "Step0.c"
- In the project menu, remove the reference to Application.c: Right-click on the “Application.c” file in the project window, and then select "Remove" from the popup menu.
- Insert the new file named "Step0.c" in the application: Right-click the Application “Cubic_Step0” (not the top line in the project window which is preceded by the word ‘project’ but the line immediately following it), select "Add | Item" then point to the”Step0.c” file you just created.

Your Ride7 “Project” window should now look as follows:



We can now create our new application:

1. Right-click the main project node (Project “Cubic_Step0” on the first line).
2. In the popup menu, select the “Add | Add New Application” command.
3. Specify "Cubic_Step1" as name for the new application.
4. Select the “STM32_Primer2_CircleOS” ARM derivative.
5. Click on “Finish".

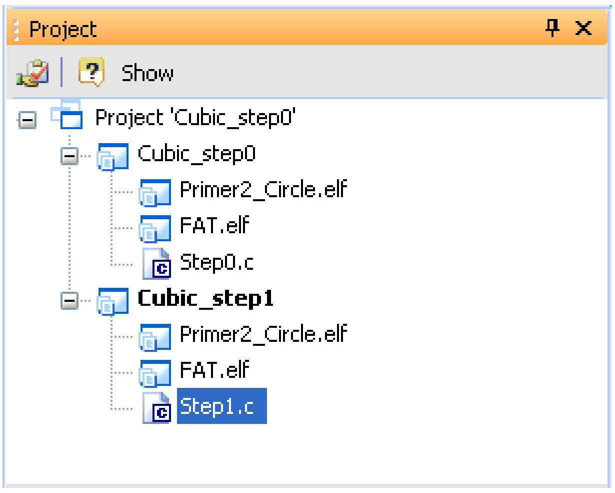
We now have two applications in the same project:

Let's set the new application as the default one for the whole project:

- Right-click the new "Cubic_Step1" application
- Select "Set as Startup Application" in the popup menu.

Now, make a copy of “Step0.c” with the name “Step1.c”, and remove the ‘Application.c’ as in the previous steps then add Step1.c to the new application (after selecting the Cubic_step1 application, choose “Add | Add Existing Item”).

Your project window should look like this:



Double-click the Step1.c file to open it in the Ride7 editor window, then in the source code change the Application Name to "Cubic-1".

3.2 Drawing a square

Next we are going to draw a small square in the center of the LCD display.

The display area is 128 X 128 pixels large (although the whole display is 160x128, but some display area is reserved for application buttons).

Let's choose, for example, to draw a 32X32 square.

We need to add the following definition/declarations to the source code in Step1.c. They define the size and location of the square on the screen. Place them near the top of the source code, in the “Public variables” section:

```
#define  SQUARE_WIDTH 32
int x = ( SCREEN_WIDTH - SQUARE_WIDTH ) / 2;
int y = ( SCREEN_HEIGHT - SQUARE_WIDTH ) / 2;
```

As we want to draw the square only once we can add the drawing code into the Application_Ini() function (remember, Application_ini() is only called once when the application starts).

In the box below you will see the source code that draws the square.

Copy this block of code into the function Application_Ini() function body, in the “step1.c” file (replace the “TODO: Write your application initialization function here” by this code:

```
    //Draw my square
    LCD_DrawRect( x, y, SQUARE_WIDTH, SQUARE_WIDTH, RGB_BLUE);
    LCD_FillRect( x+1, y+1, SQUARE_WIDTH-2, SQUARE_WIDTH-2, RGB_YELLOW);
```

The first function LCD_DrawRect() will draw (in blue) the square periphery, and the second fills the area inside the square (in yellow).

For this step, we should also modify the contents of the function Application_Handler() to allows us to return to the main menu from the application.

To do this, we have to restore the context of the main menu. We will use a simple button press to end the application and return to the main menu.

The following block of code should be added into the Application_Handler() function (in place of the “TODO: Write your application handling here” comment) to handle the button-press event:

```
    if( BUTTON_GetState() == BUTTON_PUSHED )
        {
        BUTTON_WaitForRelease();
        BUTTON_SetMode( BUTTON_ONOFF_FORMAIN );
        return MENU_Quit();
        }
```

Now build the application as you have done before, then debug and run your application.

