Marvell IMP GUI

User Guide

Ver. 1.10.XXXX

8/18/2016

Table of Contents

1. INTRODUCTION

1.1 ABOUT IMPGUI

IMPGUI (Integrated Micro Processor GUI) is a debug tool for Z80 processor, it includes four parts:

- RT Debug (*Real time debug GUI*), it has integrated the latest SDCC (*Small Device C Compiler, official website: http://sdcc.sourceforge.net/*) 3.4.0. The current version supports C and assembly source level debug, besides it also can support project management, communication port and protect memory, etc.
- IMP page is an internal register level debug page. In this page, designers can access all of the Z80 related registers and IO, besides it also provides some simple operations to control Z80.
- Management page is a memory and IO operation tool. In this page, designers can access memory and IO registers, besides it also can support load and dump image file (*So far, it only support IHX format file*).
- EEPROM Tool page is an EEPROM configuration and building page. In this page, you can configure speed, manufacturer table, enabling CRC (*This feature only enables in advanced mode.*)

So far, it can support Peridot and Topaz related product Z80 module debugging.

1.2 THIRD PARTY APPLICATION LICENSE

SDCC compiler suite is a collection of several components derived from different sources with different FOSS licenses. See the sdccman.pdf document, chapter "SDCC Suite Licenses" for details.
The code or object files generated by SDCC suite are not licensed, so they can be used in FLOSS or proprietary (closed source) applications.
The great majority of sdcc run-time libraries are licensed under the GPLv2+LE which allow linking of sdcc run time libraries with proprietary (closed source) applications.
Exception is pic device libraries and header files which are derived from Microchip header (.inc) and linker script (.lkr) files. Microchip requires that "The header files should state that they are only to be used with authentic Microchip devices" which makes them incompatible with the GPL. Pic device libraries and header files are located at non-free/lib and non-free/include directories respectively. Sdcc should be run with the --use-non-free command line option in order to include non-free header files and libraries.

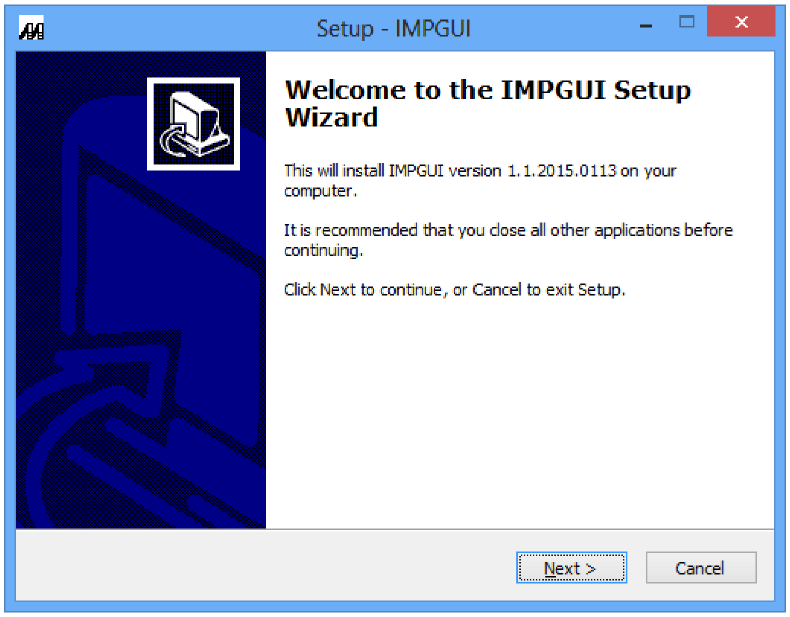
1.3 DOCUMENT

The related document is located in installation path “Doc” folder.

2. Installing IMPGUI

2.1Setup

Just double click the installation, you can see the following picture, and click next to finish the installation.



Setup - IMPGUI

Welcome to the IMPGUI Setup Wizard

This will install IMPGUI version 1.1.2015.0113 on your computer.

It is recommended that you close all other applications before continuing.

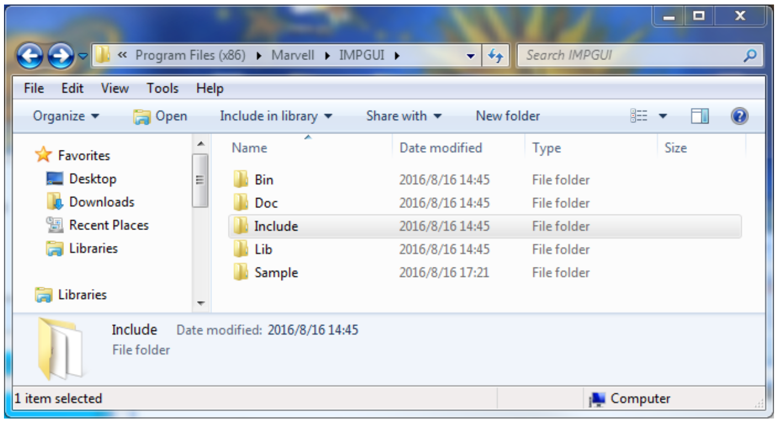
Click Next to continue, or Cancel to exit Setup.

PIC 1

2.2 Default installation path

The default path is “C:\Program Files\Marvell\IMPGUI” for win32 system, while “C:\Program Files (x86)\Marvell\IMPGUI” for win64 system. In the folder, you can see three subfolders: “Bin” (*Application*), “Doc” (*Related document*), “Lib” (*Related lib files*), “Include” (*Related header files*) and “Sample” (*Four samples*). See PIC2.

Note: The following samples are all based win64 system.



« Program Files (x86) ▸ Marvell ▸ IMPGUI ▸
Search IMPGUI
File Edit View Tools Help
Organize Open Include in library Share with New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Libraries

Name	Date modified	Type	Size
Bin	2016/8/16 14:45	File folder	
Doc	2016/8/16 14:45	File folder	
Include	2016/8/16 14:45	File folder	
Lib	2016/8/16 14:45	File folder	
Sample	2016/8/16 17:21	File folder	

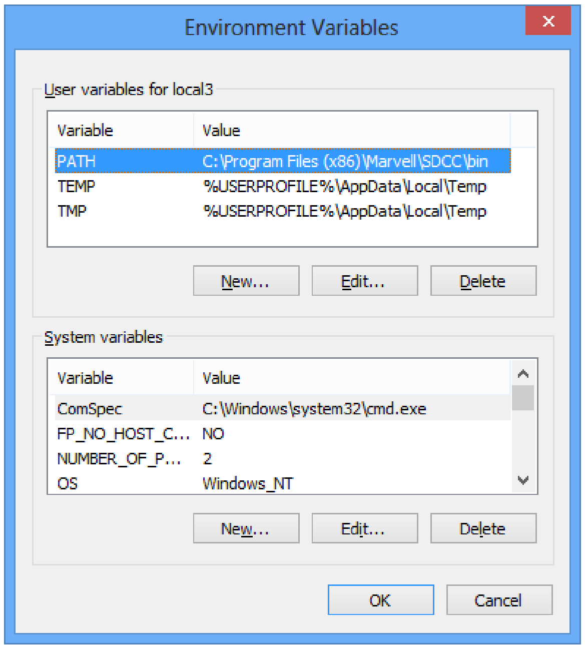
Include Date modified: 2016/8/16 14:45
File folder

1 item selected
Computer

PIC 2

2.3 ENVIRONMENT

After installation, it will add SDCC path into environment automatically. See PIC3.



