Getting Started with SDP-B using GCC Toolchain. Version 1.0.

Prasanth Rajagopal

Analog Devices India Product Development centre.

Getting Started with SDP-B using GCC toolchain:

The SDP (System Demonstration Platform) is a collection of controller boards, interposer boards, and daughter boards used for easy, low cost evaluation of ADI components and reference circuits. The controller boards provide a means of communicating to the PC from the system being evaluated or demonstrated. The SDP-B is a Blackfin, ADSP-BF527, based controller board. The SDP-B has on board JTAG, SDRAM, Flash memory and an LED making it an attractive package for those looking at low cost development platforms on Blackfin. This, combined with the open source GNU tool-chain and a low-cost emulator, reduces the cost factor significantly. SDP-B is available at: USD 99 [1] and the Low-cost emulator ICE-100B is available at: USD 150 [2], so one could kick-off 600Mhz Blackfin based development in just USD 250 [1] [2]. User can also opt between Windows or Linux host development platforms.

This article is a quick start guide for those who wants to evaluate SDP-B in a GCC environment. SDP-B Rev-B & Windows host is used for testing purpose. Refer to the reference section for various related links.

The SDP-B board features:

• Analog Devices ADSP-BF527 Blackfin processor

• Core performance up to 600 MHz

• 208-ball CSP-BGA package

• 24 MHz CLKIN oscillator

• 5 Mb of internal RAM memory

• 32 Mb flash memory
    Numonyx M29W320EB (Rev B) or
    Numonyx M25P32 (Rev 1.3)

• SDRAM memory
    Micron MT48LC16M16A2P-6A - 16 Mb x 16 bits (256 Mb/32 MB)

• 3 × 120-pin small foot print connectors
    Hirose FX8-120P-SV1(91), 120-pin header

• JTAG header footprint

• USB 2.0 device interface to host PC.

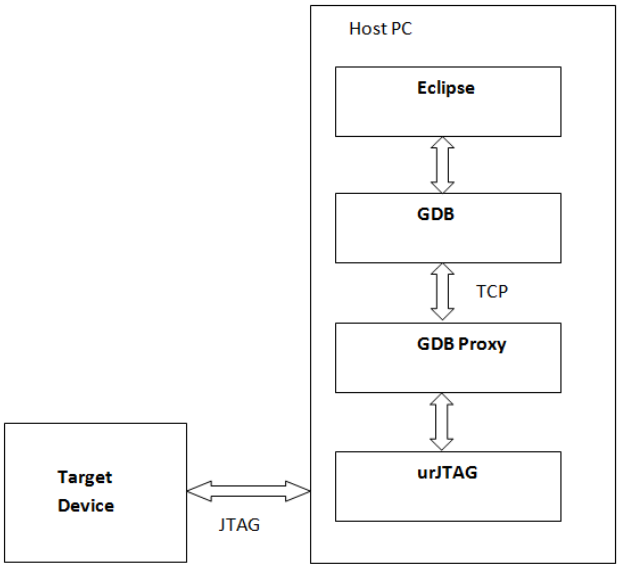
• Blackfin processor peripherals exposed
    SPI
    SPORT
    TWI/I2C
    GPIO
    PPI
    Asynchronous parallel
    Timers

Development System:

SDP-B is a USB powered board [3], so one needs to have a USB mini-cable to power on the system. Typical development environment consists of a JTAG-based emulator and a GUI-based Debugger. ICE-100B is a low cost emulator from ADI that has direct support for the GCC toolchain. Another USB-cable is required to hook up the emulator to the JTAG connector in SDP-B board. Eclipse is an open source IDDE, which presents a complete debugger front-end to the developer. It can support almost any debugging requirement such as build code, execute, stop-at-breakpoints, single-step through code, view memory, MMRs etc.

GNU Debugger:

GDB (GNU Debugger) is the debugger utility for debugging applications built using GCC toolchain. User should be aware of the basic infrastructure that enables Eclipse to debug applications via gdb. A typical setup is given below.



Eclipse needs to use gdbproxy in order to talk to the target device. Gdbproxy, as the name indicates, is a proxy server or a stub which in turn uses urJTAG software for the actual JTAG operations.

Installation of bare metal toolchain and Eclipse:

The latest release of the toolchain and the utilities (blackfin-toolchain-win32-) can be downloaded from the following link:

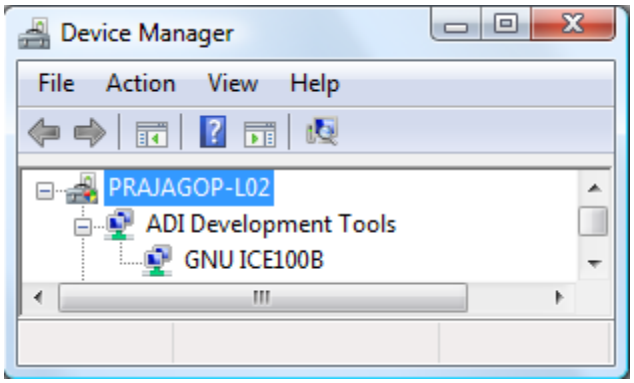
http://blackfin.uclinux.org/gf/project/toolchain/frs/