You will see “Cubic1” application in the main menu of your Primer2. When you select it, the application will draw a square on the screen. Pressing the orange button will exit your application and CircleOS will come back to the main menu.

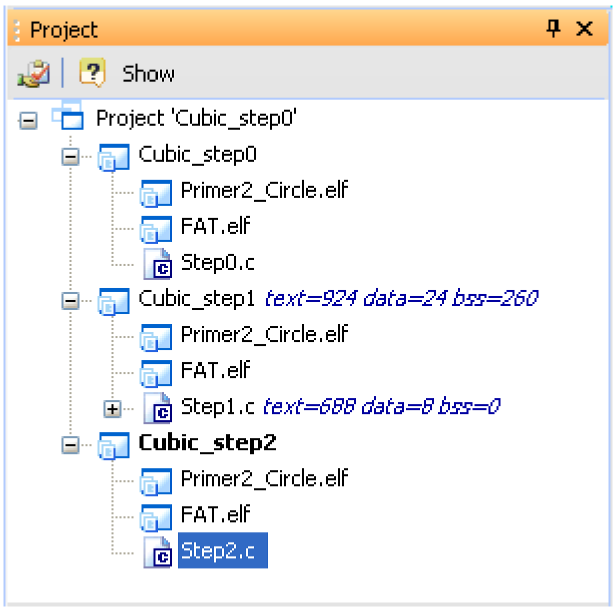
Note: Your application is responsible for restoring the hardware context of the main menu when exiting. In case the application has been modifying some general parameters (CPU speed, font size, ...) they should be restored when closing the application.

4. Using the information from the MEMS

As we did during Step1, create a new application within the project called Cubic_Step2 that contains a file named step2.c that is a copy of our current step1.c (see the first part of the previous chapter for a reminder on how to do this).

Again, we must make this new application as the current one (right-click to get the popup menu, then select “Set As Startup Application”).

Your project should now look like this:



```
Project 'Cubic_step0'
  Cubic_step0
    Primer2_Circle.elf
    FAT.elf
    Step0.c
  Cubic_step1 text=924 data=24 bss=260
    Primer2_Circle.elf
    FAT.elf
    Step1.c text=688 data=8 bss=0
  Cubic_step2
    Primer2_Circle.elf
    FAT.elf
    Step2.c
```

4.1 Reading the information from the MEMS

In this part of the tutorial we will modify the height of the square depending on the orientation of the Primer, using data taken from the on-board 3D MEMs sensor (for more information on the MEMS sensor go to http://www.st.com/mems).

The function MEMS_GetInfo() function returns a pointer to a structure that contains the current state of the MEMS information. For a detailled description of this function/structure (and all others used in CircleOS), please refer to the CircleOS documentation which can be found at http://www.stm32circle.com/circleos_doc/.

CircleOS permanently updates the contents of the structure pointed to by MEMS_GetInfo(). This is done through an interrupt (Timer2 IRQ) with a high priority.

If we want the current MEMS output value on the Y axis, we can use the OutY value:

```
s16 Current_MEMS_Y = MEMS_GetInfo()->OutY;
```

The MEMs device is very sensitive and the output will change with the slightest of movements so it may be better in some circumstances to filter the results from the MEMS sensor. If we want a value filtered on the last 4 acquisitions, we can use OutY_F4 instead of OutY:

```
s16 Current_MEMS_Y = MEMS_GetInfo()->OutY_F4 >> 2;
```

A filtered value will provide more stability. Note that OutY_F4 value must be divided by 4 (it corresponds to the sum of the last 4 values).
Next we need to calculate the movement:

```
s16 delta = Current_MEMS_Y - Initial_MEMS_Y;
```

The value given by the MEMS is typically in the range of [-1000, +1000]. The unit is "mg" (1g/1000). Therefore we can (for example) modify the height of the rectangle by looking at the "delta/100" , i.e. we will typically add or remove 10-20 pixels to the height.

We need to add code in several places to make this part of the application work, in the “Public variables” section at the top of the source code add:

```
s16 Initial_MEMS_Y;
```

The Application_Ini()function should contain:

```
    //Draw my square
    LCD_DrawRect( x, y, SQUARE_WIDTH, SQUARE_WIDTH, RGB_BLUE);
    LCD_FillRect( x+1, y+1, SQUARE_WIDTH-2, SQUARE_WIDTH-2, RGB_YELLOW);

    Initial_MEMS_Y = (MEMS_GetInfo())->OutY_F4;
```

The contents of the Application_Handler() function is shown on the next page.