Variable	Value
PATH	C:\Program Files (x86)\Marvell\SDCC\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

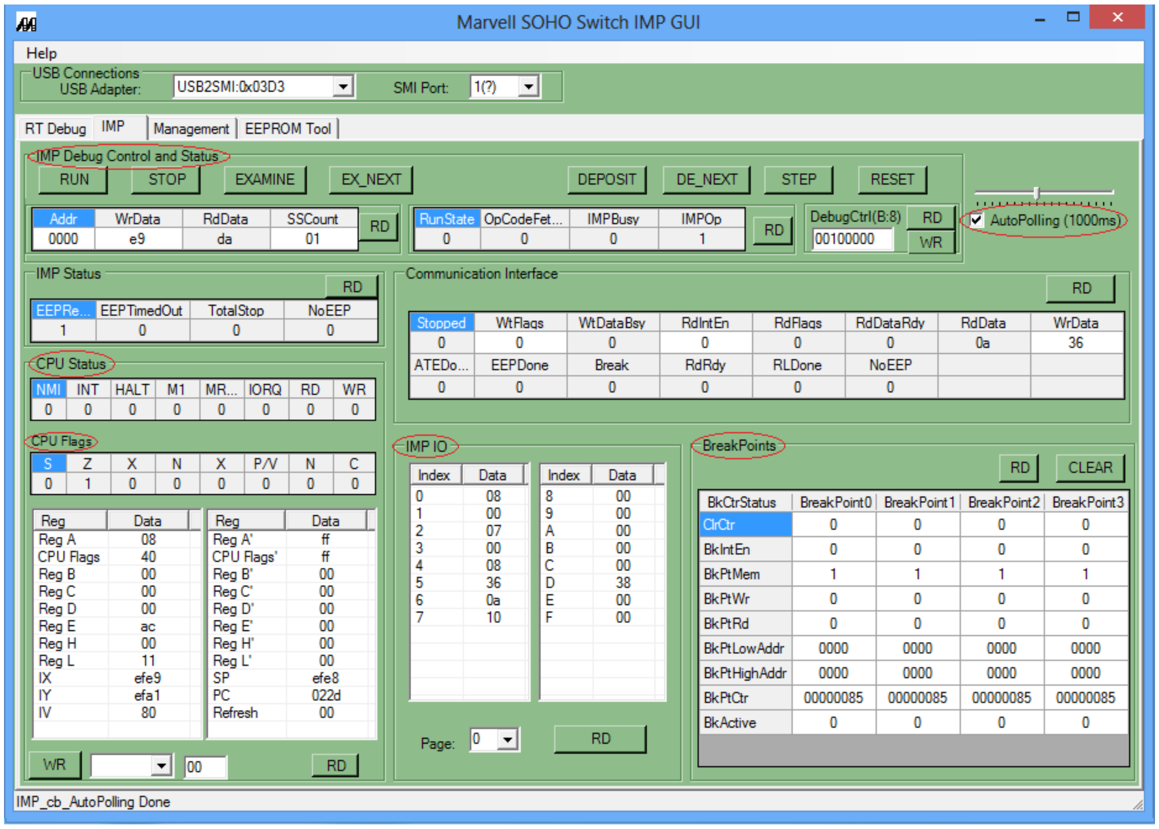
Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

PIC 3

3. USING IMPGUI

3.1 IMP PAGE

Using IMP page, you can access “IMP Debug Control and Status”, “IMP status”, “CPU Status”, “IO”, “Breakpoints” and so on. For detailed information of the registers in this page, you can refer to IMP datasheet. In CPU status zone, it provides a write button (Left bottom of above picture) for designer input, while designer can modify “IMP IO” value in the following Management page. See PIC4.



Addr	WrData	RdData	SSCount
0000	e9	da	01

RunState	OpCodeFet...	IMPBusy	IMPOp
0	0	0	1

DebugCtrl(B:8): 00100000

[✓] AutoPolling (1000ms)

IMP Status

EEPRe...	EEPTimedOut	TotalStop	NoEEP
1	0	0	0

Communication Interface

Stopped	WtFlags	WtDataBsy	RdIntEn	RdFlags	RdDataRdy	RdData	WrData
0	0	0	0	0	0	0a	36
ATEDo...	EEPDone	Break	RdRdy	RLDone	NoEEP		
0	0	0	0	0	0		

CPU Status

NMI	INT	HALT	M1	MR...	IORQ	RD	WR
0	0	0	0	0	0	0	0

CPU Flags

S	Z	X	N	X	P/V	N	C
0	1	0	0	0	0	0	0

Reg	Data	Reg	Data
Reg A	08	Reg A'	ff
CPU Flags	40	CPU Flags'	ff
Reg B	00	Reg B'	00
Reg C	00	Reg C'	00
Reg D	00	Reg D'	00
Reg E	ac	Reg E'	00
Reg H	00	Reg H'	00
Reg L	11	Reg L'	00
IX	efe9	SP	efe8
IY	efa1	PC	022d
IV	80	Refresh	00

IMP IO

Index	Data	Index	Data
0	08	8	00
1	00	9	00
2	07	A	00
3	00	B	00
4	08	C	00
5	36	D	38
6	0a	E	00
7	10	F	00

BreakPoints

BkCtrStatus	BreakPoint0	BreakPoint1	BreakPoint2	BreakPoint3
ClrCtr	0	0	0	0
BkIntEn	0	0	0	0
BkPtMem	1	1	1	1
BkPtWr	0	0	0	0
BkPtRd	0	0	0	0
BkPtLowAddr	0000	0000	0000	0000
BkPtHighAddr	0000	0000	0000	0000
BkPtCtr	00000085	00000085	00000085	00000085
BkActive	0	0	0	0

PIC 4

On top right of this page, there is an “AutoPolling” checkbox, you can change the auto-polling intervals and check it, then above selected by red circle zones will refresh automatically.

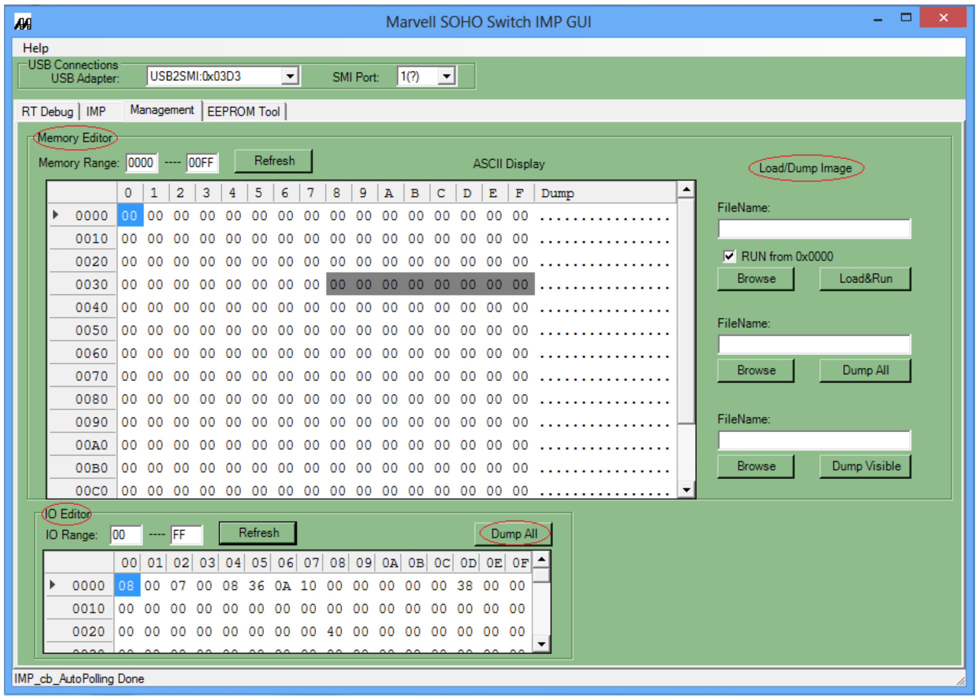
Note: When using this page, you should pay more attention to the following operation:

- *Writing some bits maybe influence “RT Debug”;*
- *Because of software breakpoint solution, one of the breakpoint values maybe different from your expectation;*
- *When you read some “Communication interface” value from this page, some characters maybe not display in communication port console as your expectation;*

3.2 Management Page

Using this page, designers can read and write memory and IO registers, besides it also can provide loading image from file and dump memory to a specified file. All of the operations in this page must make sure that CPU is in stopped status. Besides, in memory range, 0x38~0x3F are reserved, but these ranges also can be edited, so you should pay more attention if you want to touch these ranges.