Once all files are installed, the emulator driver must be installed through *.inf file modifications. This file can be found in the toolchain installation path: "..\GNU Toolchain\2010R1\gnICE-drivers". Next, modify the *.inf to include support for ICE-100B, as given below:

```
;[Devices]
;%gnICE_JTAG%  = USB_Install, USB\VID_0456&PID_f000&MI_00
;%gnICE+_JTAG% = USB_Install, USB\VID_0456&PID_f001&MI_00
"GNU ICE100B" = USB_Install, USB\VID_064b&PID_0225&REV_0100
[Devices.NTX86]
%gnICE_JTAG%  = USB_Install, USB\VID_0456&PID_f000&MI_00
%gnICE+_JTAG% = USB_Install, USB\VID_0456&PID_f001&MI_00
"GNU ICE100B" = USB_Install, USB\VID_064b&PID_0225&REV_0100
[Devices.NTAMD64]
%gnICE_JTAG%  = USB_Install, USB\VID_0456&PID_f000&MI_00
%gnICE+_JTAG% = USB_Install, USB\VID_0456&PID_f001&MI_00
"GNU ICE100B" = USB_Install, USB\VID_064b&PID_0225&REV_0100
```

Connecting JTAG first time:

1. Connect emulator.
2. Connect USB cable from emulator to PC.
3. When PC asks for driver software, install the same from the path:
   "..\GNU Toolchain\2010R1\gnICE-drivers
4. Restart PC.
5. Check whether the device is listed in the device manager such as below.



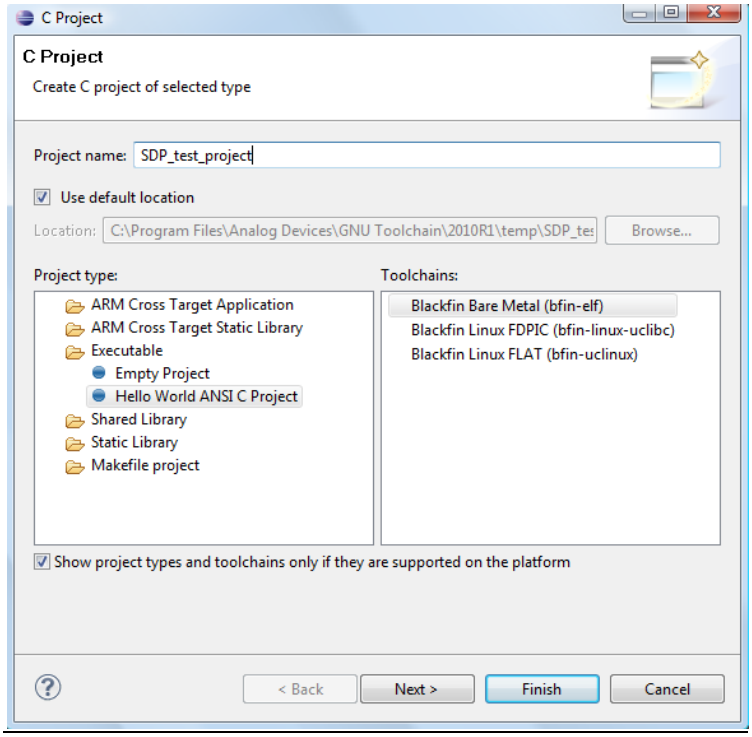
Running gdbproxy:

Before any actual program can be debugged, the gdbproxy must successfully detect the jtag device. Gdbproxy maybe untouched after the connection is established. Example log is given below, after running this command:

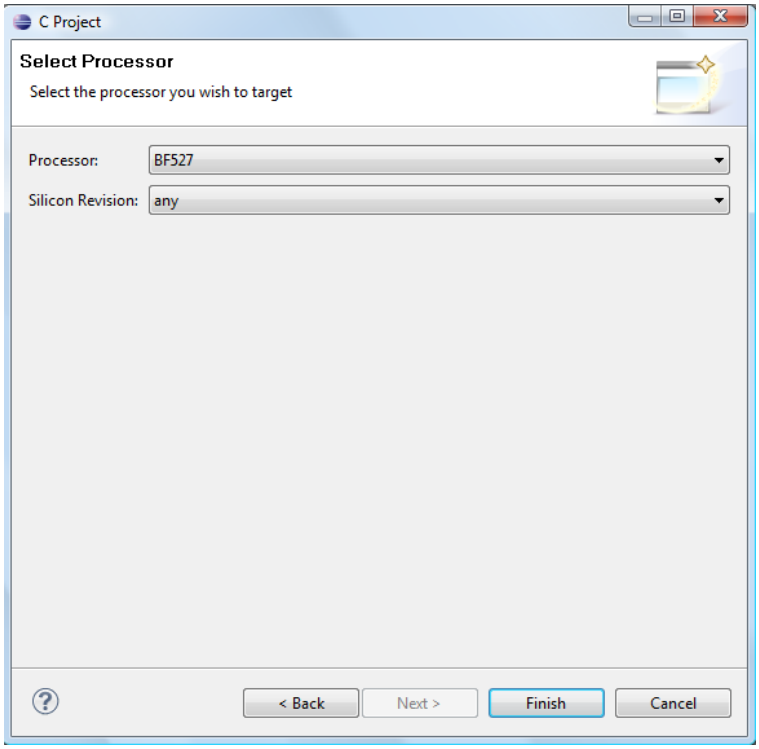
```
C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>bfin-gdbproxy.exe

Remote proxy for GDB, v0.7.2, Copyright (C) 1999 Quality Quorum Inc.
MSP430 adaption Copyright (C) 2002 Chris Liechti and Steve Underwood
Blackfin adaption Copyright (C) 2008 Analog Devices, Inc.

GDBproxy comes with ABSOLUTELY NO WARRANTY; for details use `--warranty' option. This is Open
Source software. You are welcome to redistribute it under certain conditions. Use the '--copying'
option for details.

Found USB cable: ICE-100B
ICE-100B firmware version is 2.0.6
IR length: 5
Chain length: 1
Device Id: 00100010011111100000000011001011 (0x227E00CB)
  Manufacturer: Analog Devices, Inc. (0x0CB)
  Part(0):      BF527 (0x27E0)
  Stepping:     2
  Filename:     c:\program files\analog devices\gnu toolchain\svn-20101128\elf\b
in\../share/urjtag/analog/bf527/bf527
warning:   bfin: no board selected, BF527 is detected
notice:    bfin: jc: waiting on TCP port 2001
notice:    bfin: jc:  (you must connect GDB before using jtag console)
notice:    bfin-gdbproxy.exe: waiting on TCP port 2000
```

