User Guide

IDAP-M

CMSIS-DAP Debug JTAG Module

i-SYST INC

Limited Warranty

The IDAP-M board is warranted against defects in materials and workmanship for a period of 90 days from the date of purchase from I-SYST inc. or from an authorized dealer.

Disclaimer

I-SYST inc. reserves the right to change this product without prior notice. Information furnished by I-SYST inc. is believed to be accurate and reliable. However, no responsibility is assumed by I-SYST inc for its use; nor for any infringement of patents nor other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of I-SYST inc.

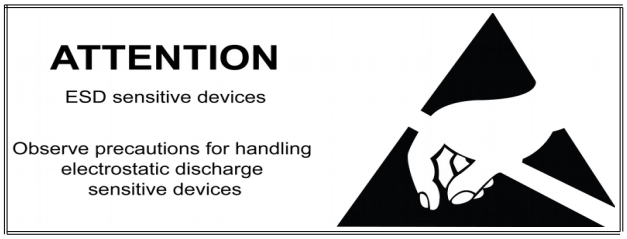


Table of Contents

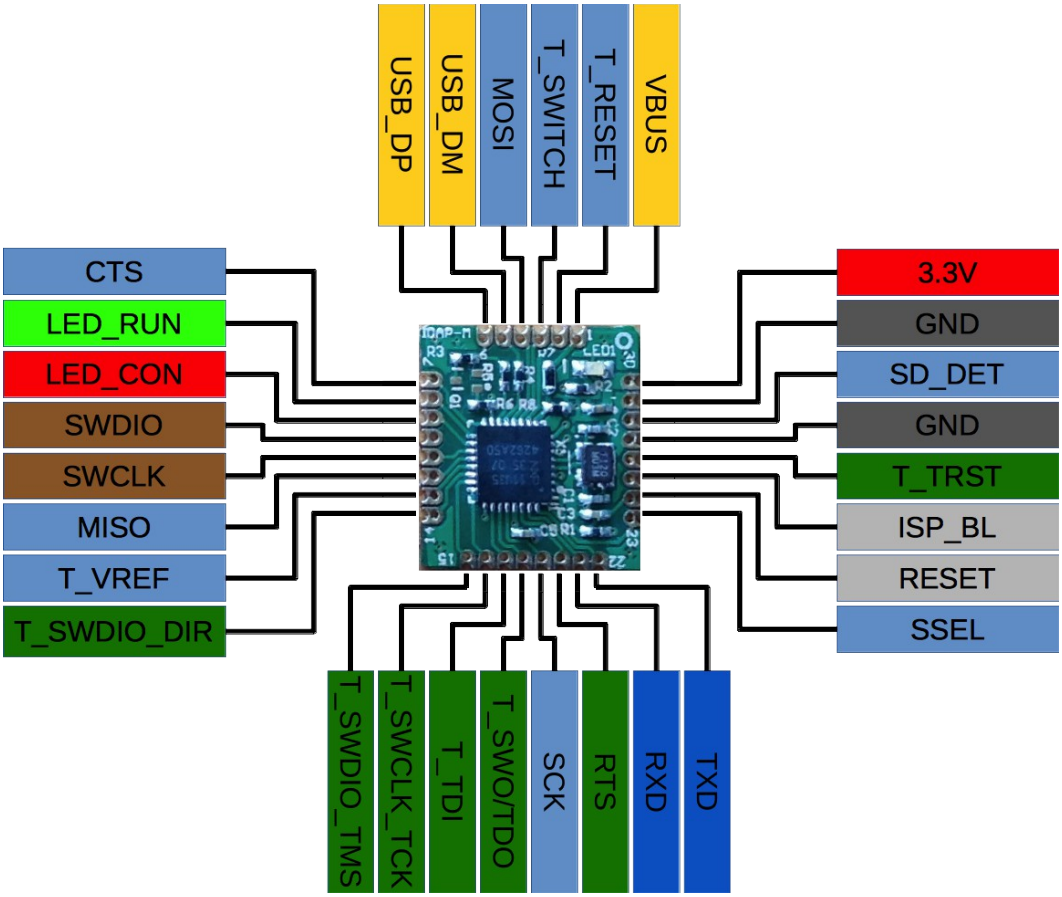
Introduction 1
Features : 1
Pin Out 2
Target MCU interfacing 3
Example Schematic 4
Switches 5
S1 – ISP boot/Program 5
S2 – IDAP-Link Reset 5
Windows CDC driver installation 5
IDAP-M Firmware Update 7
Eclipse Development Evironment 7
Creating custom target core support 9
Target Flash Programming 9

Introduction

The IDAP-M is a low cost CMSIS-DAP debug JTAG on module with enhanced features. Designed to be integrated onboard of target MCU. Beside from full JTAG/SWD debug, it is a USB composite device providing a UART to USB bridge for the target MCU, mass storage device to program target by drag & drop. These features can turn the target device into mBed enable. It can also be used as an ultra low cost solution to production programming by integrating multiple modules for parallel programming. Besides from our proprietary firmware, BSP is provided for use with mBed.org Open Source CMSIS-DAP firmware which make it totally customizable.