The Application_Handler() function should contain:

```
enum MENU_code Application_Handler( void )
    {
    s16 Current_MEMS_Y, delta;
    s16 y1, h1;

    if( BUTTON_GetState() == BUTTON_PUSHED )
        {
        BUTTON_WaitForRelease();
        BUTTON_SetMode( BUTTON_ONOFF_FORMAIN );
        return MENU_Quit();
        }

    Current_MEMS_Y = (MEMS_GetInfo())->OutY_F4;
    delta = Current_MEMS_Y - Initial_MEMS_Y;
    DRAW_SetDefaultColor();
    DRAW_Clear();

    y1 = ( ( SCREEN_HEIGHT - SQUARE_WIDTH ) / 2 ) - (delta/100);
    h1 = SQUARE_WIDTH + 2 * (delta/100);

    if( h1 < 2 )
        {
        y1 = SCREEN_HEIGHT / 2 - 1;
        h1 = 2;
        }
    else if( h1 > SCREEN_HEIGHT )
        {
        y1 = 0;
        h1 = SCREEN_HEIGHT;
        }

    LCD_DrawRect( x, y1, SQUARE_WIDTH, h1, RGB_BLUE);
    LCD_FillRect( x+1, y1+1, SQUARE_WIDTH-2, h1 - 2, RGB_YELLOW);

    return MENU_CONTINUE;   // Returning MENU_LEAVE will quit to CircleOS
    }
```

All the necessary code for step2 is now in place.

Build the application and then start the debug. When you Run the software and enter into your application from the main menu you will see the rectangle changes size as you move the Primer.

5. Optimizing screen updates

In the previous step, the complete rectangle is redrawn on screen for every adjustment in position. This takes a lot of time, as the Primer2 display controller is not able to handle such complex operations, so each pixel must be independently drawn on the screen.

In this step of the tutorial, you will learn how to optimize the rectangle update.

Start by adding a new project to the application just as before and this time call it Cubic_Step3. Again, copy and rename the step2.c file into step3.c and add it to the Cubic_Step3 application. Remember to use the right-click popup menu to select the Cubic_step3 application as the current one. If you need a reminder on how to do any of this, just refer to the earlier stages in this tutorial.

In Step2 the complete rectangle was redrawn for every change in data from the MEMS sensor. This is quite time consuming, so it would be better to only draw the parts of the rectangle that have changed.

A new line in the definitions section is required to support the new code, that you must add to the “Public variables” section at the top of the source code:

```
s16 Current_Pos = ( SCREEN_HEIGHT - SQUARE_WIDTH ) / 2;
```

The complete Application_Handler() function with the new code from above should look like:

```
enum MENU_code Application_Handler( void )
    {
    s16 Current_MEMS_Y, delta;
    s16 y1, h1;

    if( BUTTON_GetState() == BUTTON_PUSHED )
        {
        BUTTON_WaitForRelease();
        BUTTON_SetMode ( BUTTON_ONOFF_FORMAIN );
        return MENU_Quit();
        }

    Current_MEMS_Y = (MEMS_GetInfo())->OutY_F4;
    delta = Current_MEMS_Y - Initial_MEMS_Y;
    DRAW_SetDefaultColor();
    DRAW_Clear();

    y1 = ( ( SCREEN_HEIGHT - SQUARE_WIDTH ) / 2 ) - (delta/100);
    h1 = SQUARE_WIDTH + 2 * (delta/100);
    if( h1 < 2 )
        {
        y1 = SCREEN_HEIGHT / 2 - 1;
        h1 = 2;
        }
    else if( h1 > SCREEN_HEIGHT )
        {
        y1 = 0;
        h1 = SCREEN_HEIGHT;
        }

    if( y1 == Current_Pos )
        {
        return MENU_CONTINUE;   // Returning MENU_LEAVE will quit to CircleOS
        }

    //Draw only the modified area
    else if( y1 < Current_Pos ) //rectangle is expanded
        {
        LCD_FillRect( x+1, y1+1, SQUARE_WIDTH-2, Current_Pos - y1, RGB_YELLOW);
        LCD_FillRect( x+1, Current_Pos + Current_Height - 2, SQUARE_WIDTH-2,
                      Current_Pos - y1 + 2, RGB_YELLOW);
        LCD_DrawRect( x, y1, SQUARE_WIDTH, h1, RGB_BLUE);
        }
    else //( y1 > Current_Pos ) //rectangle is shrinked
        {
        LCD_FillRect( x, Current_Pos, SQUARE_WIDTH, y1 - Current_Pos, RGB_WHITE );
        LCD_FillRect( x, y1 + h1 - 1, SQUARE_WIDTH, y1 - Current_Pos + 1, RGB_WHITE);
        LCD_DrawRect( x, y1, SQUARE_WIDTH, h1, RGB_BLUE);
        }

    //Keep the current pos/height for the next time.
    Current_Pos    = y1;
    Current_Height = h1;

    return MENU_CONTINUE;   // Returning MENU_LEAVE will quit to CircleOS
    }
```