Creating project in Eclipse (only sample screenshots and explanations):

Start with new C project and appropriate toolchain:



Click on next until you arrive at processor definition:



A new project is now created with default file template. Remember that there are no low level drivers for stdio functions; you need to create your own implementation with newlib library for this. So delete the following code for now:

```
#include <stdio.h>

puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
```

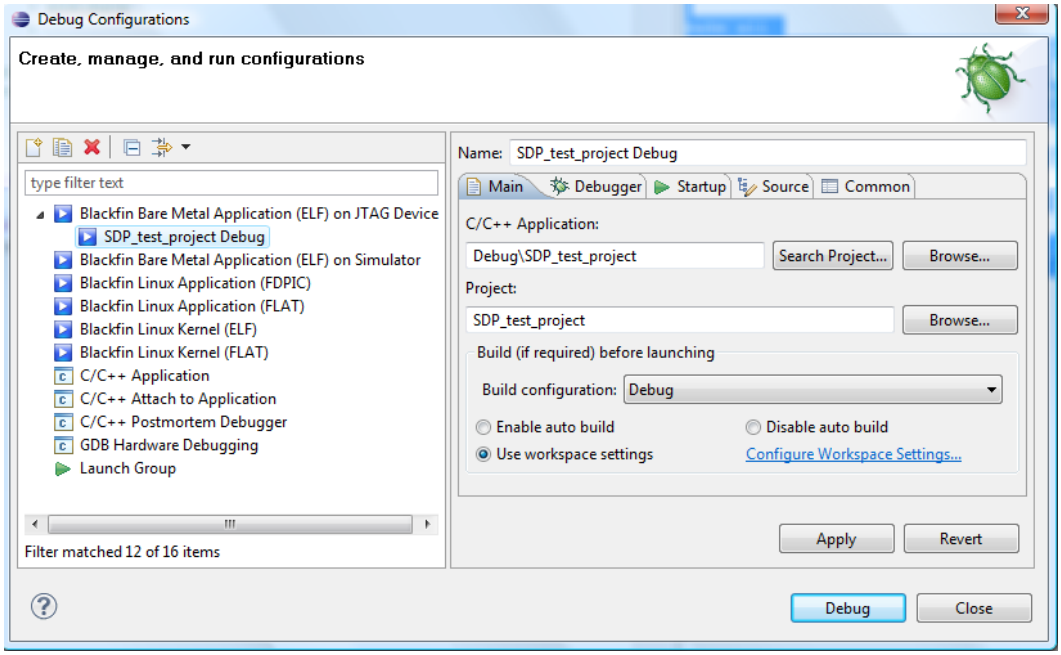
Right click on the project tab and build the code. Eclipse throws out verbose like below:

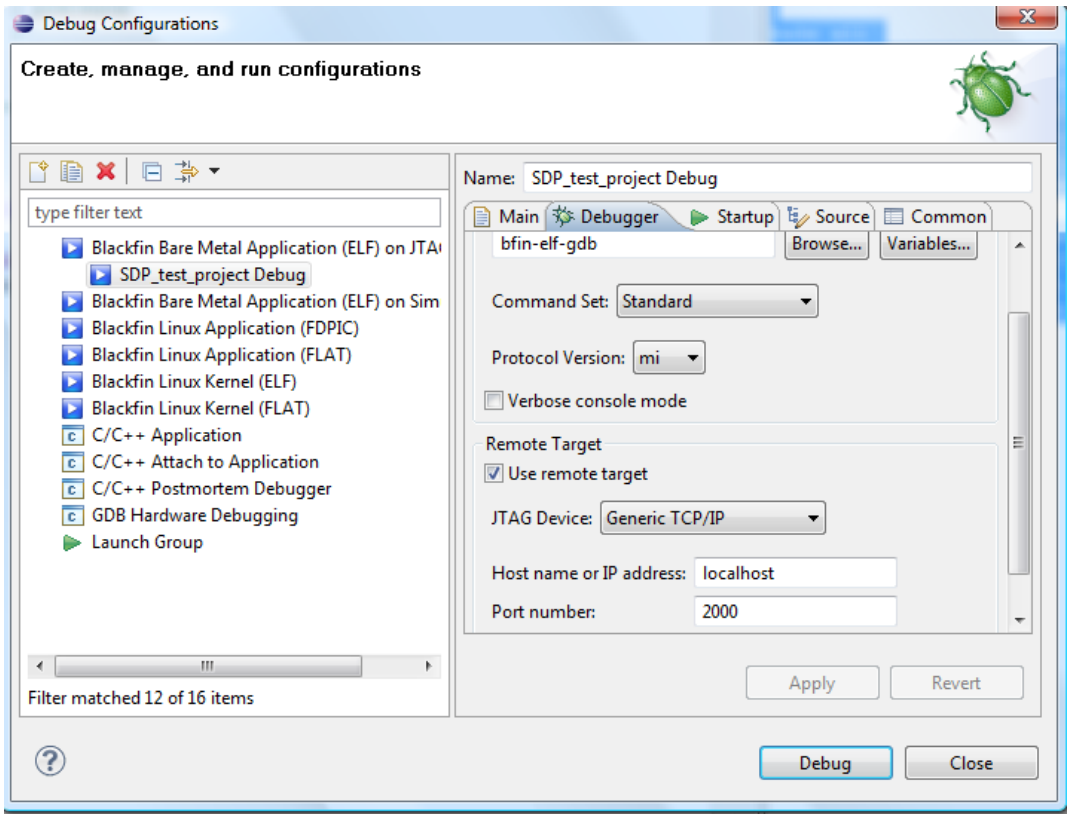
```
**** Build of configuration Debug for project SDP_test_project ****

make all
'Building file: ../src/SDP_test_project.c'
'Invoking: Blackfin ELF C Compiler'
bfin-elf-gcc -O0 -g3 -Wall -c -fmessage-length=0 -mcpu=bf527-any -MMD -
MP -MF"src/SDP_test_project.d" -MT"src/SDP_test_project.d" -
o"src/SDP_test_project.o" "../src/SDP_test_project.c"
'Finished building: ../src/SDP_test_project.c'
' '
'Building target: SDP_test_project'
'Invoking: Blackfin ELF C Linker'
bfin-elf-gcc -mcpu=bf527-any -
o"SDP_test_project"  ./src/SDP_test_project.o
'Finished building target: SDP_test_project'
' '
```