Features :

- Support both SWD & JTAG mode
- Debug compatibility with most IDE such as Keil, CrossWorks, Eclipse, etc..
- UART to USB bridge for communication between target and PC
- SPI interface for SD card
- USB mass storage for firmware Drag & Drop
- mBed enable
- Firmware flashing by drap & drop simply by copying file over
- BSP is provided for Open Source CMSIS-DAP firmware from mBed.org
- Dimention : 16 x 15 mm



Pin Out

1	VBUS	USB power input detection
2	T_RESET	Target hardware reset pin
3	T_SWITCH	Reserved for future use
4	MOSI	IDAP-M SPI MOSI
5	USB_DM	USB Data Minus
6	USB_DP	USB Data Plus
7	CTS	IDAP-M UART CTS
8	LED_RUN	LED run indicator
9	LED_CON	LED connection indicator
10	SWDIO	SWD interface for programming the IDAP-M module
11	SWCLK	SWD interface for programming the IDAP-M module
12	MISO	IDAP-M SPI interface MISO
13	T_VREF	Target voltage sensing
14	T_SWDIO_DIR	Target SWDIO pin direction control
15	T_SWDIO_TMS	Target SWDIO/TMS pin
16	T_SWCLK_TCK	Target SWCLK/TCK pin
17	T_TDI	Target TDI pin
18	T_SWO_TDO	Target SWO/TDO pin
19	SCK	IDAP-M SPI interface clock
20	RTS	IDAP-M UART RTS
21	RXD	IDAP-M UART RXD
22	TXD	IDAP-M UART TXT
23	SSEL	IDAP-M SPI interface select
24	RESET	IDAP-M reset pin
25	ISP_BL	IDAP-M ISP/PROG button input.
26	T_TRST	Target TRST pin
27	GND	Gound
28	SD_DET	SD Card detect
29	GND	Ground
30	VCC	+3.3 V power input

Target MCU interfacing

The diagrams bellow show how to put IDAP-M onboard of target MCU. There are 2 types of interfaces SWD mode and JTAG mode.

