MicroDAQ QUICK START GUIDE

1. Powering MicroDAQ

MicroDAQ can be powered from USB or external power supply with 5V only. For intensive computing or using external USB devices connected to MicroDAQ it is not recommended to use USB power supply. MicroDAQ in normal operation mode without external USB devices (WIFI dongle, USB flash drive) MicroDAQ drains 200-300mA.

In order to power MicroDAQ with USB use USB B type connector, connect your USB cable to power PC or active USB hub and then plug USB cable to MicroDAQ. In case when you use external power supply make sure that it is 5V power supply.

MicroDAQ has three LED which informs user about device state. Led above power supply connector informs that MicroDAQ is powered with USB or external power supply. During power up LED D1 and D2 informs about boot procedure. After 500ms D1 LED should be on that informs that MicroDAQ bootloader firmware starts. Led D2 informs that microSD card was detected and Linux kernel image is loaded into RAM memory. Then after 500ms D1 and D2 should be off this indicates that Linux kernel initialization is done.

2. MicroDAQ Linux

To increase connectivity and flexibility MicroDAQ uses Linux operating system. User can use different services provided by Linux community as well as develop its on applications. MicroDAQ is shipped with dedicated MLink software which gives possibility to connect and exchange data between device and PC computer.

2.1. Connectivity

User can connect to MicroDAQ by using three different ways. Ethernet, USB and WIFI connections are available. The default and recommended is Ethernet.

Ethernet

User can use Ethernet to connect to MicroDAQ device. In order to connect via Ethernet user has to connect PC and MicroDAQ with Ethernet. Leds on MicroDAQ Ethernet connector informs about link state.

USB

MicroDAQ has USB ports, USB2.0 port which is configured as a device, and USB1.1 port configured as a host. Default function on USB2.0 port is mass storage device.
User can use USB to connect to MicroDAQ device. In this mode MicroDAQ uses IP protocol on top of physical USB layer (RNDIS protocol).

WIFI

WIFI can be used to communicate with MicroDAQ device. By using USB WIFI dongle user can connect to MicroDAQ wireless. After attaching USB WIFI device MicroDAQ will be visible as an wireless access point.

Default IP settings

MicroDAQ device by default uses static IP addresses for network interfaces.
Ethernet – IP: 10.10.1.1, netmask: 255.255.255.0
WIFI – IP: 10.10.2.1, netmask: 255.255.255.0

In order to connect to MicroDAQ device user has to set up PC network interface with proper IP settings eg. Ethernet interface IP address can be 10.10.1.10, netmask 255.255.255.0.

In order to change default MicroDAQ IP settings use Mlink-web interface (not jet available).

2.2. Services

User can use different services to control MicroDAQ device.

Mlink

Mlink software allows data exchange with PC. User can use Mlink with Ethernet, USB or WIFI connection. Mlink simplifies using MicroDAQ as a control measurement system by providing features like controlling DSP applications from PC.

SHH

User can log in to MicroDAQ Linux using SSH protocol on port 22.
Accounts:
user: root password: root
user: microdaq password: microdaq

3. MicroDAQ DSP and Code Composer Studio

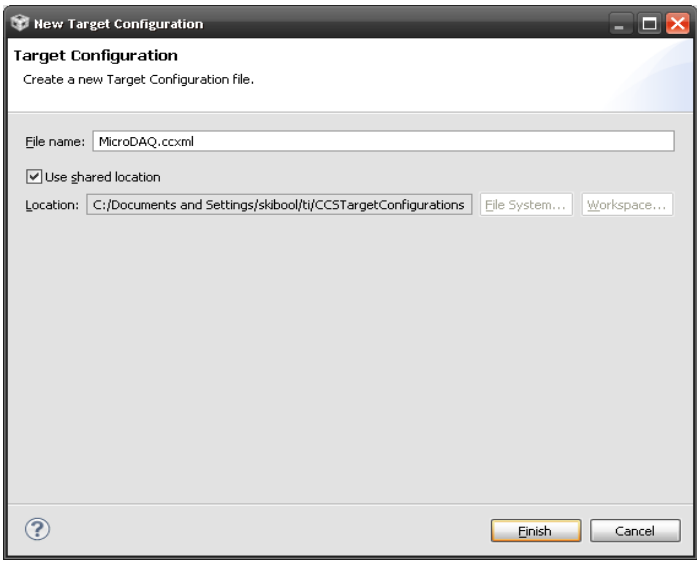
MicroDAQ is equated with C674x Fixed/Floating-Point VLIW DSP running with 375MHz max. TI Code Composer Studio is a native environment to develop software for TI processors.