If the status is reporting as finished building, we have an executable ready for testing.

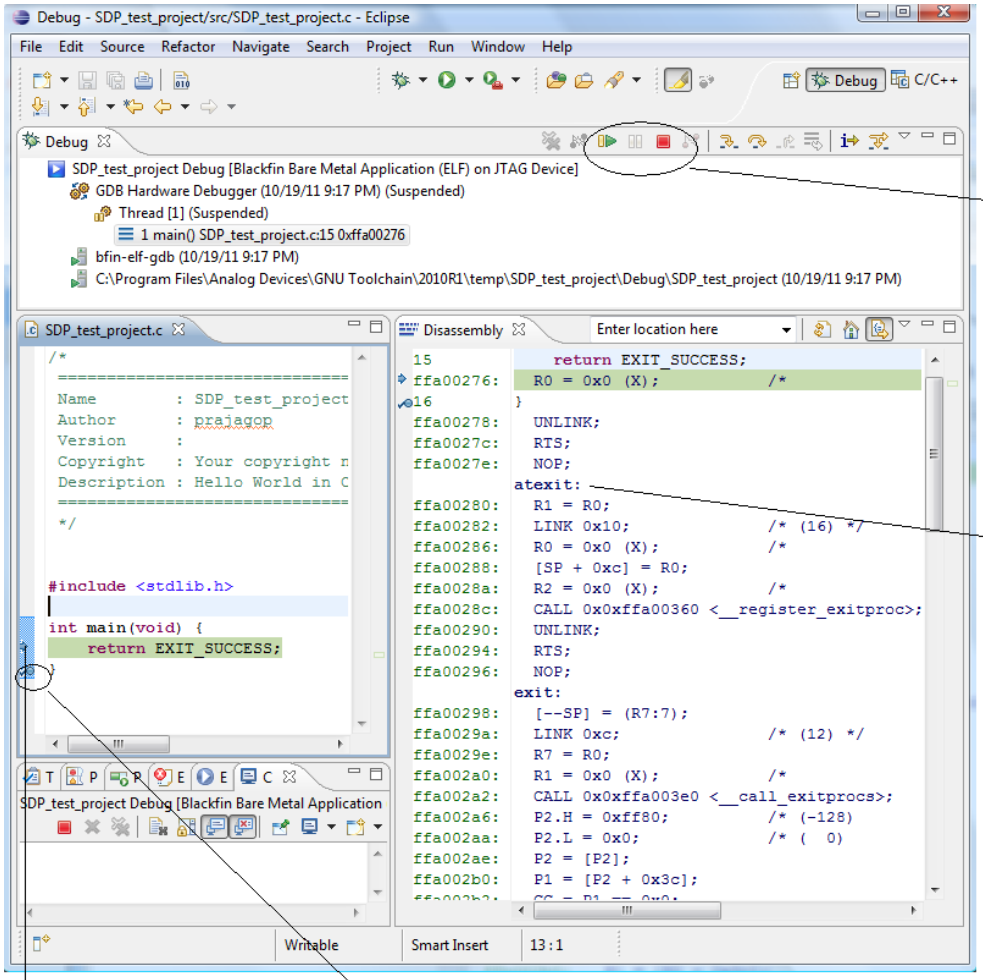
Debugging with Eclipse:

Next step is to create a debug configuration for the project. Right click on project and choose Debug Configurations. Create a new configuration under “bare metal application on JTAG device”. Go to Debugger tab and change the port number to 2000. Click on “apply” and “debug”. Sample screenshots are given below.