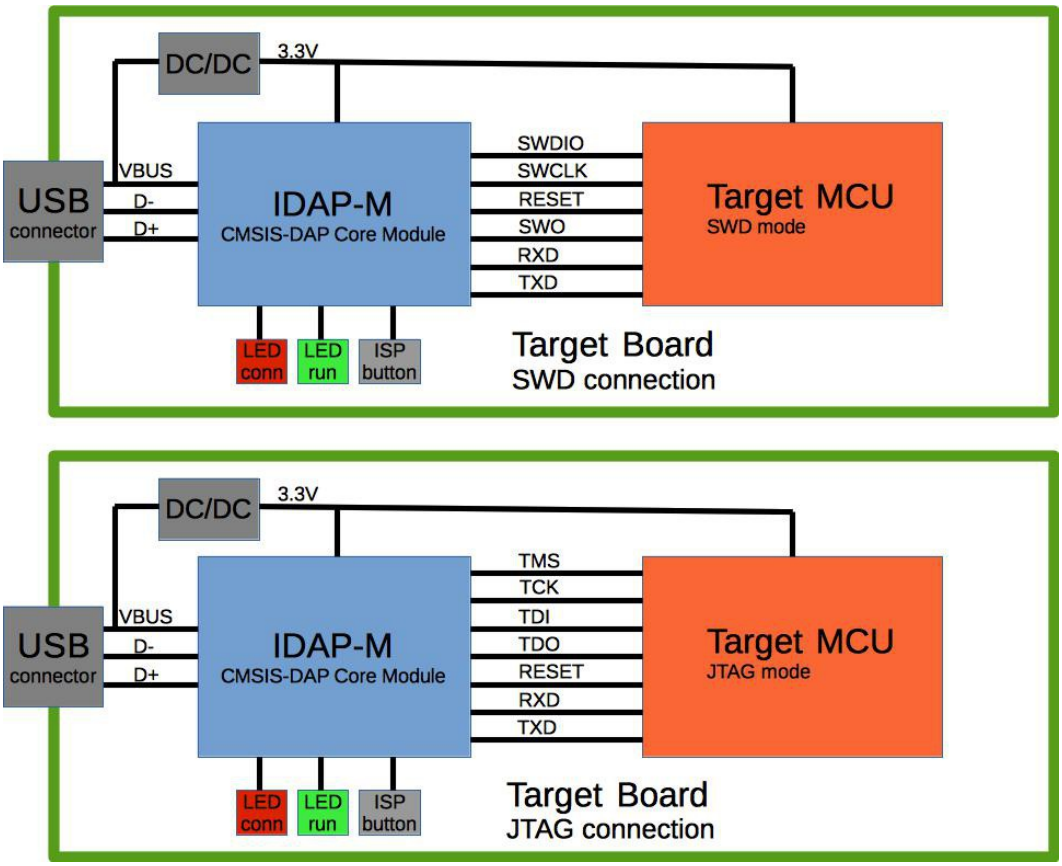


Fig. 1: Target connections

Example Schematic

The schematic bellow shows how to interface the IDAP-M to the IMM-NRF51x22 to make an mBed board.