Now you should build the code and enter into debug.

Run the application and select it from the main menu on the Primer.

You should now see that the movement of the rectangle in relation to the movement of the Primer is smoother. Our optimization has been efficient.

6. Writing a bubble level application

As with the previous steps in this tutorial, start by adding a new application to the project, this time call it Cubic_Step4. Copy and rename step3.c to step4.c and add it to the new application.

In this part of the tutorial we will develop the code from Step3 to create a ‘bubble level’ working on two axis.
The concept here is when your Primer is level, the rectangle drawn will be reduced to a small dot, but as you tip the primer the rectangle will be redrawn to show the direction and a graphical representation of the angle.

This time, we will just explain the changes made from Step3 as we will not be starting from scratch.
Instead of starting from the current position, we assume that:

```
    Initial_MEMS_X = 0;
    Initial_MEMS_Y = 0;
```

The size of the rectangle will be null when MEMS_outX and MEMS_outY are 0.

The orientation of the screen must be fixed in the Application_Ini() function in order to draw the rectangle following the physical orientation information from the MEMS sensor. Just add the following line in Application_Ini():

```
    LCD_SetScreenOrientation( V12 );
```

In Application_Handler(), we now manage both X and Y axis, in the same way.

When you keep the Primer horizontal, the rectangle is reduced to a simple dot. When you change the orientation, the dot becomes a rectangle.
This a simple bubble level working on two axis.

A few additional hints on how the code is working:

- Instead of calling the MEMS_GetInfo() function for each parameter, we can save the multiple calls by defining a pointer "pMEMS_info" as global variable, initialized in the Application_Ini() routine.
- In the Step4, we add a line that shows the direction of the gravity. The CircleOS does not provide a DrawLine universal function (it only allows horizontal/vertical lines) and the Bresenham's line algorithm needs to be implemented in the application.
- "LCD_SetRotateScreen(1)" has been added to restore the ability of rotating the screen when the orientation changes, just like the Primer did when it was new from the box.
- Last, we add the OUTX / OUTY values. To display a string of characters, the following functions are called: DRAW_DisplayString () and DRAW_DisplayStringInverted ()
- Also used is UTIL_int2str () that converts a signed integer into a string.

Here is all the code for the Bubble Level Application:

The definitions section requires the following definitions, that you must add to the "Public variables" section at the top of the source code:

```
    const char Application_Name[8+1] = {"Cubic-4"};

    int x = ( SCREEN_WIDTH ) / 2;
    int y = ( SCREEN_HEIGHT ) / 2;
    s16 Current_Height = 2;
    s16 Current_Width  = 2;
    s16 Current_PosY    = ( SCREEN_HEIGHT ) / 2;
    s16 Current_PosX    = ( SCREEN_HEIGHT ) / 2;
    u16 Current_Color = RGB_YELLOW;

    s16 Initial_MEMS_X, Initial_MEMS_Y;
```