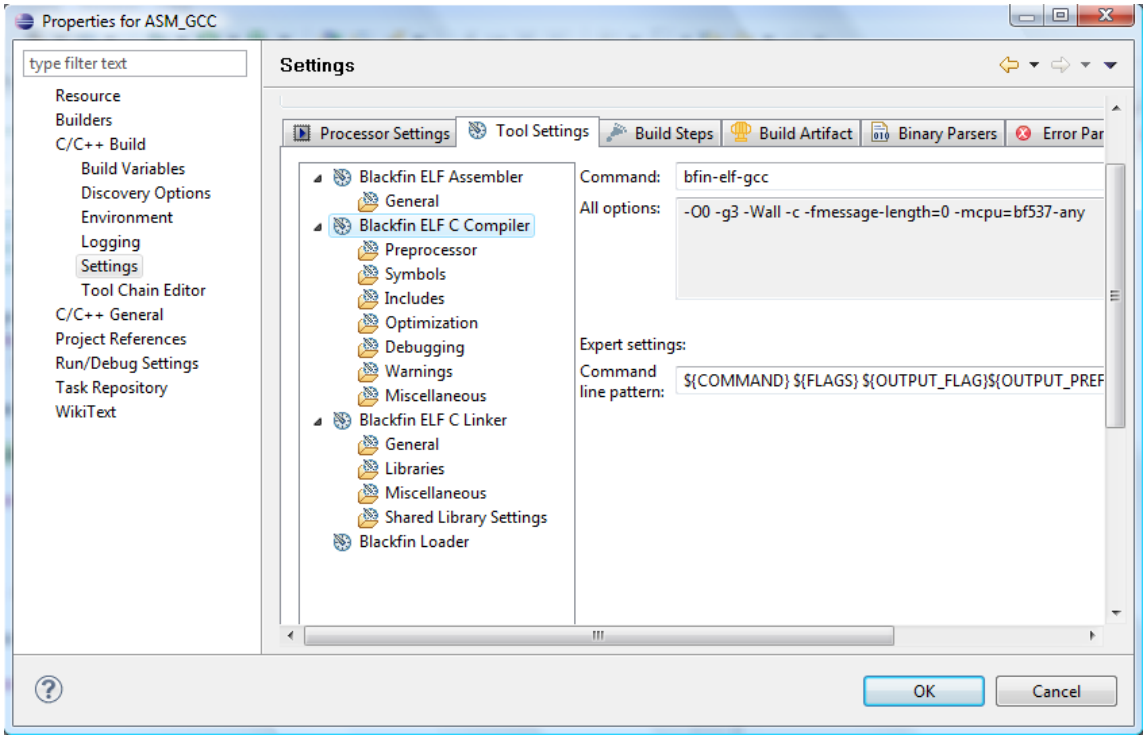

Debug Configurations
Create, manage, and run configurations
type filter text
Blackfin Bare Metal Application (ELF) on JTAG Device
SDP_test_project Debug
Blackfin Bare Metal Application (ELF) on Simulator
Blackfin Linux Application (FDPIC)
Blackfin Linux Application (FLAT)
Blackfin Linux Kernel (ELF)
Blackfin Linux Kernel (FLAT)
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
GDB Hardware Debugging
Launch Group
Filter matched 12 of 16 items
Name: SDP_test_project Debug
Main
Debugger
Startup
Source
Common
C/C++ Application:
Debug\SDP_test_project
Search Project...
Browse...
Project:
SDP_test_project
Browse...
Build (if required) before launching
Build configuration: Debug
Enable auto build
Disable auto build
Use workspace settings
Configure Workspace Settings...
Apply
Revert
Debug
Close