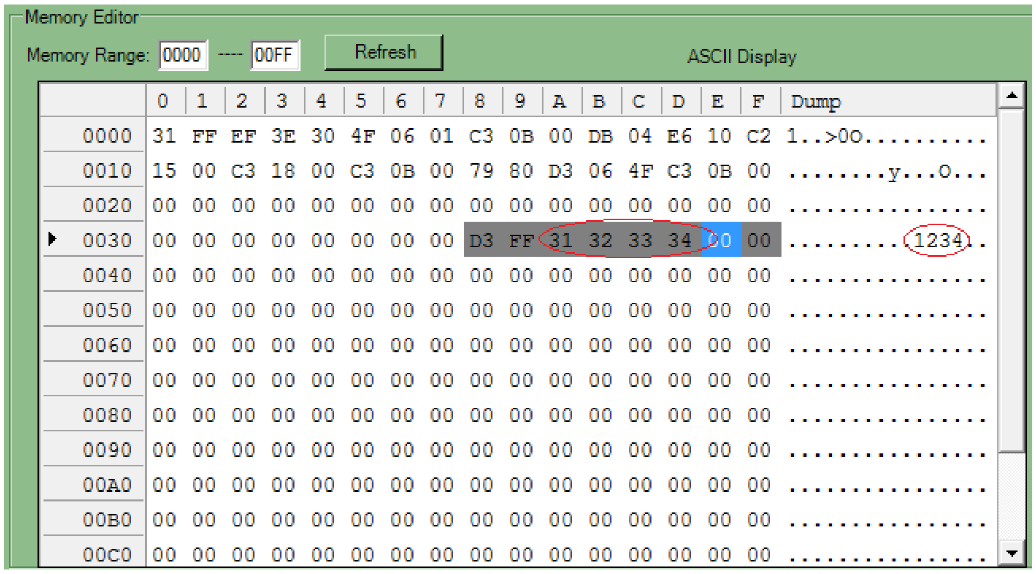
a. PIC5 will show the main features of this page. In memory editor zone, designers can edit the memory and the related ASCII code will change as soon as the value changed. See PIC6.



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  Dump
0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000  08 00 07 00 08 36 0A 10 00 00 00 00 00 38 00 00
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020  00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
```

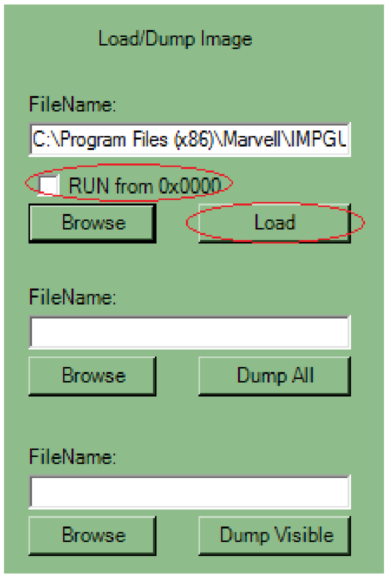
PIC 5



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  Dump
0000  31 FF EF 3E 30 4F 06 01 C3 0B 00 DB 04 E6 10 C2 1..>0O..........
0010  15 00 C3 18 00 C3 0B 00 79 80 D3 06 4F C3 0B 00 ........y...O...
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0030  00 00 00 00 00 00 00 00 D3 FF 31 32 33 34 00 00 ..........1234..
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

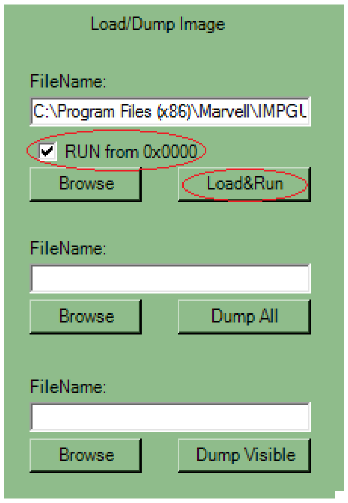
PIC 6

b. PIC7 show the function of Load Image. So far, it only support IHX format image (Intel hex format: http://en.wikipedia.org/wiki/Intel_HEX) and the suffix of the file must be ".ihx". If not check "RUN from 0x000" , and then click "Load" button after you select one ".ihx" file, the GUI will load the content into corresponding memory.



PIC 7

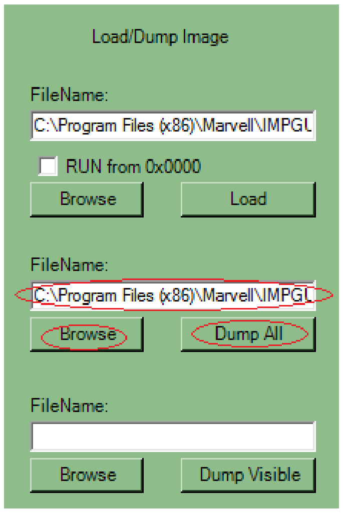
PIC8 show if check "RUN from 0x000", the "Load" button will be change to "Load&Run". When click this button after you select ".ihx" file, the GUI will do RESET and RUN function after the content be loaded into memory completely.



PIC 8

c. PIC9—PIC10 show the function of Dump Image”. It only support IHX format image and the suffix of the file must be “.ihx”, too. “Dump All” means dump whole memory (0x0000-0xefff) to file; it will take a long time. “Dump Visible” means dump “Memory Range” to a file.

Note: If you want to change dumped file and reload it, please change the checksum (The last byte of each line.) number accordingly, otherwise the load will be fail.



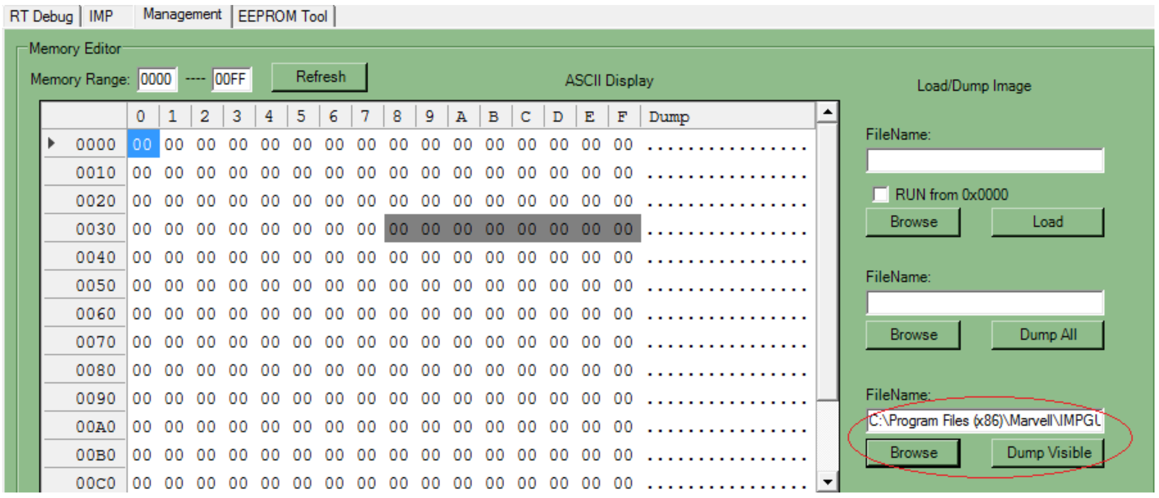
FileName: C:\Program Files (x86)\Marvell\IMPGU

☐ RUN from 0x0000

FileName: C:\Program Files (x86)\Marvell\IMPGU

FileName:

PIC 9



Memory Range: 0000 ---- 00FF

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  Dump
0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

FileName:

☐ RUN from 0x0000

FileName:

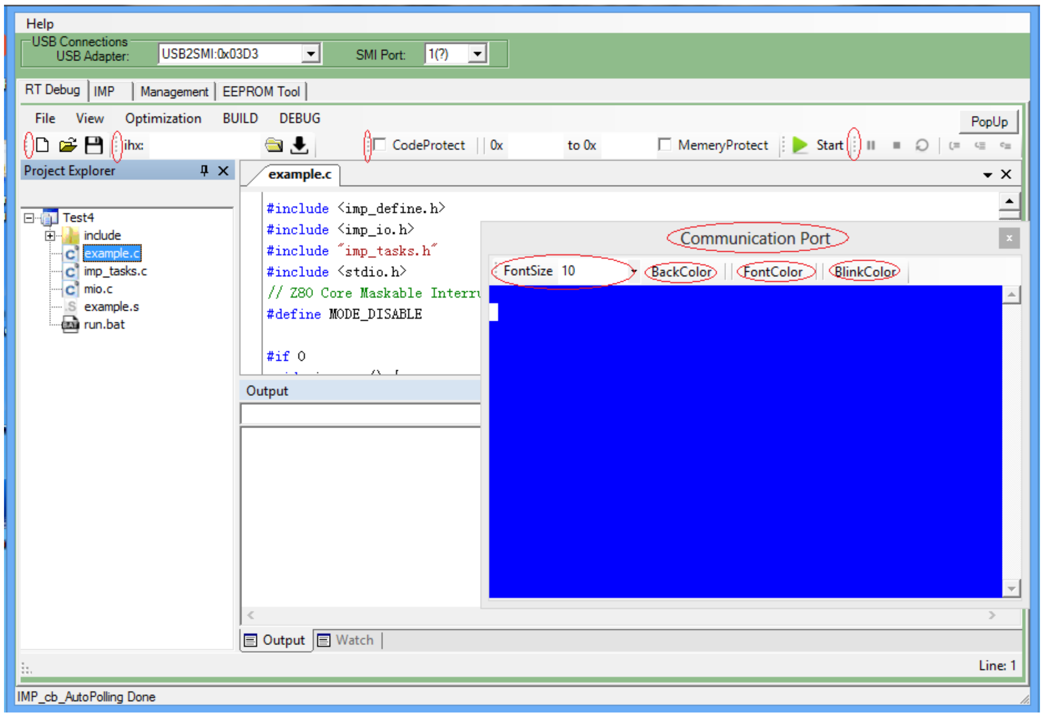
FileName: C:\Program Files (x86)\Marvell\IMPGU

PIC 10

3.3 RT Debug Page

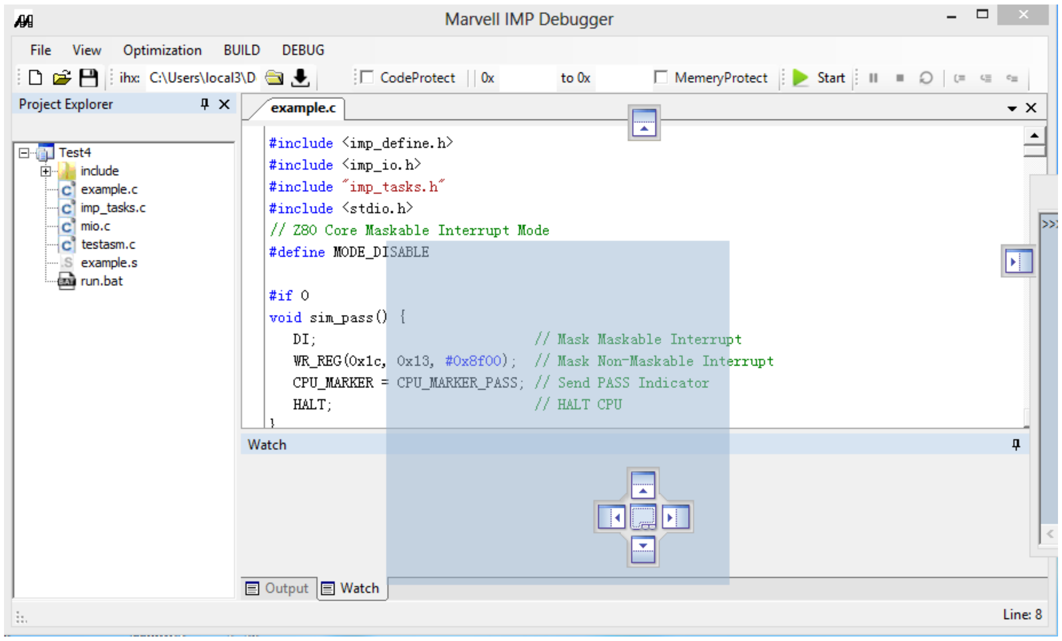
3.3.1 Overview

RT Debug is a very flexible GUI, almost all the item and windows can be dragged to anywhere as you like. On top right there is a “PopUp” button, click it the whole RT Debug GUI will be popped out. If you had closed some windows of the debugger, you can reopen it in the menu “View” item. See PIC11 and PIC12.



