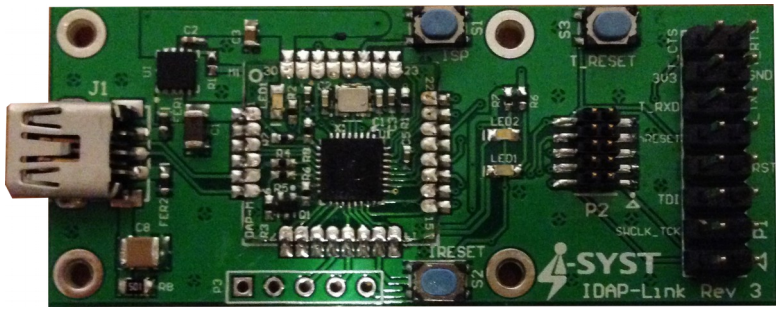
User Guide

IDAP-Link™

CMSIS-DAP Debug JTAG



i-SYST INC

Limited Warranty

The IDAP-Link 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.

In no event shall I-SYST inc. be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of I-SYST inc. hardware and software, even if advised of the possibility of such damage.

I-SYST inc. products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury.
I-SYST inc. customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify I-SYST inc. for any damages resulting from such improper use or sale.

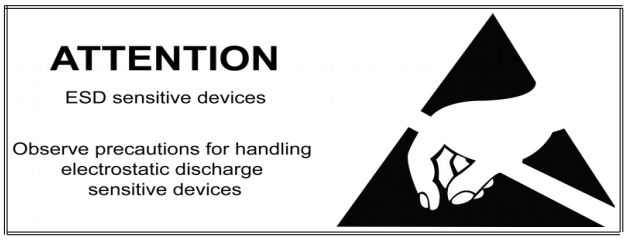


Table of Contents

Introduction

The IDAP-Link™ is a low cost CMSIS-DAP JTAG debug probe with enhanced features. It can do more with it than just debugging. It will appears as a USB disk drive. This allows firmware flashing easily by copying the firmware file over without requiring any special flashing software and work instantly with any operating system. It provides a UART to USB bridge for communication between the target device and the PC. It also provides a regulated 3.3V to directly power the target device without addition power source by taking advantage of the USB power source. These feature turn the target device into mBed enable. It can be used as an ultra low cost solution to production programming. BSP is provide for use with Open Source CMSIS-DAP firmware from mBed.org which make it totally customizable.

Features :

- Support both SWD & JTAG mode
- Debug compatibility with most IDE such as Keil, CrossWorks, Eclipse, etc..
- Onboard 3.3V regulator to power the target device
- UART to USB bridge for communication between target and PC
- mBed enabled target board
- Firmware flashing by drag & drop simply by copying file over
- micro-SD slot for flash programming without a PC
- BSP is provided for Open Source CMSIS-DAP firmware from mBed.org

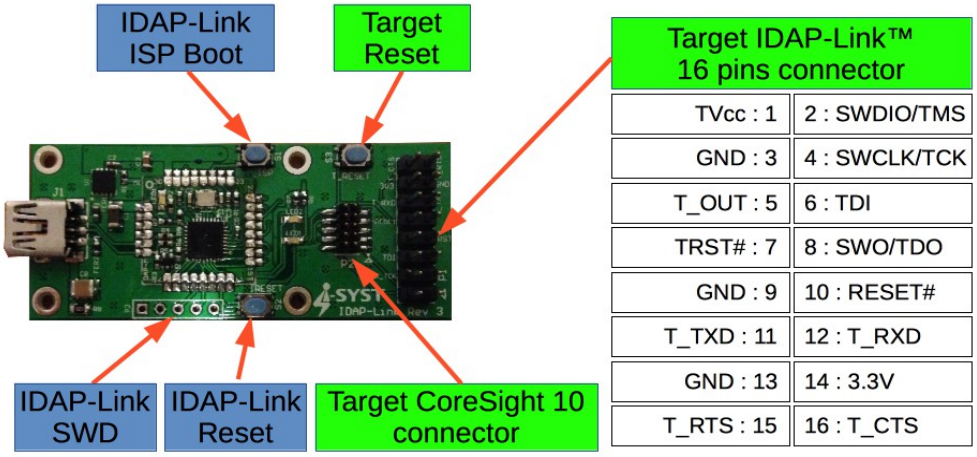


Fig. 1: IDAP-Link™ Rev. 3

Connectors

P1 – IDAP-Link connector

The connector P1 is the IDAP-Link™ target connector.