Following is the code for the Application_Ini() function. Copy this code into your Step4.c file:

```
enum MENU_code Application_Ini( void )
    {
    if(strcmp(UTIL_GetVersion(), NEEDEDVERSION) < 0)
        {
        return MsgVersion();
        }
    Initial_MEMS_X = 0;
    Initial_MEMS_Y = 0;
    LCD_SetScreenOrientation( V12 ); //Match orientation with MEMS information

    //Draw small indicators to show the direction
#define SIGN_SIZE 8
    LCD_FillRect( 1, 1, SIGN_SIZE, SIGN_SIZE, RGB_RED);
    LCD_FillRect( SCREEN_WIDTH - SIGN_SIZE - 1, 1,
                  SIGN_SIZE,SIGN_SIZE, RGB_GREEN);
    LCD_FillRect( 1, SCREEN_HEIGHT - SIGN_SIZE - 1,
                  SIGN_SIZE,SIGN_SIZE, RGB_MAGENTA);
    LCD_FillRect( SCREEN_WIDTH - SIGN_SIZE - 1,
                  SCREEN_HEIGHT - SIGN_SIZE - 1,
                  SIGN_SIZE,SIGN_SIZE, RGB_YELLOW);

    return MENU_CONTINUE_COMMAND;
    }
```

And the following are the Application_Handler() function contents:

```
enum MENU_code Application_Handler( void )
    {

    s16 Current_MEMS_X, Current_MEMS_Y, deltaX, deltaY;
    s16 x1, y1, w1, h1;
    u16 square_color = RGB_YELLOW;

    if( BUTTON_GetState() == BUTTON_PUSHED )
        {
        BUTTON_WaitForRelease();
        BUTTON_SetMode  ( BUTTON_ONOFF_FORMAIN );
        return MENU_Quit();
        }

    Current_MEMS_X = (MEMS_GetInfo())->OutX_F4;
    Current_MEMS_Y = (MEMS_GetInfo())->OutY_F4;
    deltaX = Current_MEMS_X - Initial_MEMS_X;
    deltaY = Current_MEMS_Y - Initial_MEMS_Y;
    if( deltaX < 0 )
        {
```

```
        deltaX = -deltaX;
        square_color = RGB_MAGENTA;
        if( deltaY < 0 )
            {
            deltaY = -deltaY;
            square_color = RGB_RED;
            }
        }
    else if( deltaY < 0 )
        {
        deltaY = -deltaY;
        square_color = RGB_GREEN;
        }

    if( square_color != Current_Color )
        {
        Current_Color = square_color;
        LCD_DrawRect( Current_PosX, Current_PosY, Current_Width,
                      Current_Height, RGB_BLUE);
        LCD_FillRect( Current_PosX + 1, Current_PosY + 1, Current_Width - 2,
                      Current_Height - 2, Current_Color);
        }

    DRAW_SetDefaultColor();

    x1 = ( ( SCREEN_WIDTH  ) / 2 ) - (deltaX/100);
    y1 = ( ( SCREEN_HEIGHT ) / 2 ) - (deltaY/100);
    w1 = 2 * (deltaX/100);
    h1 = 2 * (deltaY/100);

#define MAX_SIZE_RECT (SCREEN_WIDTH - 2*(SIGN_SIZE+2))
    //keep width in the range
    if( w1 < 2 )
        {
        x1 = SCREEN_HEIGHT / 2 - 1;
        w1 = 2;
        }
    else if( w1 > MAX_SIZE_RECT )
        {
        x1 = 0;
        w1 = MAX_SIZE_RECT;
        }

    //keep height in the range
    if( h1 < 2 )
        {
        y1 = SCREEN_HEIGHT / 2 - 1;
        h1 = 2;
        }
    else if( h1 > MAX_SIZE_RECT )
        {
        y1 = 0;
        h1 = MAX_SIZE_RECT;
        }

    //Now redraw the rectangle
    if( x1 != Current_PosX )
        {
        //Draw only the modified area
        if( x1 < Current_PosX ) //rectangle is expanded
            {
            LCD_FillRect( x1+1, Current_PosY+1, Current_PosX - x1, Current_Height-2,
                          square_color);
            LCD_FillRect( Current_PosX + Current_Width - 2, Current_PosY + 1,
                          Current_PosX - x1 + 2, Current_Height-2, square_color);
            LCD_DrawRect( x1, Current_PosY, w1, Current_Height, RGB_BLUE);
            }
        else //( x1 > Current_PosX ) //rectangle is shrinked
            {
            LCD_FillRect( Current_PosX, Current_PosY, x1 - Current_PosX,
                          Current_Height, RGB_WHITE );
            LCD_FillRect( x1 + w1 - 1, Current_PosY, x1 - Current_PosX + 1,
                          Current_Height, RGB_WHITE);
            LCD_DrawRect( x1, Current_PosY, w1, Current_Height, RGB_BLUE);
            }

        //Keep the current pos/height for the next time.
        Current_PosX    = x1;
        Current_Width   = w1;
        }

    if( y1 != Current_PosY )
        {
        //Draw only the modified area
        if( y1 < Current_PosY ) //rectangle is expanded
            {
            LCD_FillRect( Current_PosX+1, y1+1, Current_Width-2,
                          Current_PosY - y1, square_color);
            LCD_FillRect( Current_PosX+1, Current_PosY + Current_Height - 2,
                          Current_Width-2, Current_PosY - y1 + 2, square_color);
            LCD_DrawRect( Current_PosX, y1, Current_Width, h1, RGB_BLUE);
            }
        else //( y1 > Current_PosY ) //rectangle is shrinked
            {
            LCD_FillRect( Current_PosX, Current_PosY, Current_Width,
                          y1 - Current_PosY, RGB_WHITE );
            LCD_FillRect( Current_PosX, y1 + h1 - 1, Current_Width,
                          y1 - Current_PosY + 1, RGB_WHITE);
            LCD_DrawRect( Current_PosX, y1, Current_Width, h1, RGB_BLUE);
            }

        //Keep the current pos/height for the next time.
        Current_PosY    = y1;
        Current_Height  = h1;
        }

    return MENU_CONTINUE;   // Returning MENU_LEAVE will quit to CircleOS
    }
```