```
#include <imp_define.h>
#include <imp_io.h>
#include "imp_tasks.h"
#include <stdio.h>
// Z80 Core Maskable Interru
#define MODE_DISABLE

#if 0
```

PIC 11

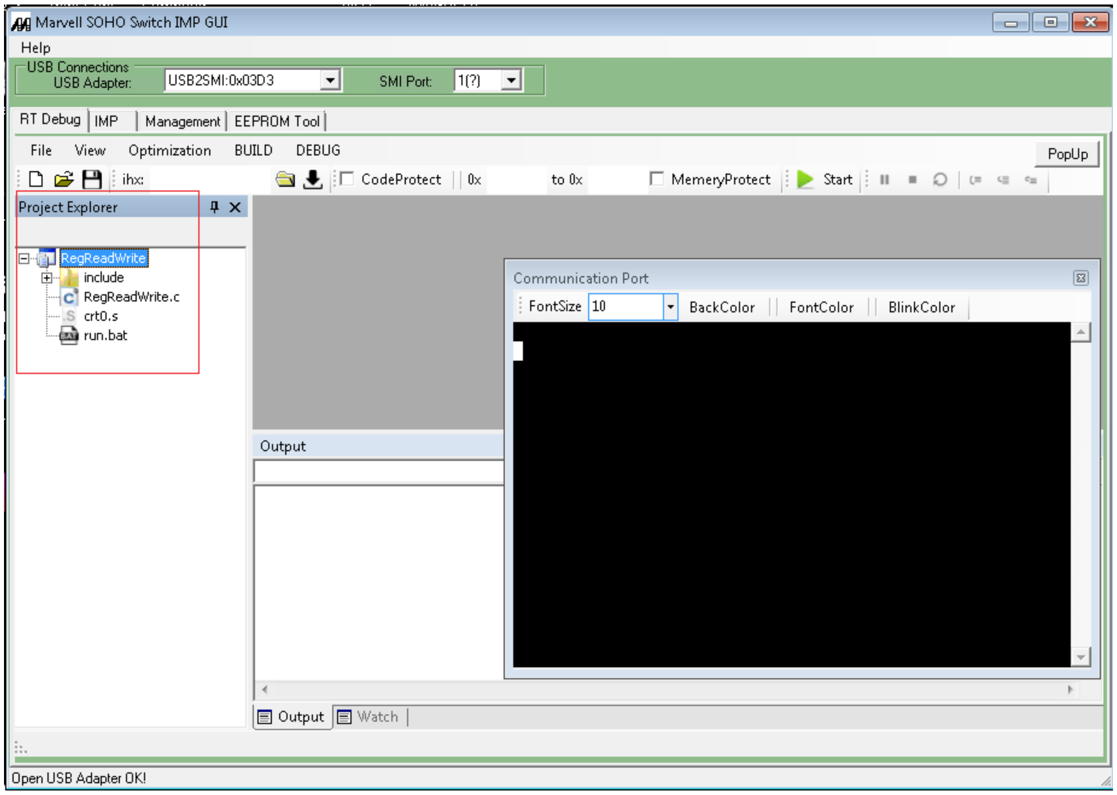


```
#include <imp_define.h>
#include <imp_io.h>
#include "imp_tasks.h"
#include <stdio.h>
// Z80 Core Maskable Interrupt Mode
#define MODE_DISABLE