TVCC	1	2	SWDIO/TMS
GND	3	4	SWCLK/TCK
GND	5	6	TDI
TRST	7	8	SWO/TDO
GND	9	10	RESET
T_TXD	11	12	T_RXD
GND	13	14	3.3V

P1 – IDAP-Link™ 14 pins connector

Starting revision 3, the connector P1 is 16 pins

TVCC	1	2	SWDIO/TMS
GND	3	4	SWCLK/TCK
T_OUT	5	6	TDI
TRST	7	8	SWO/TDO
GND	9	10	RESET
T_TXD	11	12	T_RXD
GND	13	14	3.3V
T_RTS	15	16	T_CTS

P1 – IDAP-Link™ 16 pins connector

TVCC : Target Vcc. Coming from the target device.
SWDIO/TMS, SWCLK/TCK : SWD connections.
SWDIO/TMS, SWCLK/TCK, TDI, TRST, SWO/TDO: JTAG connections
3.3V: This the 3.3V supply output from the IDAP-Link. This can be used to power target board.
T_TXD, T_RXD, T_RTS, T_CTS: UART connections
T_OUT : Digital I/O output reserved for future use
GND: Digital signal ground.

Connecting to target

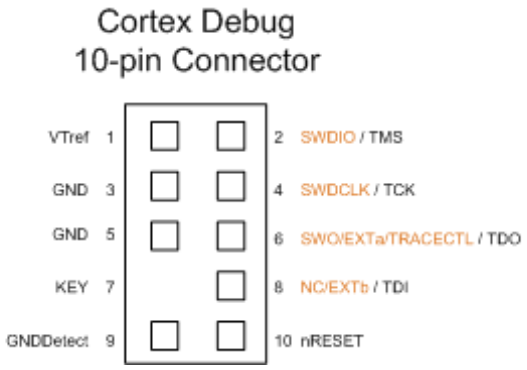
SWD mode connection to target requires the following pins : TVCC, GND, SWDIO/TMS, SWCLK/TCK

JTAG mode connection to target requires at least these pins : TVCC, GND, SWDIO/TMS, SWCLK/TCK, TDI, TDO. TRST is optional

UART bridge : Connect T_TXD to UART TXD on target, T_RXD to UART RXD, T_RTS to UART RTS, T_CTS to UART CTS on target and GND

P2 – ARM Coresight 10 pins connector

The P2 is the ARM standard 1.27mm pitch 10 pins CoreSight connector.

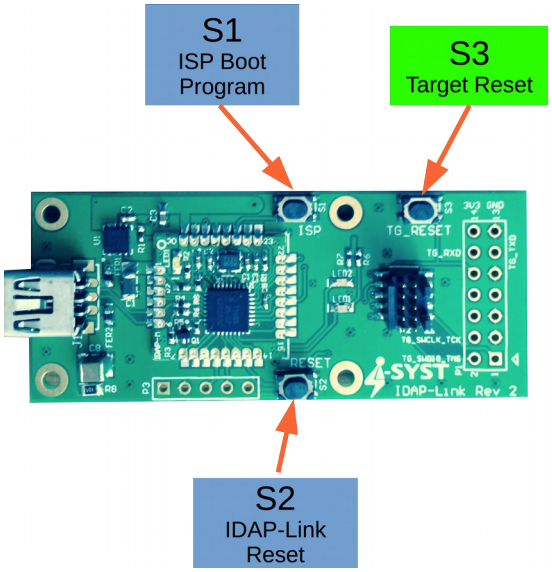


P3 – IDAP-Link core SWD connector

The P3 is the SWD port for programming firmware upgrade of the IDAP-Link.

Pin 1 : 3.3V
Pin 2 : GND
Pin 3 : SWDIO
Pin 4 : SWCLK

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.