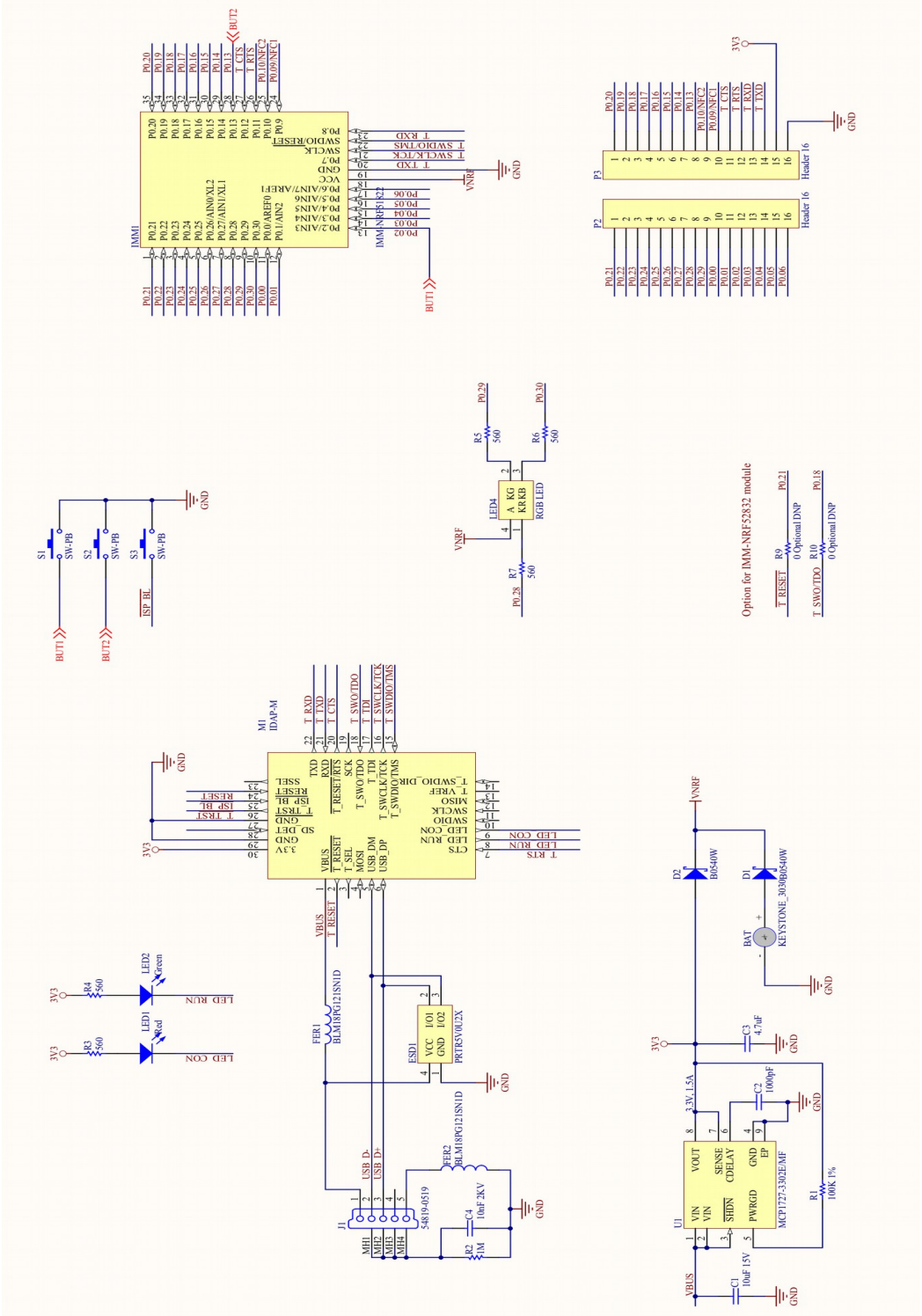


Fig. 2: mBed Bluetooth board using IDAP-M and IMM-NRF51x22 module

Switches

S1 – ISP boot/Program

This button is used to put the IDAP-Link into ISP bootloader for firmware update. Keep this button press during power up.
When the IDAP-Link is power up without connecting to PC, this button is used to activate programming target with firmware load from the microSD card.

S2 – IDAP-Link Reset

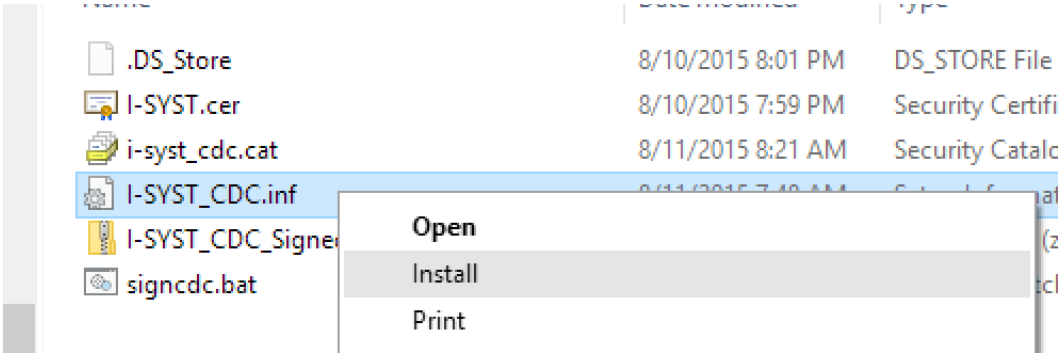
This button will reset the IDAP-Link board. To put the IDAP-Link in bootloader for firmware update. Press this reset button with the S1 (ISP) button, release S2 while keeping S1 pressed for 3 sec.

Windows CDC driver installation

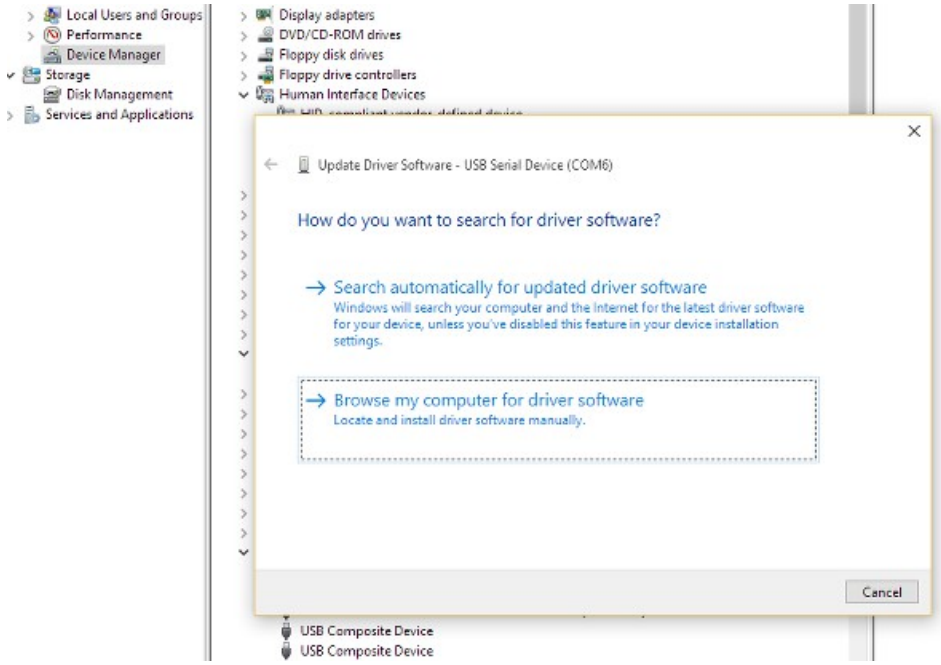
Windows 10 can now automatically detect and install CDC device without requiring to external drivers. Other Windows versions are unable to automatically install CDC driver. Follows these steps for manual driver installation.