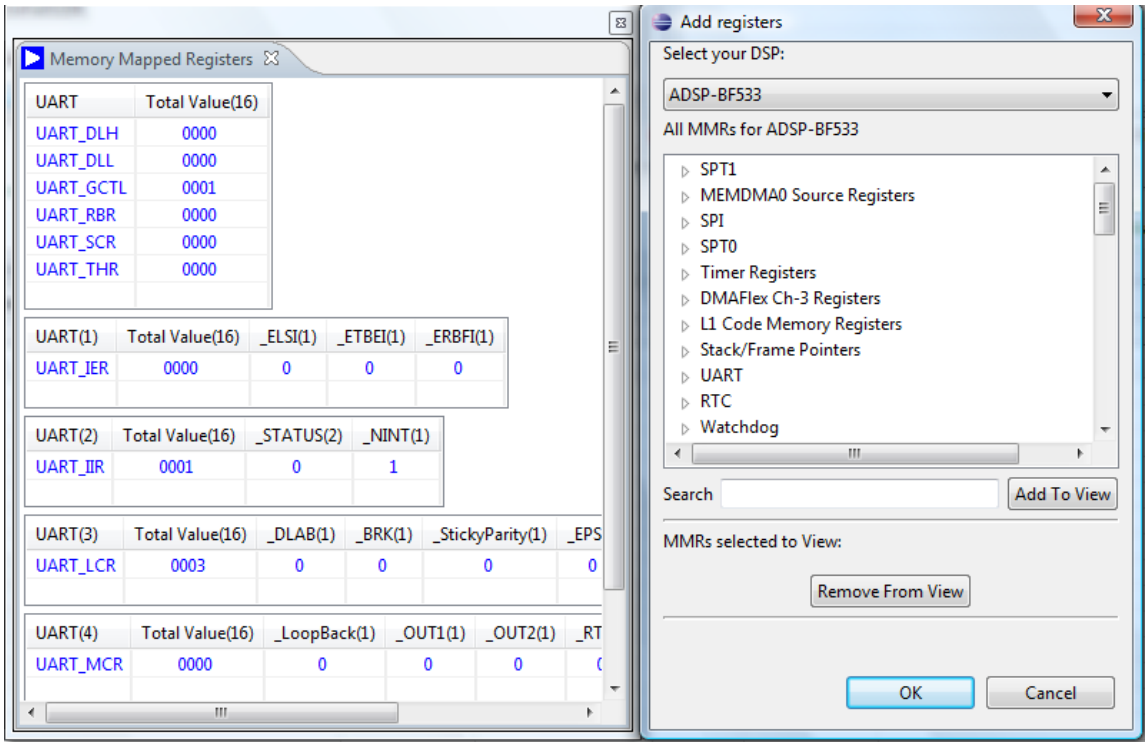

Debug Configurations
Create, manage, and run configurations
type filter text
Blackfin Bare Metal Application (ELF) on JTA
SDP_test_project Debug
Blackfin Bare Metal Application (ELF) on Sim
Blackfin Linux Application (FDPIC)
Blackfin Linux Application (FLAT)
Blackfin Linux Kernel (ELF)
Blackfin Linux Kernel (FLAT)
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
GDB Hardware Debugging
Launch Group
Filter matched 12 of 16 items
Name: SDP_test_project Debug
Main
Debugger
Startup
Source
Common
bfin-elf-gdb
Browse...
Variables...
Command Set: Standard
Protocol Version: mi
Verbose console mode
Remote Target
Use remote target
JTAG Device: Generic TCP/IP
Host name or IP address: localhost
Port number: 2000
Apply
Revert
Debug
Close

A screenshot of successful debug will look like below. Terminate with red-button and re-launch with Debug tab right click for every new build.



Explore the toolchain settings for project and the MMR window:





Minimal test of SDP-B hardware with GCC:

The following code blinks LED1 in the SDP:

```
#include <cdefBF527.h>

void writeToSdpLatch(unsigned char
latchData);
void flashLed(void);
void waitMilliSec(void);
void configEBIU(void);
void configFlashEnable(void);

#define PIN_PG0 0x0001
unsigned char latchValue = 0;


int main()
{

	configFlashEnable();
	configEBIU();

	while(1)
	{
		flashLed();

	}

	return 0;

}


void configFlashEnable()
{
	*pPORTGIO_DIR |= PIN_PG0;
	*pPORTGIO_SET = PIN_PG0;
	asm("ssync;");

	latchValue = 0;

}


void configEBIU()
{
	*pEBIU_AMGCTL  = 0x00F9;
	*pEBIU_AMBCTL0 = 0x7BB07BB0;
	*pEBIU_AMBCTL1 = 0x7BB07BB0;
	asm("ssync;");

	latchValue = 0;

}
```

```
void writeToSdpLatch(unsigned
char latchData)
{

// write to latch (bank 2)
*((unsigned char*)0x20200000) =
latchData;
asm("ssync;");

}


void flashLed(void)
{
unsigned char i = 0;

for (i=0; i<5; i++)
{
latchValue = latchValue | 0x08;
writeToSdpLatch(latchValue);
waitMilliSec();
latchValue = latchValue & 0xF7;
writeToSdpLatch(latchValue);
waitMilliSec();
}
}


void waitMilliSec()
{
	int j = 0;

	for(j=0;j<1000000;j++)
	asm("NOP;");
}
```

Creating LDR file:

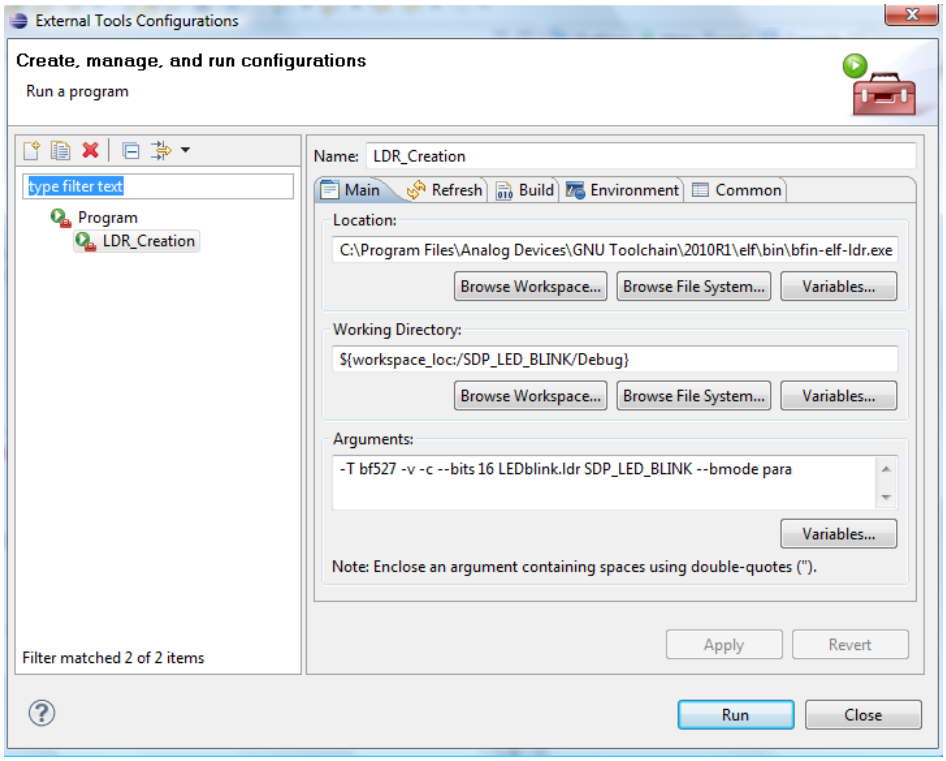
Once the project has been debugged via emulator, one needs to create the corresponding LDR (Loader) file in order to use in a standalone system. A standalone system consists of a boot source, typically a non-volatile memory from where the LDR file is read by the on-chip BOOT-ROM in Blackfin. The BOOT-ROM copies the various code and data objects in to various memories available in the system. Once BOOT-ROM has finished this operation, it jumps to the start of application to perform further execution.