S3 – Target hardware Reset

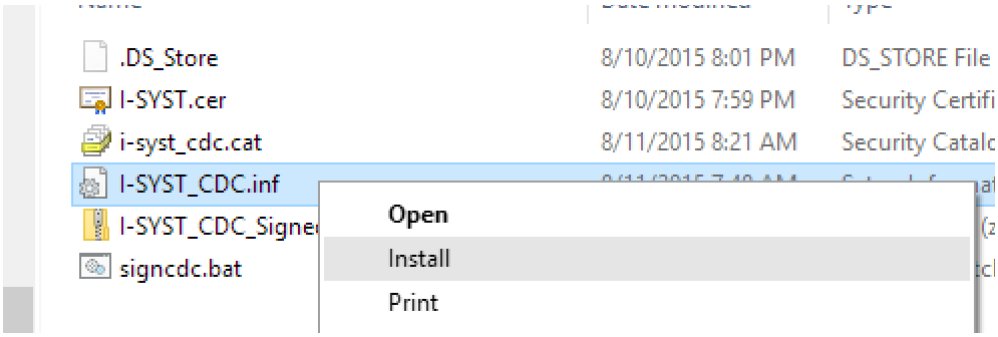
This button is connected to the target reset pin. Pressing this button will reset target if the target has reset pin connected to the JTAG/SWD connector (P1 or P2)

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.

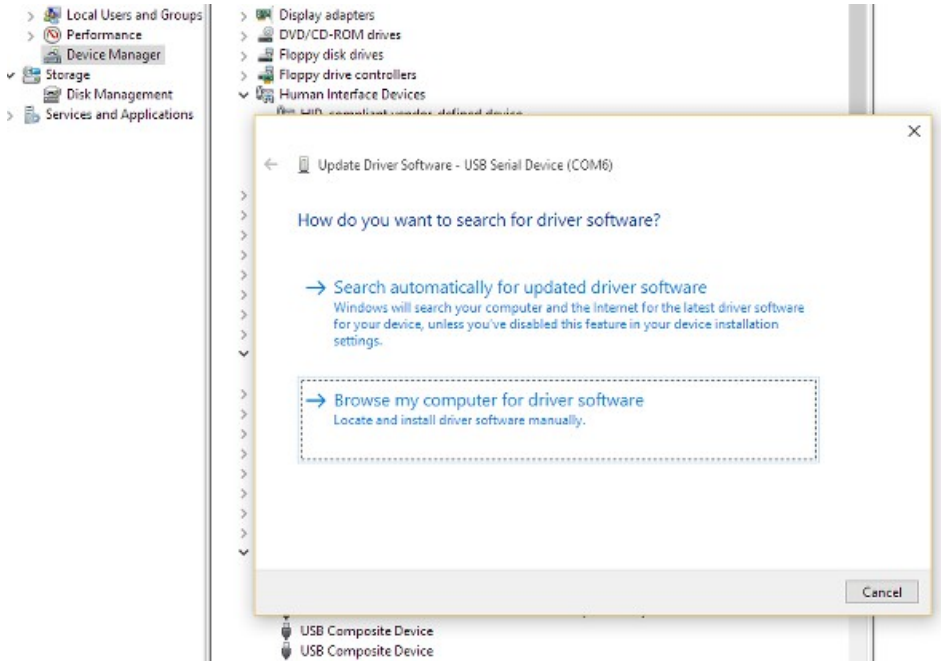
Download the Windows driver and software from
http://sourceforge.net/projects/idaplinkfirmware/files/?source=navbar