Download the Windows driver and software from
http://sourceforge.net/projects/idaplinkfirmware/files/?source=navbar

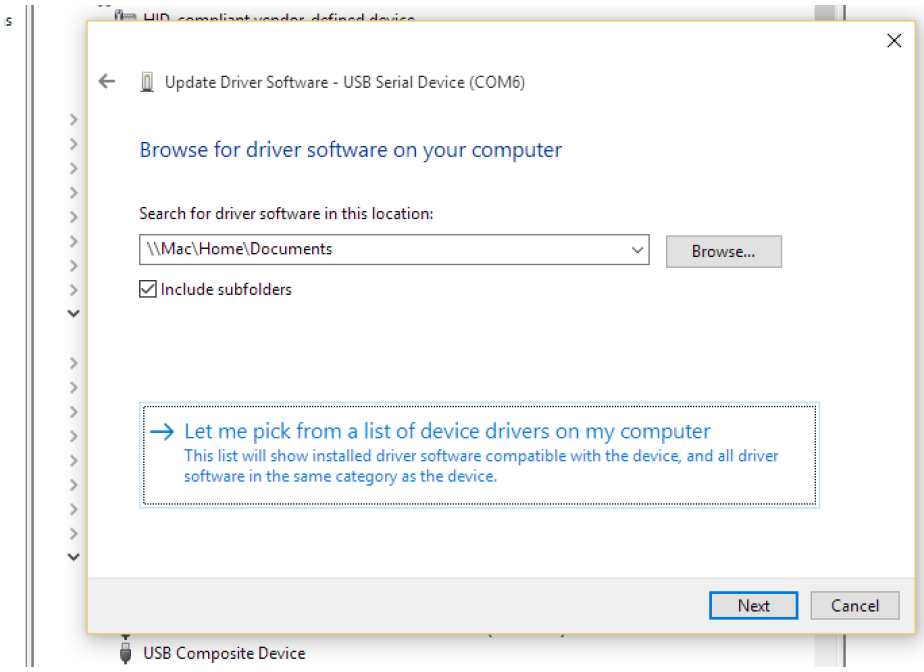
Install the driver .inf file by right-clicking on the .inf file then select “install” from the popup menu



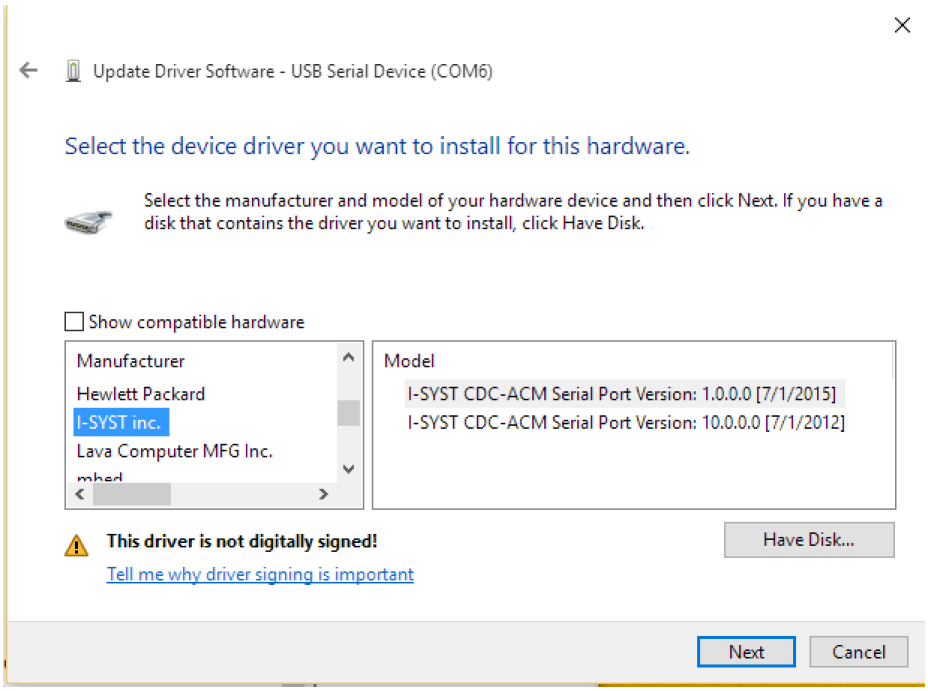
Locate the Install the CDC device from the Windows “Device Manager”. Right-click and select update driver… Select “Browse my computer..”