#if 0
void sim_pass() {
    DI;                             // Mask Maskable Interrupt
    WR_REG(0x1c, 0x13, #0x8f00);  // Mask Non-Maskable Interrupt
    CPU_MARKER = CPU_MARKER_PASS; // Send PASS Indicator
    HALT;                           // HALT CPU
}
```

PIC 12

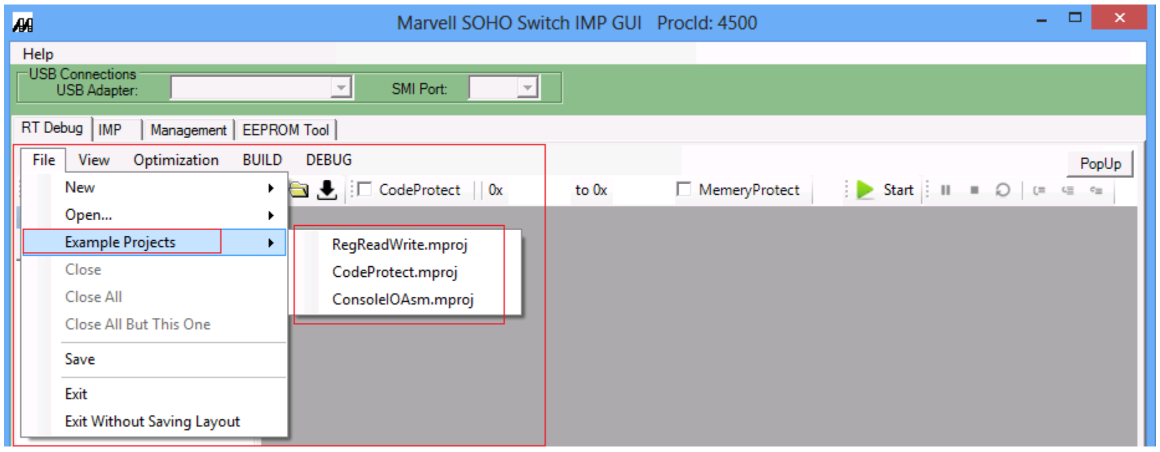
3.3.2 Mange project

When you open the IMP GUI, RT Debug tab is the initial tab and it will open the project of “RegReadWrite” by default as PIC13.



PIC 13

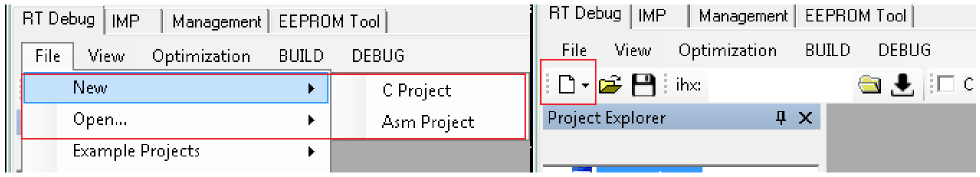
And we provide other sample projects for user as PIC14



PIC 14

Note: Project Explorer cannot rename the project. If you want to rename a project, you must to create a C project and save it into the folder.

The GUI provides two approaches to create C or Asm project as PIC15.

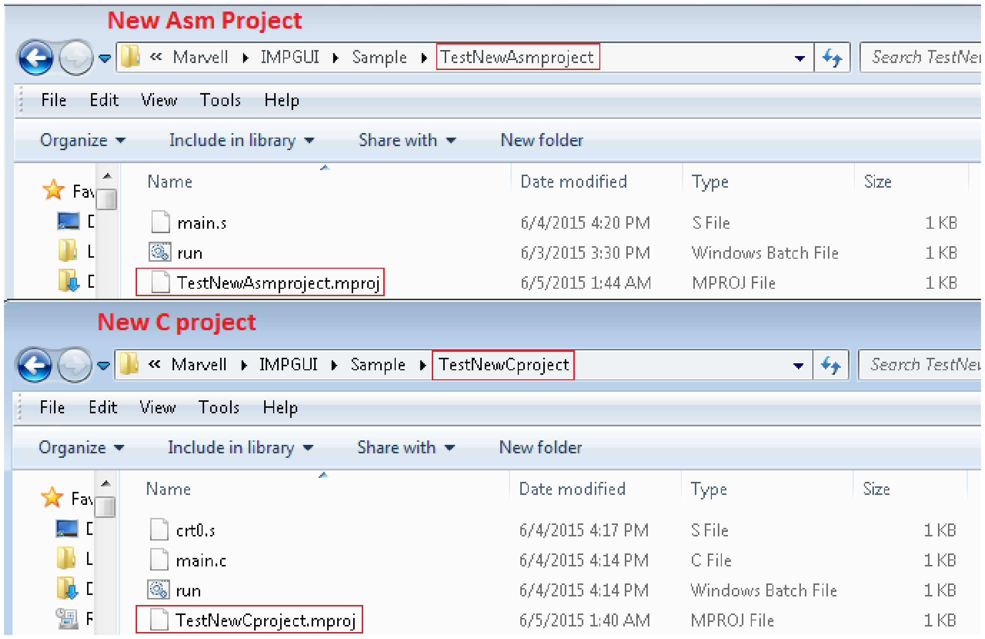


PIC 15

When you are creating a new C or Asm project:

1. If the target folder without related files, the IMPGUI will generate a new folder with all related files and the project file with the same name as PIC16.

   C project related files are xxxx.s, xxxx.c and run.bat
   Asm project related files are xxxx.s and run.bat

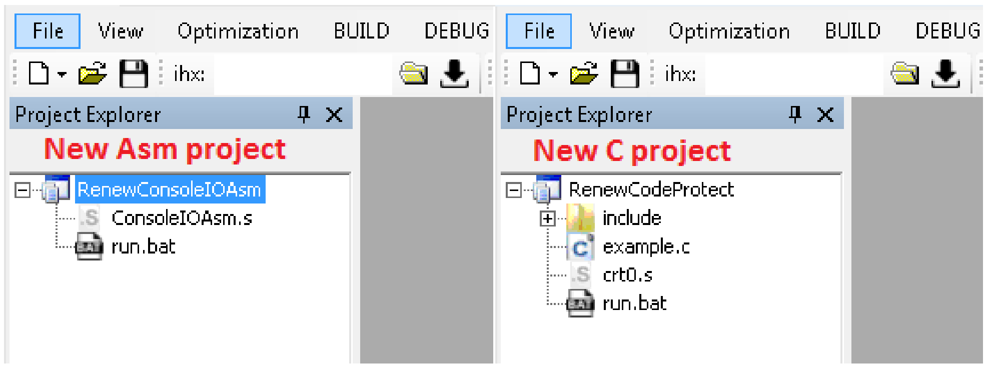


Name	Date modified	Type	Size
main.s	6/4/2015 4:20 PM	S File	1 KB
run	6/3/2015 3:30 PM	Windows Batch File	1 KB
TestNewAsmproject.mproj	6/5/2015 1:44 AM	MPROJ File	1 KB

Name	Date modified	Type	Size
crt0.s	6/4/2015 4:17 PM	S File	1 KB
main.c	6/4/2015 4:14 PM	C File	1 KB
run	6/4/2015 4:14 PM	Windows Batch File	1 KB
TestNewCproject.mproj	6/5/2015 1:40 AM	MPROJ File	1 KB

PIC 16

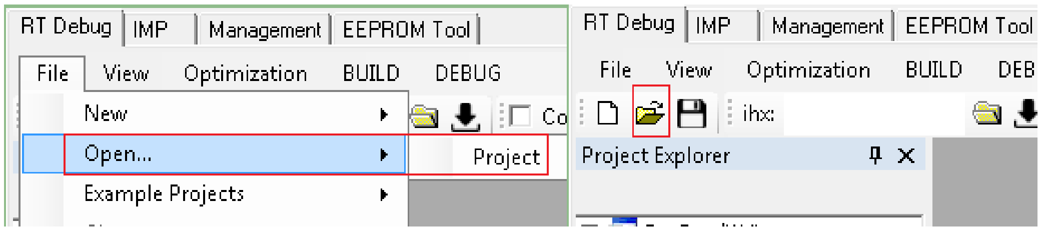
2. If the target folder has the related files that match our file creating spec, the IMPGUI will load the related files into the project as PIC17.



PIC 17

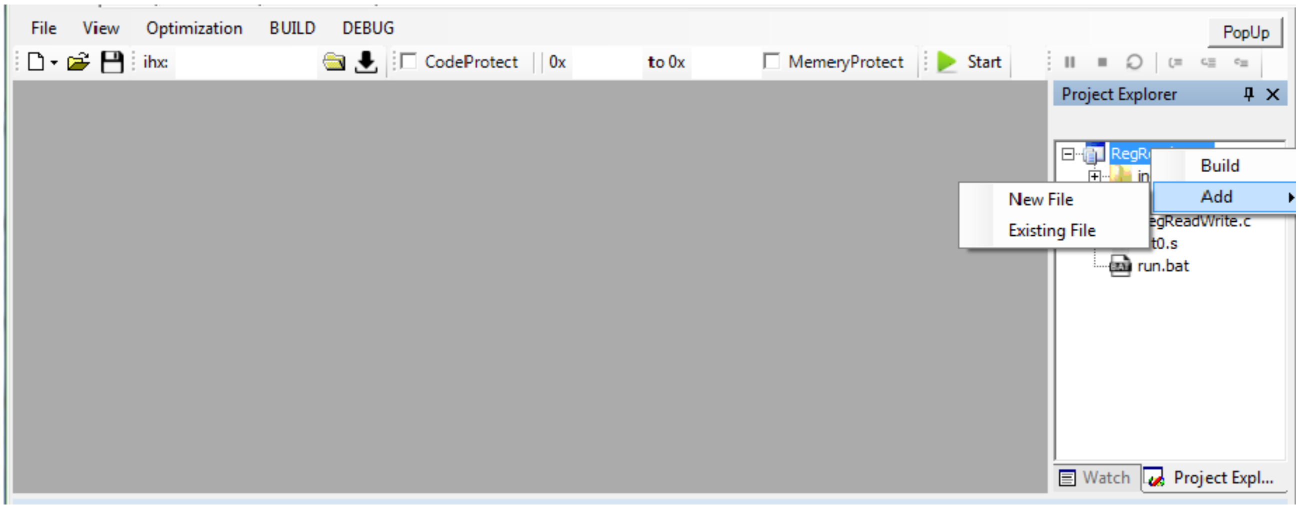
After you open another project, all existing documents in Editor Window will be closed.

The GUI provides two approaches to open project as well, one through file menu one through toolbar icon as PIC18



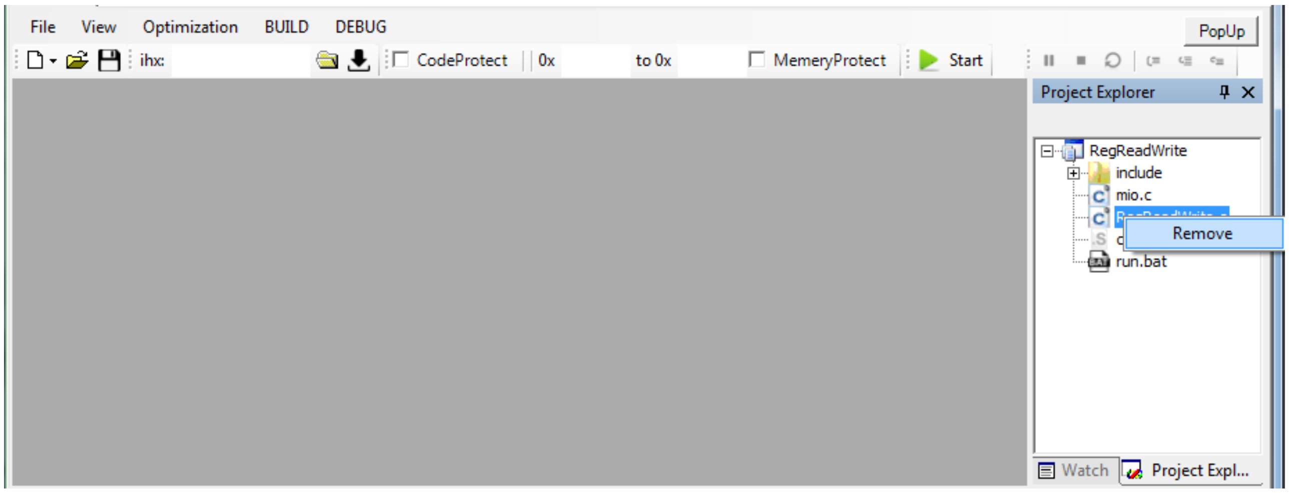
PIC 18

After load project finished, you can add new file or existing file as PIC19 and the file must in current project folder.



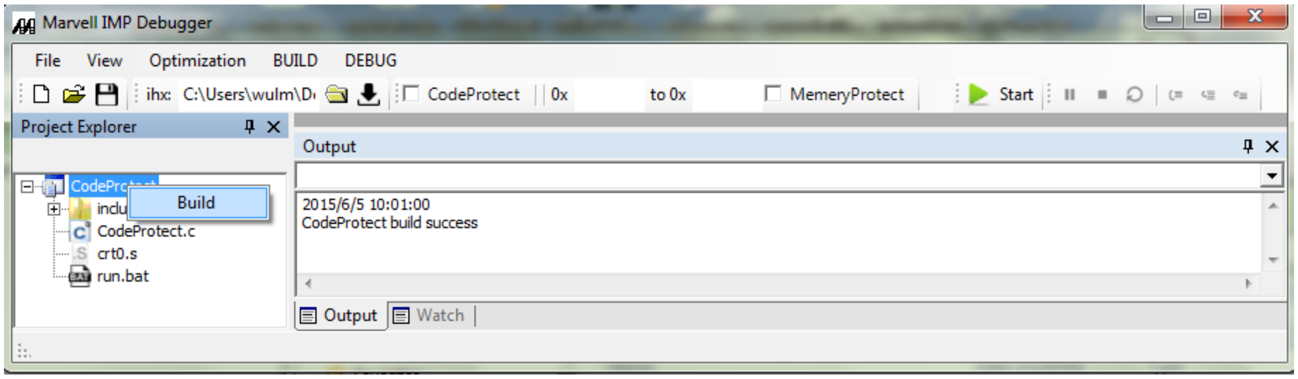
PIC 19

You can also remove the file from project as PIC20.



PIC 20

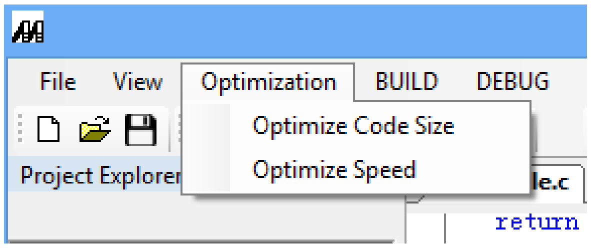
After you create or open a project, you need build it and the output console will display the build log as PIC21.