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

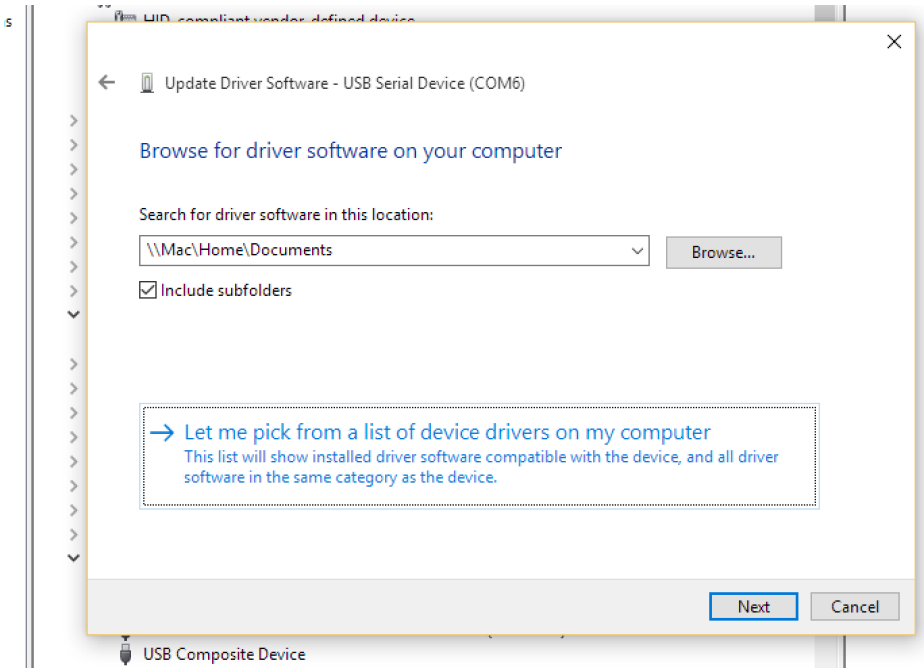


Locate the Install the CDC device from the Windows “Device Manager”. Right-click and select update driver…

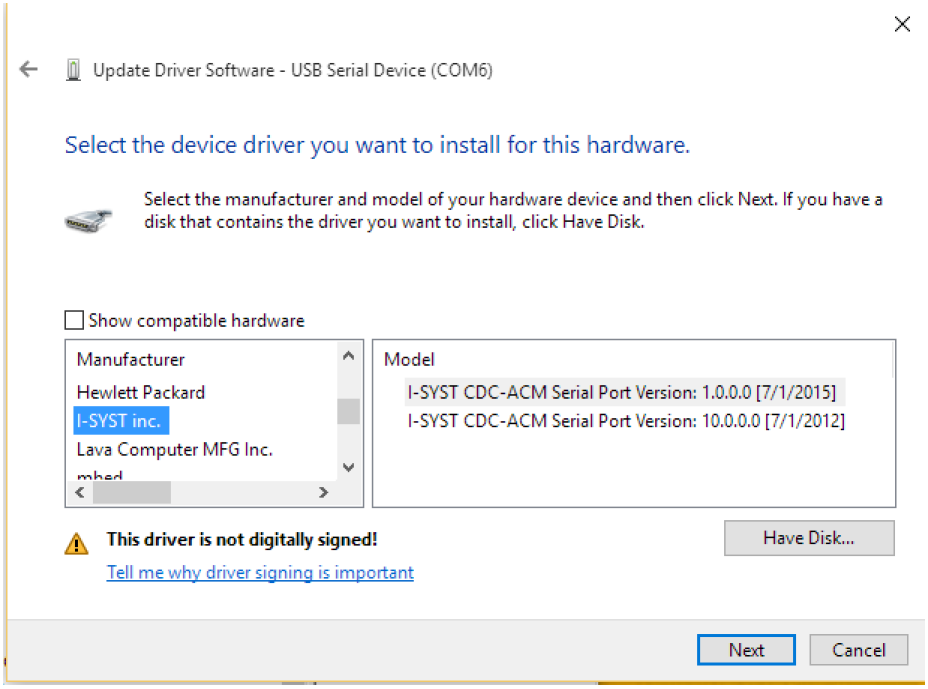
Select “Browse my computer..”



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-Link™ Firmware Update

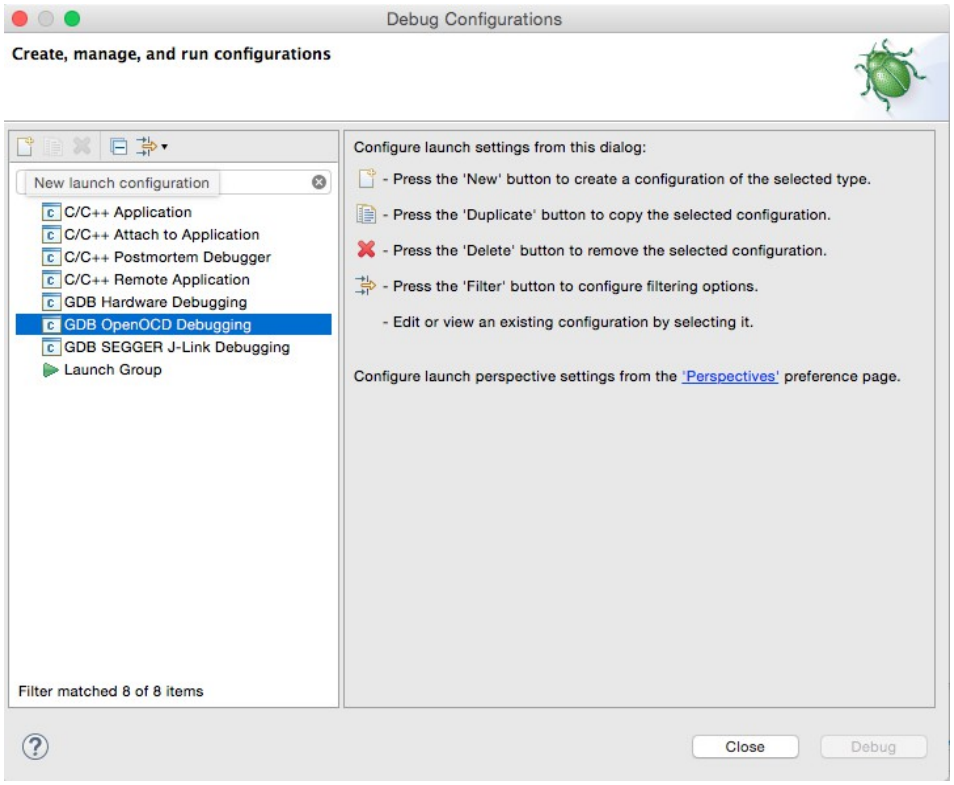
Boot the IDAP-Link into ISP mode by pressing the S1 (ISP) button and the S2 (RESET) at the same time. Release S2 while keeping the S1 pressed for about 3 sec. The IDAP-Link will appear to the PC as a removable disk with volume name 'CRP DISABLD'. Copy the new firmware.bin over to replace the old one.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