Select “Let me pick from a list...”



Uncheck the “Show compatible hardware” checkbox. Then locate “I-SYST inc.” from the Manufacturer list to install the driver



IDAP-M Firmware Update

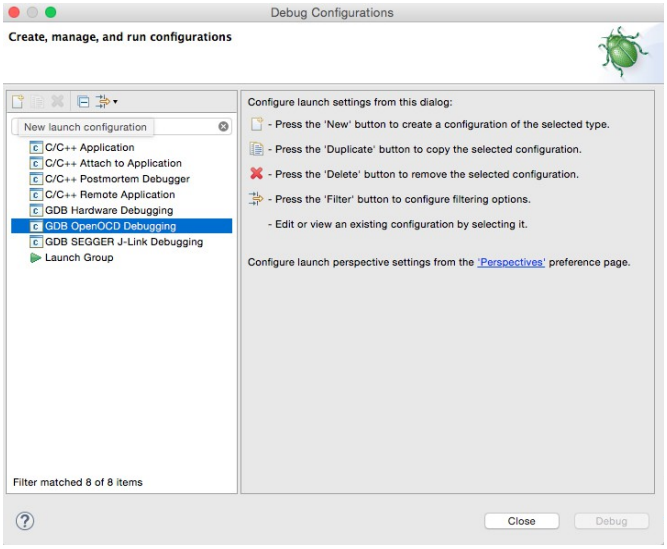
Boot the IDAP-Link into ISP mode by pressing the S1 (ISP) button and the S2 (RESET) at the same time. Release S2 while keeping the S1 pressed for about 3 sec. The IDAP-Link will appear to the PC as a removable disk with volume name 'CRP DISABLD'. Copy the new firmware.bin over to replace the old one. On Windows 8, the old firmware.bin must be deleted before copying the new one over.

Note : This process seem not to work on OSX due to NXP ROM firmware bug. In order to update firmware on OSX. A shell cp command is required.

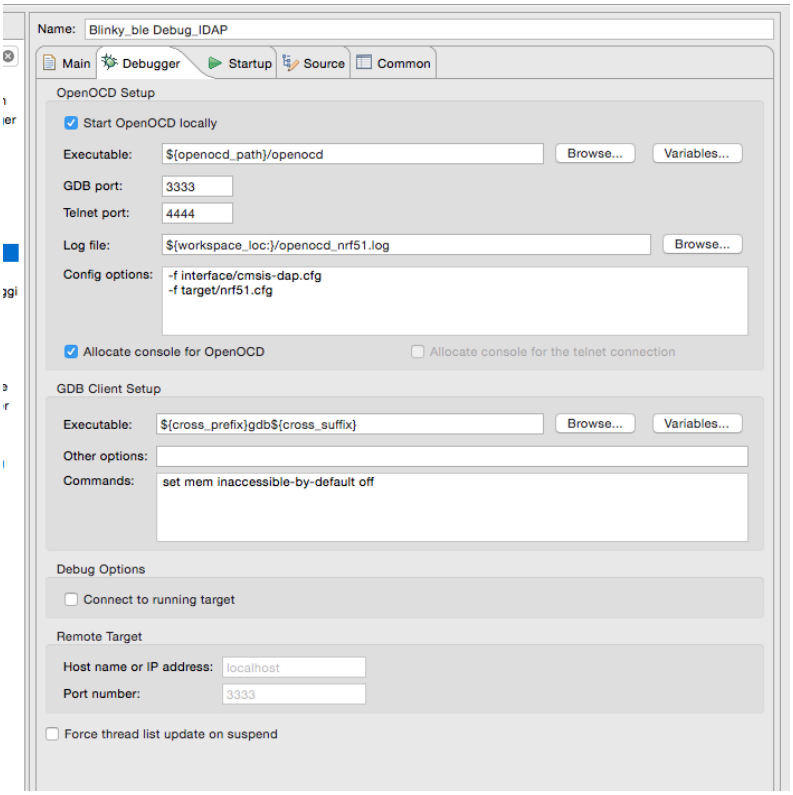
```
cp firmware.bin "/Volumes/CRP DISABLD"
```

Eclipse Development Evironment

The OpenOCD version 0.9 or above is required to use with Eclipse IDE. For Eclipse setup, follow the blog site http://embeddedsoftdev.blogspot.ca/p/eclipse.html. To enable debugging in Eclipse, select the menu Run/Debug Configuration. A popup as bellow will appear. Then create new GDB OpenOCD debugging configuration.