```
2015/6/5 10:01:00
CodeProtect build success
```

PIC 21

This version GUI provides two kinds of optimizing code methods as PIC22.



PIC 22

When you select “Optimize Code Size”, it will add line “set cflag=%cflag% --opt-code-size” in “run.bat”, while select “Optimize Speed”, it will add line “set cflag=%cflag% --opt-code-speed”.

The RT debugger requires a lot of built information, so the run.bat will disable most of optimization build options. For example: -Dint="volatile int" -Dchar="volatile char”, if you need any detail, please check the SDCC user manual.

3.3.3 Make file and crt0

Make file (run.bat) is following SDCC standard, for detailed information, you can see sddcman.pdf. *"run.bat" file name can't be changed.* For debugger use, we just want to reduce code optimization as much as possible; following is "EnableInterrupt" project's sample Make file we provide:

1. *set ComplPATH="C:\Program Files (x86)\Marvell\SDCC\bin\"*
2. *set ComplIncludePATH="C:\Program Files (x86)\Marvell\SDCC\include"*
3. *set cflag=-Dint="volatile int" -Dchar="volatile char" --debug --less-pedantic --fno-omit-frame-pointer --nooverlay --nogcse --noinduction --noinvariant --nojtbound --noloopreverse --nolabelopt --no-peep --no-peep-return --nolospre*
4. *%ComplPATH%sdasz80 -l -o -y .\crt0.rel .\ crt0.s*
5. *%ComplPATH%sdcc %cflag% -mz80 -c -I .\include I %ComplIncludePATH% .\EnableInterrupt.c*
6. *%ComplPATH%sdcc %cflag% -mz80 -c -I .\include -I %ComplIncludePATH% .\mio.c*
7. *%ComplPATH%sdcc --debug -mz80 --code-loc 0x200 --data-loc 0x0000 --no-std-crt0 -I .\include --verbose -o EnableInterrupt.ihx .\mio.rel .\EnableInterrupt.rel .\crt0.rel*Line1 and 2 are complier path setting;

line 3 are debug parameters, for example, "Dint" and "Dchar" are reduce "int" and "char" type variables optimization, for example, if you use "int" and "char" in your source file, you must specify like this in make file, otherwise you can't debug on variables with " int" and "char" type. For detailed information, please see sdccman.pdf;

Line 4 is compiling crt0 file, from line8, we can see "—no-std-crt0", it means we don't use the standard crt0, so "crt0.s" is a customized crt0;

Line5-line6 is compiling other related files;

Line7 is the final linker clause, it will specify the code locate address "--code-loc 0x200" and the IHX file name.

Following is part of sample "crt0.s" file:

```
  .module crt0
 .globl     _main
  .globl  _isr
  .globl  _nmi_isr
  .area     _HEADER (ABS)
  ;; Reset vector
  .org      0H0000
  jp        init
 .org    0H0028
  ret
 .org    0H0030
  ret
 .org       0H040
```

```
    .dw _isr
     .org       0H0066
     call       _nmi_isr
     reti
     .org       0H0080
init:
```

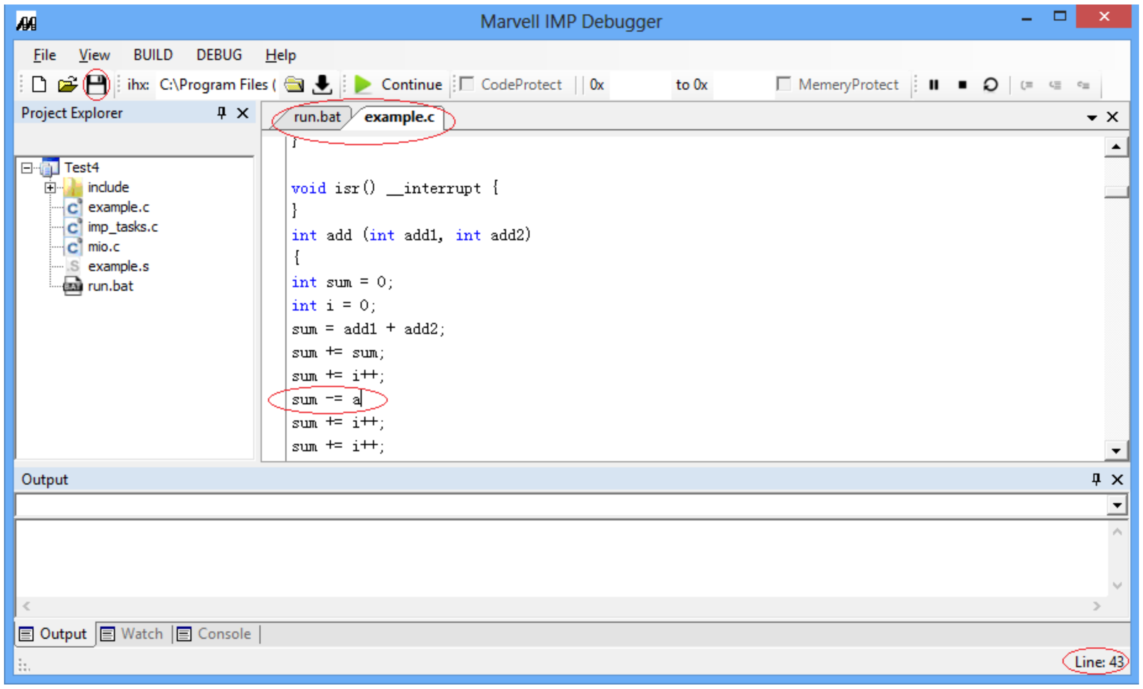
Above file will specify the code start address.

Note: 1. Code start should be specified in "example.s" file, and the code locate address should change in "rnu.bat" ("--code-loc 0x200") file accordingly to a reasonable value.

2. Add '-l' parameter in "run.bat" for assembly files and there will be one ".lst" file after your build. In this file you can find the listing of the assembly code with the Address & Data. Detail information can be found in SDCC document. For example, the "run.bat" in sample test5, "sdasz80 -l -o .\consol.rel .\consol.s".

3.3.4 Edit and save source files

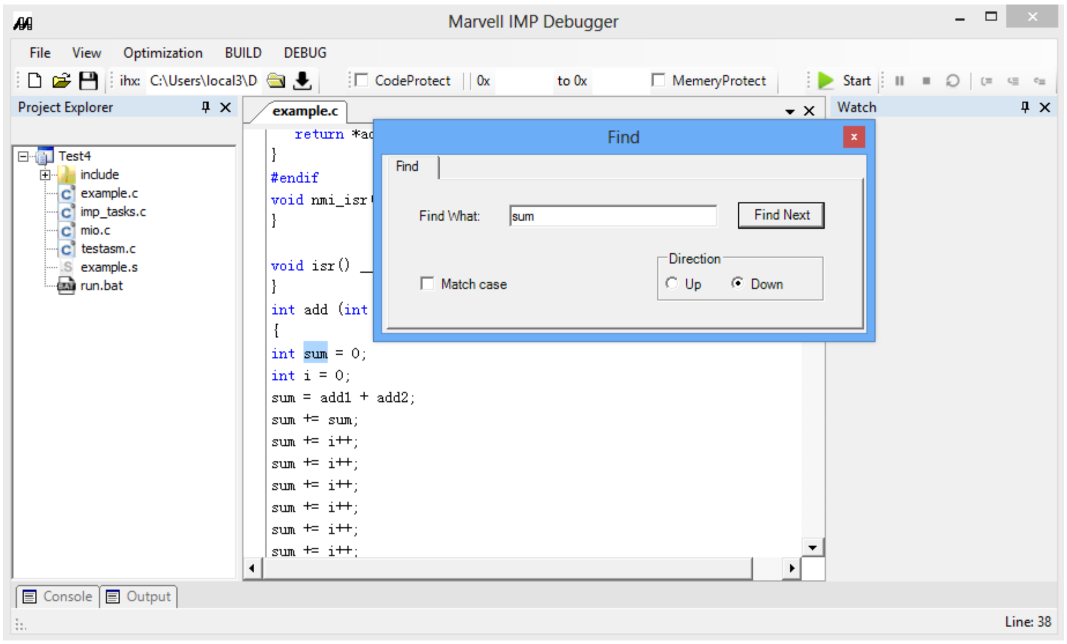
This version GUI provides editing and saving files, you can use "save" button and "ctrl + s" to save a file. On bottom right of the GUI will display the edited line number. This debugger can support multiple files editing. Please see PIC 23.