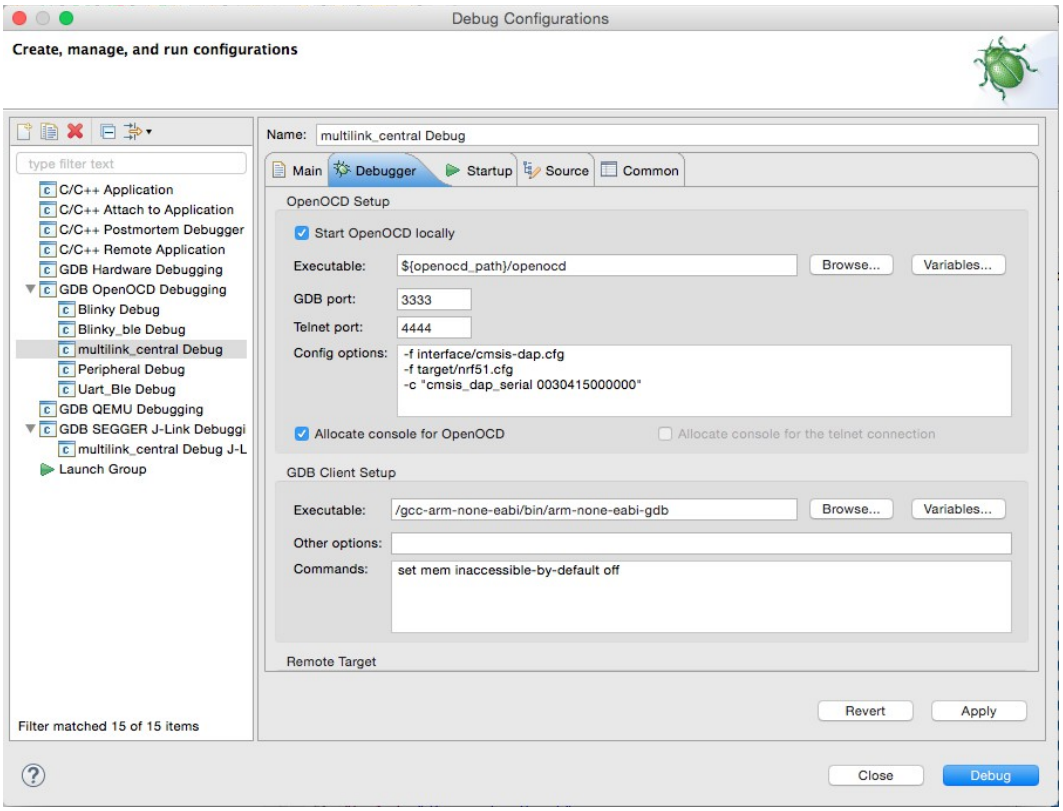
Eclipse Development Evironment

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.