In the OpenOCD configuration popup, select the Debugger tab to configure OpenOCD. OpenOCD requires configuration files .cfg for the target device and the interface device. The interface device should be set with *-f interface/cmsis-dap*. The target device depends on which MCU being used. The picture bellow shows configuration example for the nRF51 series.



Creating custom target core support

The IDAP-Link™/M firmware is very flexible. It support dynamic target core selection. The new target core selection is done using the IDAPSetTarget program. This program uploads target core data into the IDAP-Link™/M board. Hence allowing target core selection without requiring a dedicated firmware. This section will show how to create the target core data for a custom device.

Target Flash Programming

Flash programming is very dependent on the target MCU. Each manufacturer and device family has their own way to allow programming of the device. Most devices do not allow writing to program memory section externally but via internal firmware. Therefore a special firmware with a few functions running of the RAM memory section to provide support for Flash programming of the target is required. Bellow is a template to implement the functions require by IDAP-Link™/M. This firmware needs to be compiled as free standing position independent. The GCC compile flags are -ffreestanding -fPIC. There is no linker script needed.

```
/*
 * Template to create
 * target Flash algorithm for IDAP-Link/M
 *
 * NOTE : This code must be compiled in freestanding & position independent mode
 * gcc flags : -ffreestanding -fPIC
 *
 * Function parameters are passed via registers
 *        r0 : First param
 *        r1 : 2nd param
 *        ...
 *
 * Copyright 2015, I-SYST inc.
 */

#include <stdint.h>

// Main entry breakpoint
//
int main()
{
     __asm("BKPT");
     __asm("BKPT");
     return 0;
}

// BSP initialization
//
// IDAP-Link will call this to initialize target
//
// @return     0 - success
//
int Init()
{
     return 0;
}

//
// Permform mass erase
//
// @return     0 - success
int EraseAll()
{
     return 0;
}

//
// Erase n consecutive Flash page
//
// @param      PageAddr  : Start of page address.  This is absolute address
//             NbPage    : Nb of pages to erase

// @return     0 - success
int ErasePage(uint32_t PageAddr, int NbPage)
{
     return 0;
}

//
// Blank check
//
// @param      Addr : Start location to check
//             Len  : Length in bytes to check
//
// @return     0 - success
//
int BlankCheck(uint32_t Addr, int32_t Len)
{
     return 0;
}

//
```

```
// Verify programmed block
//
// @param      Addr      : Start address to verify
//             *pData    : Pointer to RAM location containing data to verify
//             Len       : Length in byte to verify
//
// @return     0 - succcess
//
int Verify(uint32_t Addr, uint8_t *pData, uint32_t Len)
{
     return 0;
}

//
// Program Flash.  This ooperation does verify that data are written correctly
//
// @param      Addr      : Start address to program
//             *pData    : Pointer to RAM location containing data to be programmed
//             Len       : Number of byte to write
//
// @return     0 - success, only of verify passed
//
int Program(uint32_t Addr, uint8_t *pData, uint32_t Len)
{
     return Verify(Addr, pData, Len);
}

//
// Post-processing after programming completed.  This function is optional.
// It will be called after programming completed if entry is set in the
// TARGET_DESC structure
//
// @param      FIdxFlag : Indicating which file was flashed
//                        Bit 0 – Set if file1 was flashed
//                        Bit 1 – Set if file2 was flashed
//                        Bit 2 – Set if file3 was flashed
//                        ...
//             Parm1-4 : User defined
//
// @return     0 - success
//
int UserFunction(uint32_t FIdxFlag, uint32_t Parm1, uint32_t Parm2, uint32_t Parm3, uint32_t Parm4 )
{
     return 0;
}
```