Now build the application, go into debug and run the application.

Select it from the main menu on the Primer and play about with the Bubble Level.

7. Distributing your application

We have seen how to create new Primer2 applications, program and debug them.

Once your application is finished i.e., once the required features are properly implemented in your application, it is possible to distribute it in binary form.

Distributing your application in binary form has several advantages:

- It is not necessary to recompile the application in order to load it on a Primer2.
- The CircleOS binaries can be upgraded without having to reload your application once this operation is performed.
- Many different applications can be loaded on a single Primer (although no debugging capability is offered in this mode). The number of loadable applications is in fact limited only by your Primer flash size.

Depending on your needs, your application sources may fit in one single C file, or may need several of them.

For single-file applications

When your application fits into a single C file, you just have to distribute the object file (with extension ".o") for other people to load your application on their own Primer2. You can click on the "add_to_circle.bat" file which is installed by the CircleOS application wizard in your "objdebug" directory. This will load your application on the connected Primer.

For multiple-files applications

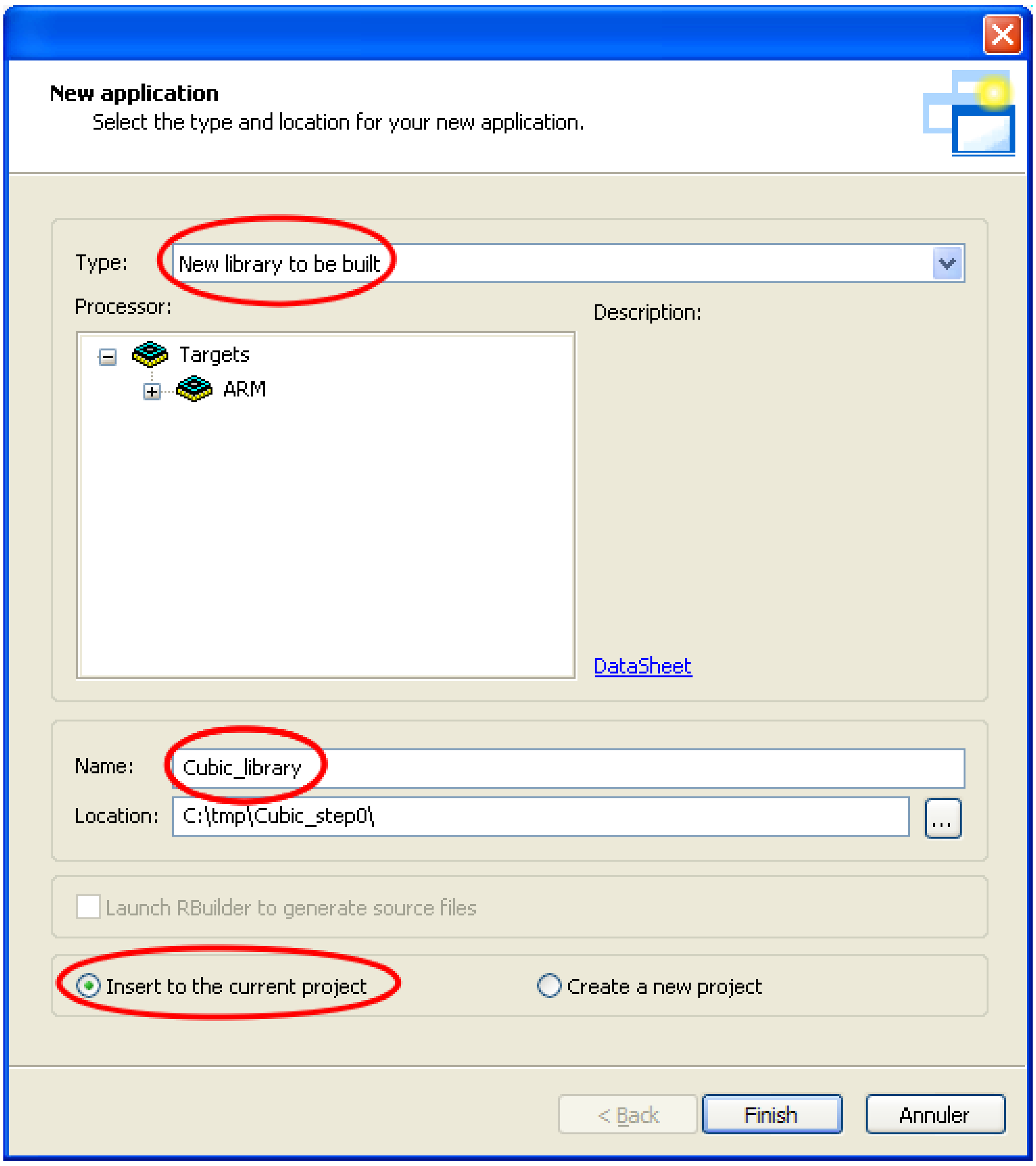
In case your application requires several C files, each of these C files will produce an object file. You just have to aggregate those object files into a library file (with ".lib" extension). Once this is done, you can click on the "add_to_circle.bat" file which is installed by the CircleOS application wizard in your "objdebug" directory. This will load your application on the connected Primer.

Building a library containing all your object files

To build a library with all your project files, right-click on your Project and then select the "Add | Add new application" from the popup window.

A window such as the following will appear.

- In the “Type” box select “New library to be built”
- In the Name field enter your library name
- Select the “Insert to the current project” button