3.1. Creating Target Configuration for MicroDAQ device.

In order to begin work with MicroDAQ and DSP processor a TI Code Composer Studio 5.3 or newer with C6000 family, SYS/BIOS and XDS100 emulator support has to be installed. After lunching CCS (Code Composer Studio) a Target Configuration has to be created. Target Configuration defines JTAG connection and processor type. In order to create and verify new target configuration MicroDAQ has to be on and JTAG XDS100v2 needs to be connected to JTAG connector on MicroDAQ and connected to PC with USB cable.

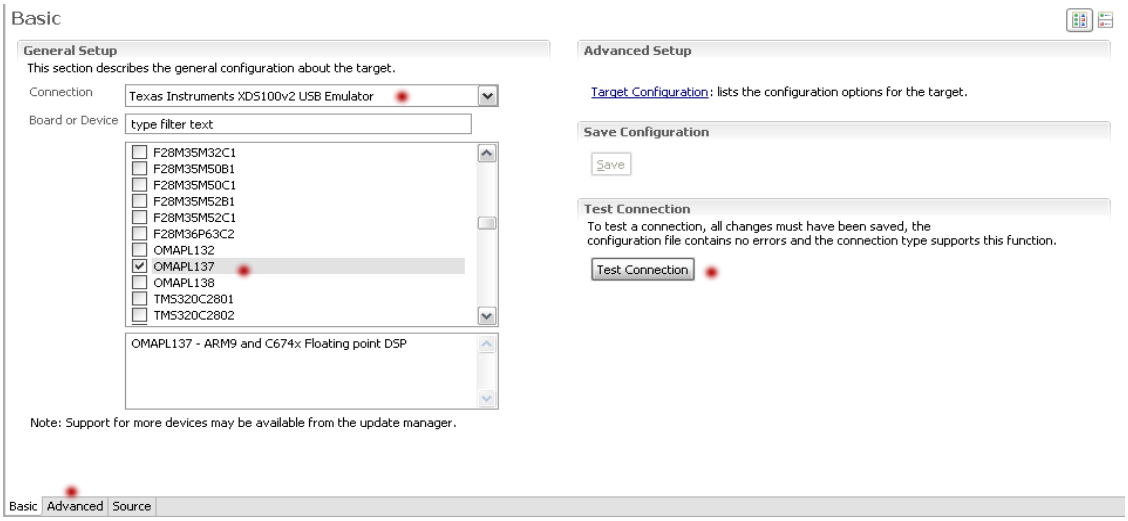
In order to create new target configuration user has to select:

File->New->Target Configuration File



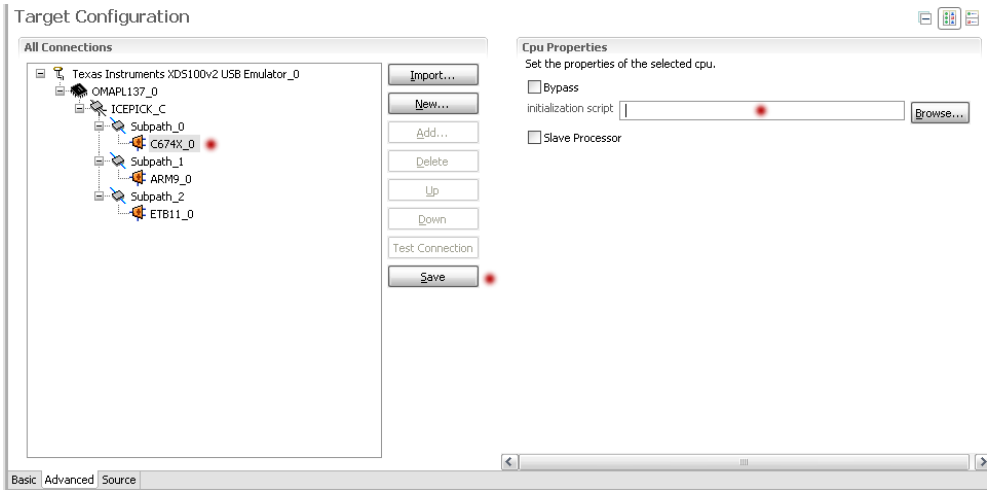
Picture 1: Target configuration wizard

New target configuration file name is MicroDAQ.ccxml and confirm with Finish.



Picture 2: Target configuration wizard – basic settings

User has to select Connection type (JTAG type) Texas Instruments XDS100v2 USB emulator and OMAPL137 from Device list. After saving selected configuration connection test can be done with Test Connection button. If test passed we can switch to Advanced tab at the bottom of the window.



Picture 3: Target configuration wizard - advanced settings

In Advanced settings from the All Connections list DSP (DSP674X_0) core has to be selected and initialization script should be removed. Initialization script performs processor initialization but in case of MicroDAQ initialization is done by bootloader so there is no need to use this script. After removing script path configuration has to be saved with Save button.

At this point new target configuration is ready and we can start work with CCS and DSP core.

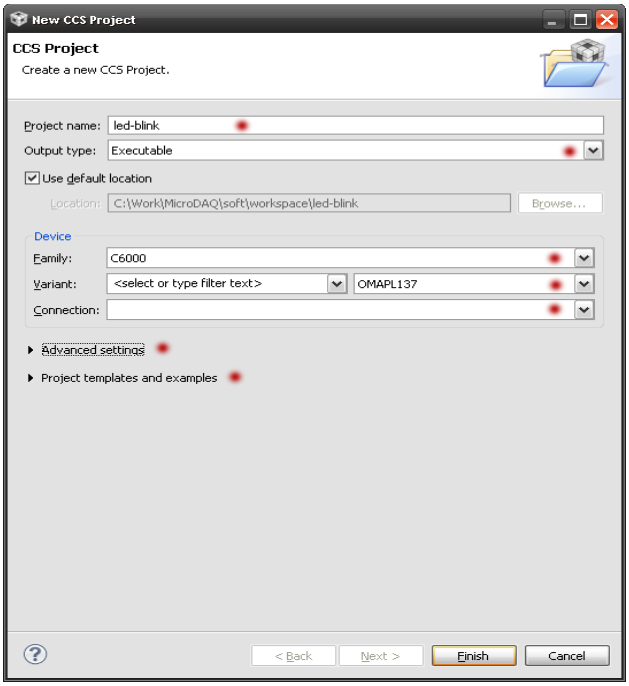
3.2. Led blink DSP application

Code Compose Studio allows user to create two kinds of projects – with SYS/BIOS operating system and non-os application.

SYS/BIOS™ 6.x is an advanced, real-time kernel for use in a wide range of DSPs, ARMs, and microcontrollers. It is designed for use in embedded applications that need real-time scheduling, synchronization, and instrumentation. It provides preemptive multitasking, hardware abstraction, and memory management. Compared to its predecessor, DSP/BIOS™ 5.x, it has numerous enhancements in functionality and performance.