Data structure defining target device

```
/*
 * target_desc.h
 *
 * This file defines data structure for the creation of target programming algorithm
 * to be loaded by IDAP-Link/M.  It is to allow users to create their own custom algorithm
 *
 * Created by Hoan on 2015-02-01.
 * Copyright (c) 2015 I-SYST inc. All rights reserved.
 *
 */
#include <stdint.h>

#ifndef __TARGET_DESC_H__
#define __TARGET_DESC_H__

#pragma pack(push, 4)

// Consecutive memory section
typedef struct _Memory_Section {
     uint32_t  PgSize;               // Page size, this is a page erase size
     uint32_t  TotalSize;            // Total size in bytes
     uint32_t  StartAddr;            // Mem block start address
} MEMSECT;

typedef enum _Firmware_File_Type : uint8_t {
     FW_FTYPE_NONE = 0,
     FW_FTYPE_BIN = 1,
     FW_FTYPE_HEX = 2
} FW_FTYPE;

// Tartget MCU max name length
#define TARGET_NAME_LEN                         20

// Specialty MCUs may have multiple firmware to be
// programmed, main firmware (App) + Bootloader + Comm Stack.  For example the
// Nordic nRF5x has Softdevice, main app, and DFU
#define FWFILE_MAX_FILE                         3               // Max number of firmware supported
#define FWFILE_NAME_MAX_LEN                     16              // Max length for firmware name

/*
 * This structure defines the target MCU and its flash loader
 */
typedef struct _Target_Descriptor {
     uint32_t       Size:16;                    // Length this structure sizeof(TARGET_DESC)
     uint32_t       Vers:16;                    // Version
     uint32_t       IdCode;                     // Chip IDCODE for detection
     char           Name[TARGET_NAME_LEN];      // Chip name
     MEMSECT        ProgSect;                   // Program memory section desc
     MEMSECT        DataSect;                   // Main data memory section desc
     uint8_t        mBedId[4];                  // mBed board ID
     uint8_t        mBedSecret[9];              // mBed board secret
     FW_FTYPE       FwFType;                    // mBed compiled result file type
     int            NbFWFiles;                  // Number of firmware files
     char           FWFileName[FWFILE_MAX_FILE][FWFILE_NAME_MAX_LEN];          // Firmware file names
     int            DapType;                    // Debug interface port 1 = SWD or 2 = Jtag
     int            JtagIRLen;                  // J-Tag IR length in bits
     uint32_t       SwdCfg;                     // SWD config :
                                                //        Bit 0-1 : Turn around cycle (1-4 clocks)
                                                //        Bit 2 : Data phase on WAIT/FAULT (0 - no, 1 - yes)
```

```
    uint32_t    InitEntry;                  // Board init function
    uint32_t    EreaseAllEntry;             // Erase All function
    uint32_t    EreasePageEntry;            // Erase Page function
    uint32_t    BlankCheckEntry;            // Blank check function
    uint32_t    ProgramEntry;               // Flash program function
    uint32_t    VerifyEntry;                // Verify function
    uint32_t    UserEntry;                  // User defined function. If non null, it will be call
                                            // after programming complete
    uint32_t    UserParam[4];               // Parameters for user defined function
    uint32_t    BrkPoint;                   // Break point function
    uint32_t    StaticBase;                 // Pointer to bss area for target code
    uint32_t    StackPointer;               // Stack pointer
    uint32_t    Buffer;                     // Data buffer
    uint32_t    BufferLen;                  // Data buffer length
    uint32_t    LoaderStart;                // RAM target location for loader code
    uint32_t    LoaderSize;                 // Size in byte of loader code
} TARGET_DESC;


#pragma pack(pop)

#endif
```

Definition example

```
TARGET_DESC g_TargetData = {
    // TARGET_DESC
    sizeof(TARGET_DESC),
    TARGET_DESC_VERS,
    0xbb11477,                  // IDCODE
    "LPC11U35",                 // Target name
    {4096, 64*1024, 0},         // Program memory
    {1, 8*1024, 0x20000000},
    {0,},                       // mBed ID
    {0,},                       // mBed secret code
    FW_FTYPE_HEX,               // Firmware type hex
    1,                          // 2 firmware files
    {
        "firmware.hex",
    },
    1,                          // DAP interface 1 = SWD, 2 = JTAG
    0,                          // JTAG IR length in bits
    5,                          // SWD : 1 clock turn around, Data phase on
    0x20000001,                 // Init function entry
    0x20000011,                 // EraseAll function entry
    0x20000021,                 // ErasePage function entry
    0x20000031,                 // BlankCheck function entry
    0x20000041,                 // Program function entry
    0x20000051,                 // Verify function entry
    NULL,                       // User function entry
    {0,0,0,0},                  // User function parameters
    0x20000071,                 // Main breakpoint function entry
    0x20000100,                 // BSS pointer
    0x20020000,                 // Stack pointer
    0x20000200,                 // data ram buffer
    4096,                       // data ram length
    0x20000000,                 // RAM location to load target algorithm code
    1024                        // Length of target code in bytes
};
```