- Then click the “Finish” button.



New application

Select the type and location for your new application.

Type: New library to be built

Processor: Targets / ARM

Description:

Name: Cubic_library

Location: C:\tmp\Cubic_step0\

[ ] Launch RBuilder to generate source files

[✓ Insert to the current project] Create a new project

You then have a new node in your project window that corresponds to your library.

- In case the “Application.c”, “FAT.elf” and “Circle.elf” elements are present in the library node, remove them (right-click and select the “remove” command).
- Add each of your C files in the library by right-clicking the library node and selecting "Add | Item", then pointing to each C file to be added.
- Build your library: Right-click your library node and select the “Make” command from the popup window.
- Your library will then be built, available as a “.lib” file

- The “add_to_circle.bat” script can then be used to load your application on the connected Primer.

Note: If you want to reprogram CircleOS on your Primer, you can use the cortex_pgm.exe utility from the “C:\Program Files\Raisonance\Ride\Lib\ARM\CircleOS” directory.
This will erase all the flash memory on your Primer.

Sharing your applications

Once your application is finished, you can share it with the STM32-Primer user community. Just go to the http://www.stm32circle.com Web site and create your free account. It will then be possible to upload your application so that other Primer users can benefit from it.

8. Conclusion

This tutorial was designed to introduce you to the basics of CircleOS and how to write applications on the STM32-Primer2.

Further information and detailed documentation for the Primer and CircleOS can be found at **http://www.stm32circle.com**. Raisonance will be pleased to see you joining the STM32-Primer community and share your experience and applications through the Web site.

History

Date	Modification
January 2009	Initial release, based on the application note AN0053 which was intended for STM32-Primer (not Primer2). Added a section describing how to register the compiled application on CircleOS. Added a section describing how to publish the application on http://www.stm32circle.com.

Disclaimer