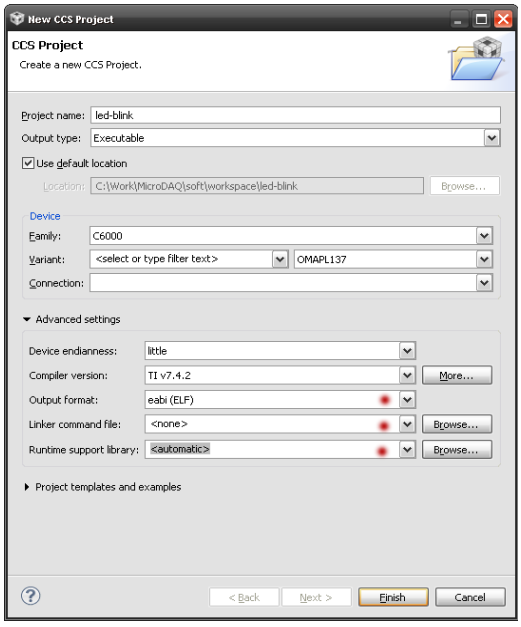
In order to create new CCS project with SYS/BIOS os select:

File->New->CCS Project



Picture 4: New project wizard - basic settings

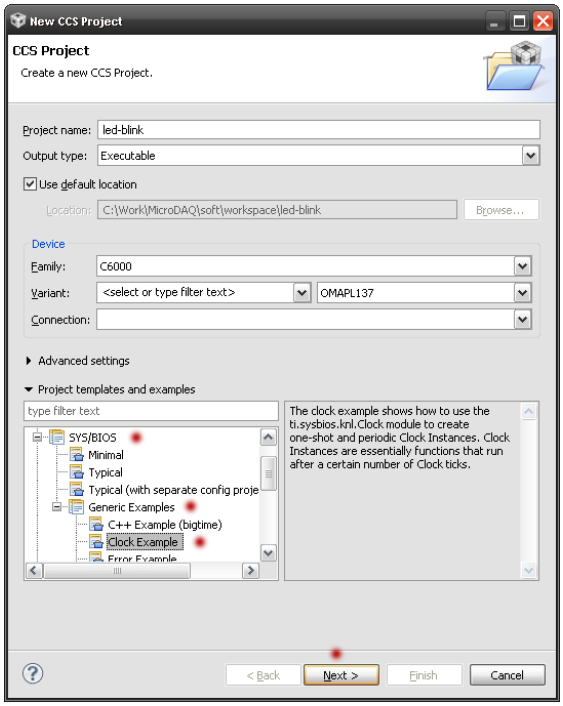
In the new project wizard window user sets basic settings for project – project name (eg. Led-blink), output type – executable. In Device section user has to select family of device – for MicroDAQ it is C6000 and OMAPL137 from Variant list. Connection has to be empty – we will use created target configuration for connection.



Picture 5: New project wizard - advanced settings

In Advanced settings Output format should be eabi(ELF) and Linker command file should be set to <none> because SYS/BIOS will generate linker file according to platform settings. Mlink supports only DSP applications in ELF format.

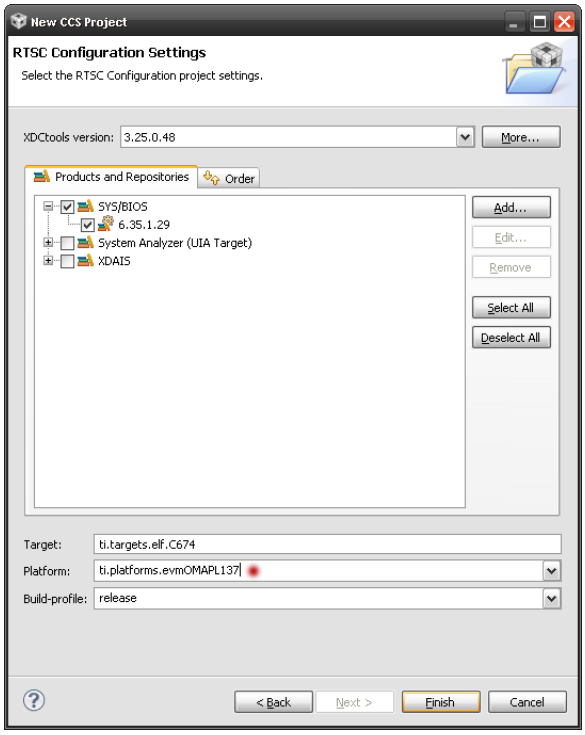
Finally project type has to be selected in Project templates and examples in New CCS Project window.