In GCC environment, this LDR file is created via ld-utils provided in "\GNU Toolchain\2010R1\elf\bin" folder in the installation directory. Command bfin-elf-ldr.exe along with its parameters needs to be run in order to burn the ldr in the correct way to the flash. For example, the following command operates on the executable named SDP_LED_BLINK and creates an LDR file called LEDblink.ldr for BF527 supporting a 16-bit Flash device.

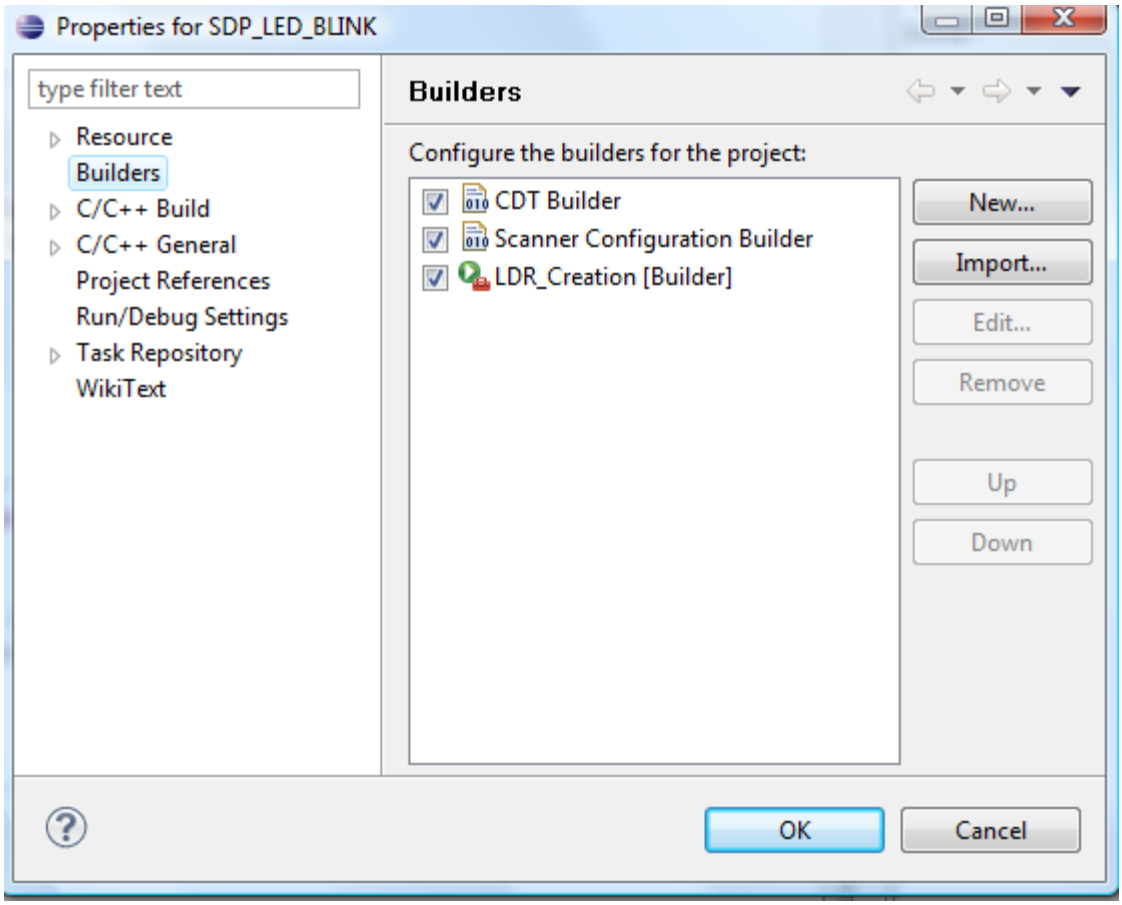
```
C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>bfin-elf-ldr.exe -T
 bf527 -v -c --bits 16 LEDblink.ldr SDP_LED_BLINK --bmode para
Creating LDR LEDblink.ldr ...
 Adding DXE 'SDP_LED_BLINK' ... [jump block to 0xFFA00000] [ELF block: 1104 @ 0x
FF800000] [ELF block: 1620 @ 0xFFA00000] OK!
Done!
```

Automating LDR creation with Eclipse

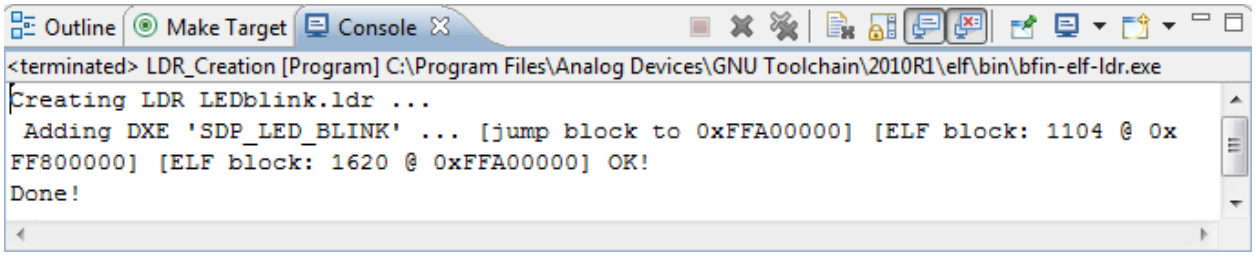
Create a new configuration under Run -> External Configuration, and appropriate parameters.