OpenOCD with multi-board

When multiple IDAP-Links are connected to the PC, OpenOCD needs to know which is to be used for the debug session. This can be accomplished using the OpenOCD command 'cmsis_dap_serial' to select the target board to use using its serial #. Type in the Config options box *-c "cmsis_dap_serial #######"* where ###### is the serial number. The image above shows the selection of the IDAP-Link with serial number '0030415000000' for the debug session.

Target Flash programming with microSD

The IDAP-Link has an onboard microSD interface. This interface allow Flash programming as target device without requiring a PC. This is accomplished by following the procedure bellow.

- Connect the IDAP-Link to PC. Run the command line IDAPSetTarget program to select the target device. Pass the index number of the target device as argument to the IDAPSetTarget program. Running the IDAPSetTarget without argument will display a list of supported target device along with its index number. Once the target is successfully programmed into the IDAP-Link, the a list of require firmware filename is listed. This step is needed only when selecting a different target core.
- Copy the the firmware files with predefined filenames onto the microSD card. The firmware file name must be exactly the same as those listed during the target selection step above.
- Power up the IDAP-Link or press the Reset button (S2) with the microSD in the slot. The microSD card must be inserted prior to power up the IDAP-Link otherwise it will not switch to microSD programming mode
- If the IDAP-Link is still connected to the PC. The microSD will show up. Eject it from the PC prior to start flashing.
- Press ISP/PROG button (S1) to start Flashing. The green LED will turn on or blink. The programming status will be also be printed to the USB CDC COM port.
- Once programming completed, the green LED will turns off. If programming failed, the red LED will blink at 1 sec interval. All LED are off when programming is successful.

Parallel Flashing Nordic nRF5x using multiple IDAP-Link™

IDAPnRFProg is a command line tool for Flashing Nordic nRF51 & nRF52 SoC. It is available on Windows and OSX. It can flash Softdevice, Application firmware and DFU hex files all at once without requiring to merge them first.

Flashing all 3 elements :

```
IDAPnRFProg softdevice.hex Blinky_ble.hex dfu_nrf51.hex
```

Flashing softdevice only :

```
IDAPnRFProg softdevice.hex
```

or just a test program

```
IDAPnRFProg Test.hex
```

Furthermore, IDAPnRFPRog automatically scan USB for all connected IDAP-Link and flash all devices in parallel. It is a great tool for production programming.

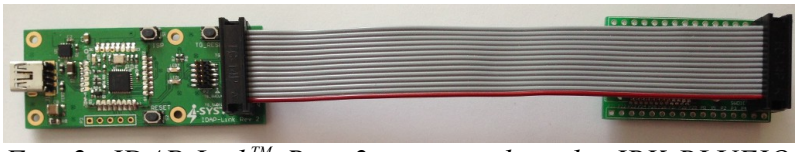


Fig. 2: IDAP-Link™ Rev. 2 connected to the IBK-BLUEIO Rev. 0. Breakout with IMM-NRF51x series module

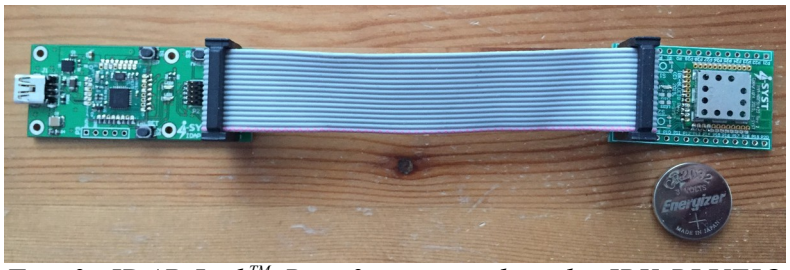


Fig. 3: IDAP-Link™ Rev. 3 connected to the IBK-BLUEIO Rev. 1. Breakout with IMM-NRF5x series module

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 -fpie. 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 -fpie
 * linker flag : -pie
 *
 * Function parameters are passed via registers
 *        r0 : First param
 *        r1 : 2nd param
 *        ...
 *
 * Copyright 2014-2016, I-SYST inc.
 */

#include <stdint.h>

#include "target_desc.h"

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

// IDAP-Link will call this first to perform initialize and identify the target.
//
// @param pChipInfo : Pointer to buffer to be fill with CHIP_INFO data
//
// @return 0 - success
//         On success buffer location is filled with CHIP_INFO data
//
int Init(CHIP_INFO *pChipInfo)
{
    // Add code to detect and fill CHIP_INFO
    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
//
// @return0 - 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-1 - success
//                 If failed, returns address of non blank page
//
int BlankCheck(uint32_t Addr, int32_t Len)
{
    return -1;
}