Picture 6: New project wizard - project template selection

From Project templates and examples list SYS/BIOS->Generic Examples-> Clock Example should be selected and confirmed by pressing Next.

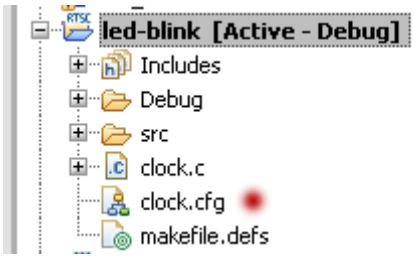
In the next window user defines RTSC platform which defines configuration for SYS/BIOS real-time kernel – memory sections for linker file generation, hardware resources etc.



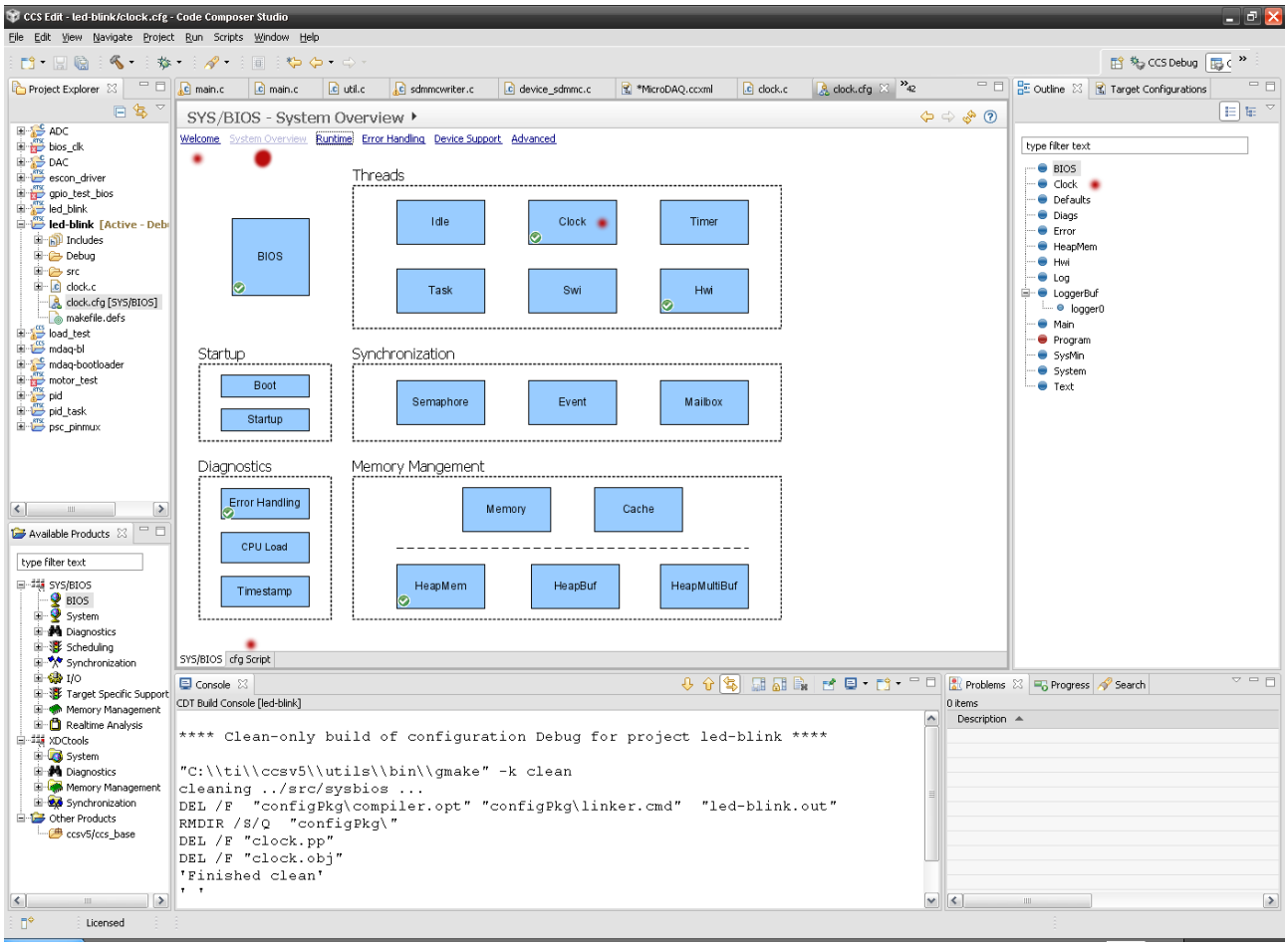
Picture 7: RTSC selection

From Platform list ti.platforms.evmOMAPL137 should be selected and confirmed by Finish.

In the Project Explorer list led-blink should appears.

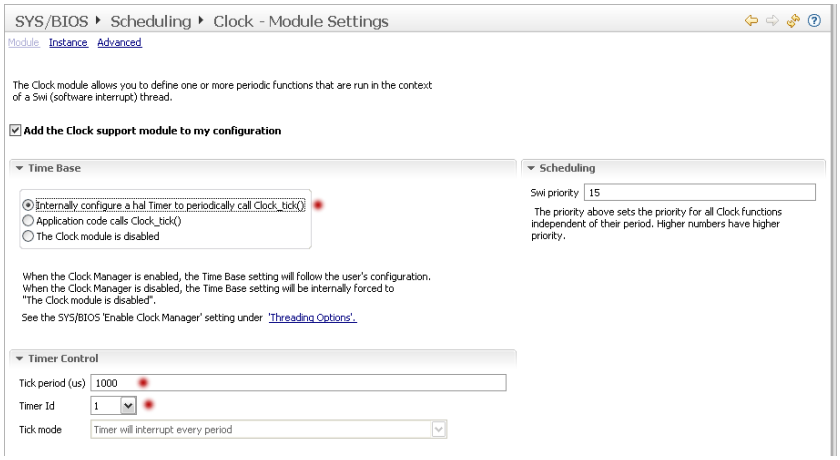


Project contains example with SYS/BIOS clock module. Clock.cfg file contains SYS/BIOS configuration, by clicking twice on this file we can see default system configuration for this example.



Picture 8: SYS/BIOS configuration

To see current system configuration System Overview needs to be selected from main window. In this example Clock and Hwi modules are enabled. Each module has its own settings. In the Clock module configuration user can define system tick and which hardware timer is used to generate interrupts. By clicking twice on Clock module in Thread section or selecting Clock from Outline module configuration is shown.