Add this tool to the Builder list so that Eclipse calls this automatically.



Upon successful build, eclipse throws the output as below:



Loading LDR file via urJTAG software:

Perform the following command sequence with urJTAG command prompt in order to load an ldr file. The tool supports auto-completion pre-load from history. Example sequence used for burning the LED_blink_ ldr:

```
C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>bfin-jtag.exe

UrJTAG 0.10 #5733
Copyright (C) 2002, 2003 ETC s.r.o.
Copyright (C) 2007, 2008, 2009 Kolja Waschk and the respective authors

UrJTAG is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
There is absolutely no warranty for UrJTAG.

warning: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.

jtag> cable ICE-100B
ICE-100B firmware version is 2.0.6
jtag> detect
IR length: 5
Chain length: 1
Device Id: 00100010011111100000000011001011 (0x227E00CB)
  Manufacturer: Analog Devices, Inc. (0x0CB)
  Part(0):      BF527 (0x27E0)
  Stepping:     2
  Filename:     c:\program files\analog devices\gnu toolchain\svn-20101128\elf\b
in\../share/urjtag/analog/bf527/bf527
jtag> initbus bf527_sdp
```

```
jtag> detectflash 0x20000000
Query identification string:
    Primary Algorithm Command Set and Control Interface ID Code: 0x0002 (AMD/Fujitsu Standard
Command Set)
    Alternate Algorithm Command Set and Control Interface ID Code: 0x0000 (null)
Query system interface information:
    Vcc Logic Supply Minimum Write/Erase or Write voltage: 2700 mV
    Vcc Logic Supply Maximum Write/Erase or Write voltage: 3600 mV
    Vpp [Programming] Supply Minimum Write/Erase voltage: 11500 mV
    Vpp [Programming] Supply Maximum Write/Erase voltage: 12500 mV
    Typical timeout per single byte/word program: 16 us
    Typical timeout for maximum-size multi-byte program: 0 us
    Typical timeout per individual block erase: 1024 ms
    Typical timeout for full chip erase: 0 ms
    Maximum timeout for byte/word program: 256 us
    Maximum timeout for multi-byte program: 0 us
    Maximum timeout per individual block erase: 8192 ms
    Maximum timeout for chip erase: 0 ms
Device geometry definition:
    Device Size: 4194304 B (4096 KiB, 4 MiB)
    Flash Device Interface Code description: 0x0002 (x8/x16)
    Maximum number of bytes in multi-byte program: 1
    Number of Erase Block Regions within device: 2
    Erase Block Region Information:
        Region 0:
            Erase Block Size: 8192 B (8 KiB)
            Number of Erase Blocks: 8
        Region 1:
            Erase Block Size: 65536 B (64 KiB)
            Number of Erase Blocks: 63
Primary Vendor-Specific Extended Query:
    Major version number: 1
    Minor version number: 1
    Address Sensitive Unlock: Required
    Erase Suspend: Read/write
    Sector Protect: 1 sectors per group
    Sector Temporary Unprotect: Not supported
    Sector Protect/Unprotect Scheme: 29BDS640 mode (Software Command Locking)
```

```
    Simultaneous Operation: Not supported
    Burst Mode Type: Supported
    Page Mode Type: Not supported
    ACC (Acceleration) Supply Minimum: 11500 mV
    ACC (Acceleration) Supply Maximum: 12500 mV
    Top/Bottom Sector Flag: Bottom boot device
jtag> endian little
jtag> flashmem 0x20000000 LEDblink.ldr
Chip: AMD Flash
    Manufacturer: ST/Samsung
    Chip: Unknown (ID 0x2257)
    Protected: 0000
program:
flash_unlock_block 0x20000000 IGNORE

block 0 unlocked
flash_erase_block 0x20000000
flash_erase_block 0x20000000 DONE
erasing block 0: 0
addr: 0x200005CC
verify:
addr: 0x200005CC
Done.
jtag> instruction BYPASS
jtag> shift ir
jtag> quit

C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>
```

Notes:

[1] – Refer to SDP web page in Analog Devices website for up to date information on pricing and availability.

http://www.analog.com/sdp

[2] - Refer to ICE 100B web page in Analog Devices website for up to date information on pricing and availability.

http://www.analog.com/en/processors-dsp/blackfin/emulator-100/processors/product.html

[3] - While the SDP can be powered over USB, when there are daughter boards attached the daughter boards should normally power the SDP. This is because of the variability of the USB power (+/-10% and current dependent on what other devices share the bus).

Reference:

SDP:	http://www.analog.com/sdp
SDP Wiki:	http://wiki.analog.com/resources/eval/sdp
Blackfin Processor:	http://www.analog.com/blackfin
Open Source documentation:	http://docs.blackfin.uclinux.org/
GCC bare metal documention:	http://docs.blackfin.uclinux.org/doku.php?id=toolchain:bare_metal
GCC Files and development:	http://blackfin.uclinux.org/gf/project/toolchain
Eclipse:	http://www.eclipse.org/

Discussion Forums:

SDP:	http://ez.analog.com/community/circuits_from_the_lab/sdp
Blackfin:	http://ez.analog.com/community/dsp/blackfin-processors
Open Source development:	http://ez.analog.com/community/dsp/open-source