//
// Enable direct Flash write.  This is an option function to allow
// directly write to Flash without passing by Program function bellow
// If this feature is not supported, NULL must be set in TARGET_DESC entry
//
// @param En : true - Enable write
//             false - Disable write
//
// @return None
//
void DirectFlashWrite(bool En)
{

}

//
// Enable read back protection
//
void Protect()
{

}

//
// Perform write buffer to Flash.  This ooperation does verify that data are written
// correctly.  This  function is invisible to IDAP
//
// @param Addr      : Start address to program
//        *pData    : Pointer to RAM location containing data to be programmed
//        Len       : Number of byte to write
//        bErase    : True – Erase before write
//                    False – No Erase
//
// @return None
//
//
void FlashWrite(uint32_t Addr, uint8_t *pData, int32_t Len, bool bErase)
{
}

//
// Checksum verify
//
// @param Addr : Start address to verify
//        Len  : Length in byte to verify
//               CheckSum: Check sum value.
//
// @return checksum value
//           0 - good
//           x - bad checksum
//
int Verify(uint32_t Addr, int32_t Len, uint32_t Checksum)
{
    uint32_t *p = (uint32_t*)(Addr & 0xFFFFFFFC);
    uint32_t cs = 0;

    while (Len > 0)
    {
        cs += *p++;
        Len -= 4;
    }

    return cs + CheckSum;
}

//
// Postprocessing 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 file1 was flashed
//                   Bit 1 - Set file2 was flashed
//                   Bit 2 - Set file3 was flashed
//        Parm1-4 : 4 User defined parameters
//
// @return0 - 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

```
/*----------------------------------------------------------------------------
File   : target_desc.h

Author : Hoang Nguyen Hoan          Feb. 1, 2015

Desc   : 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

Copyright (c) 2015, I-SYST inc., all rights reserved

For info or contributing contact : hnhoan at i-syst dot com

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

----------------------------------------------------------------------------
Modified by         Date            Description
Hoan                                    July 3, 2016        Better support for custom algorithm.
                                                            Supports JTag interface
----------------------------------------------------------------------------*/
#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 {
typedef enum {
    FW_FTYPE_NONE,
    FW_FTYPE_BIN,
    FW_FTYPE_HEX,
} FW_FTYPE;

// Tartget MCU max name length
#define TARGET_NAME_LEN                 16

// 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

#define TARGET_DESC_VERS                0x100       // Vers bit0-7 : Minor, bit8-15 : major

// This structure describes the chip variant info
// it is a variable size structure.  It should normally be casted to data buffer
// containing the info
typedef struct _Silicon_Info {
    uint32_t    Id;             // Silicon Id
    uint32_t    Rev;            // Silicon revision
    uint32_t    VarId;                  // Silicon variant id
    uint32_t    Package;        // Package Id
    uint64_t    Uid;            // Uniq ID
    char        Name[20];       // Variant name
    MEMSECT     ProgSect;       // Program memory info (Flash or RAM)
    MEMSECT     DataSect;       // Data memory info (RAM)
} CHIP_INFO;

/*
 * 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
    FW_FTYPE        FwFType;                    // 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;                  // JTag IR length in bits
    uint32_t        SwdCfg;                     // SWD config :
                                                //         Bit 0-1 : Turn around cycle (1-4 clocks).
                                                //         Bit 2 : Data on WAIT/FAULT (0 - no, 1 - yes)
    uint32_t        InitEntry;                  // Target init function
    uint32_t        EreaseAllEntry;             // Erase All function
    uint32_t        EreasePageEntry;            // Erase Page function
    uint32_t        BlankCheckEntry;            // Blank check function
    uint32_t        DirectFlashEntry;           // Enable/Disable direct flash write (optional)
    uint32_t        ProgramEntry;               // Flash program function
    uint32_t        VerifyEntry;                // Check sum verify function
    uint32_t        ProtectEntry;               // Read back protection 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        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 // __TARGET_DESC_H__
```

Target 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,                // Direct write function entry
     0x20000051,                // Program function entry
     0x20000061,                // Verify function entry
     NULL,                      // Read back protection function entry
     NULL,                      // User function entry
     {0,0,0,0},                 // User function parameters
     0x20000071,                // Main breakpoint function entry
     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
};
```