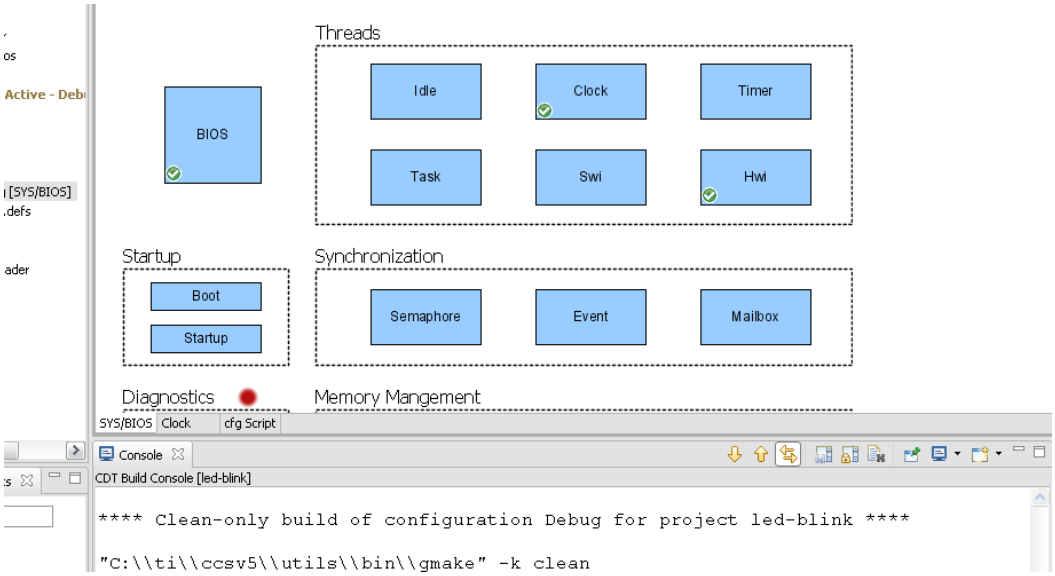
Picture 9: Clock module configuration

In configuration window user can define system clock behavior, tick period (by default is set to 1ms). User has to select Timer ID which will generate system ticks. In system where Linux runs in parallel with DSP core user has to select timer 1, timer 0 is used by Linux OS to generation time events.

SYS/BIOS configuration is possible not only from Code Composer studio, user can dynamically change system tick period with function tickReconfig():

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sysbios/6_21_00_13/exports/docs/docs/cdoc/ti/sysbios/knl/Clock.html#tick.Reconfig

User can modify SYS/BIOS configuration by editing config file in text editor.



Picture 10: Switching to text editor

By selecting tab cfg Script form bottom of the main window user can modify SYS/BIOS settings in text editor.

In order to use Mlink DSP loader feature entry point of DSP application has to be aligned with 0x800, that is why an default config has to be modified by adding code which instruct linker to place entry point in the certain aligned memory location:

```
Program.sectMap[".text:_c_int00"] = new Program.SectionSpec();
Program.sectMap[".text:_c_int00"].loadAddress = 0x80012000;
```

This code needs to be placed at the end of config file. After this modification DSP application is ready do compile.

Test application should use GPIO ports to blink user LED. In order to do that gpio library needs to be added to the project (gpio_lib.zip).

After modification and adding gpio lib our blink led application:

```
#include <xdc/std.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>

#include "gpio.h"

Void clk0Fxn(UArg arg0);
Void clk1Fxn(UArg arg0);

Void main()
{
    Clock_Handle clk2;
    Clock_Params clkParams;

    /* Create a periodic Clock Instance with period = 100 system time units */
    Clock_Params_init(&clkParams);
    clkParams.period = 100;
    clkParams.startFlag = TRUE;
    Clock_create(clk0Fxn, 10, &clkParams, NULL);

    /* Create an one-shot Clock Instance with timeout = 200 system time units */
    clkParams.period = 200;
    clkParams.startFlag = TRUE;
    clk2 = Clock_create(clk1Fxn, 11, &clkParams, NULL);

    Clock_start(clk2);

    BIOS_start();
}

Void clk0Fxn(UArg arg0)
{
        static UInt32 count;

    if ( count % 2)
        GPIO_setOutput(GP2_6, GPIO_LOW);
    else
        GPIO_setOutput(GP2_6, GPIO_HIGH);

    count++;

}

Void clk1Fxn(UArg arg0)
{

    return;
}
```

Application changes LED state by calling `GPIO_setOutput()`.Functions **`clk0Fxn() and clk1Fxn()`** are called with the number of periods specified in clkParams struct. After compilation an application can be run by pressing F11.