```
}

void isr() __interrupt {
}
int add (int add1, int add2)
{
int sum = 0;
int i = 0;
sum = add1 + add2;
sum += sum;
sum += i++;
sum -= a
sum += i++;
sum += i++;
```

PIC 23

It also support “Find” function, please see PIC24.



Find What: sum

[ ] Match case

Direction: Up [✓ Down]

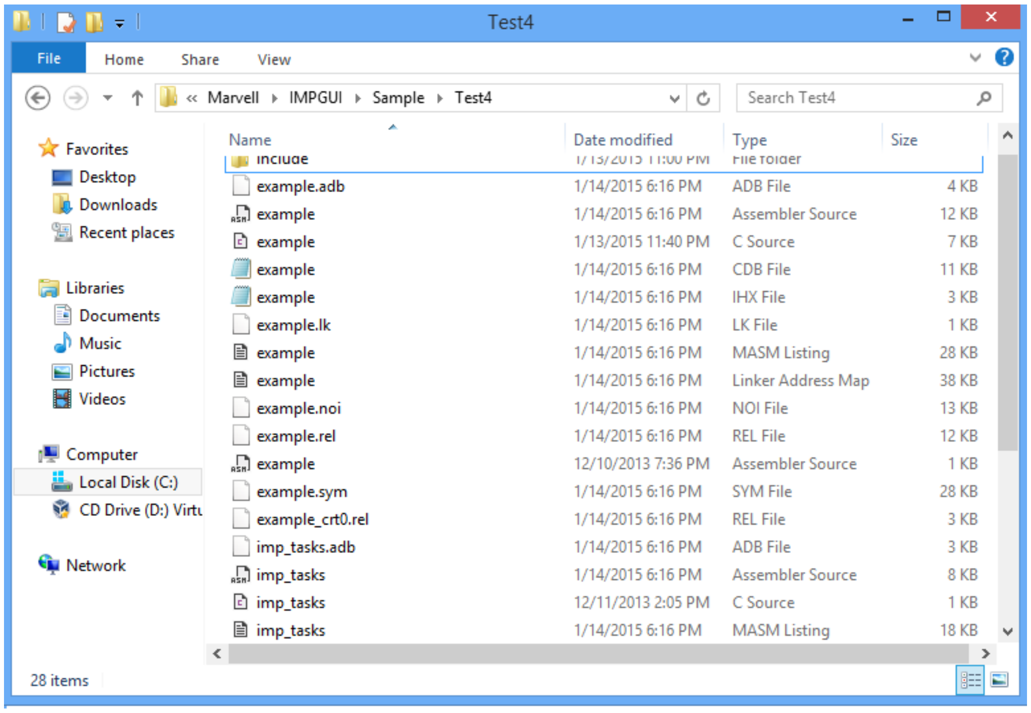
```
   return *ad
}
#endif
void nmi_isr(
}

void isr() _
}
int add (int
{
int sum = 0;
int i = 0;
sum = add1 + add2;
sum += sum;
sum += i++;
sum += i++;
sum += i++;
sum += i++;
sum += i++;
sum += i++;
```

PIC 24

3.3.5 Load IHX file

After the project has been build successfully, it will generate ".ihx", ".cdb", ".map" files and so on. You should select the ".ihx" file and load it, and the output console will display the load status. See PIC25 and PIC26.

Note: Before real time debugging, the IHX file must be loaded before.



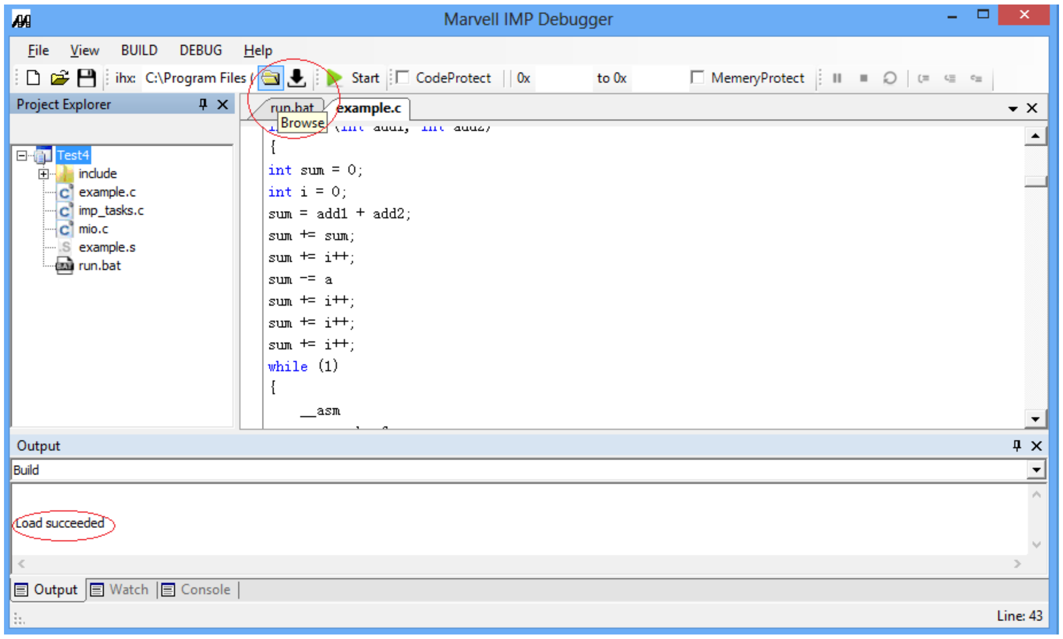
Test4

« Marvell ▸ IMPGUI ▸ Sample ▸ Test4

Name	Date modified	Type	Size
include	1/13/2015 11:00 PM	File folder	
example.adb	1/14/2015 6:16 PM	ADB File	4 KB
example	1/14/2015 6:16 PM	Assembler Source	12 KB
example	1/13/2015 11:40 PM	C Source	7 KB
example	1/14/2015 6:16 PM	CDB File	11 KB
example	1/14/2015 6:16 PM	IHX File	3 KB
example.lk	1/14/2015 6:16 PM	LK File	1 KB
example	1/14/2015 6:16 PM	MASM Listing	28 KB
example	1/14/2015 6:16 PM	Linker Address Map	38 KB
example.noi	1/14/2015 6:16 PM	NOI File	13 KB
example.rel	1/14/2015 6:16 PM	REL File	12 KB
example	12/10/2013 7:36 PM	Assembler Source	1 KB
example.sym	1/14/2015 6:16 PM	SYM File	28 KB
example_crt0.rel	1/14/2015 6:16 PM	REL File	3 KB
imp_tasks.adb	1/14/2015 6:16 PM	ADB File	3 KB
imp_tasks	1/14/2015 6:16 PM	Assembler Source	8 KB
imp_tasks	12/11/2013 2:05 PM	C Source	1 KB
imp_tasks	1/14/2015 6:16 PM	MASM Listing	18 KB

28 items

PIC 25



Marvell IMP Debugger

ihx: C:\Program Files

```
                                    (int add1, int add2)
{
int sum = 0;
int i = 0;
sum = add1 + add2;
sum += sum;
sum += i++;
sum -= a
sum += i++;
sum += i++;
sum += i++;
while (1)
{
        __asm
```

Output

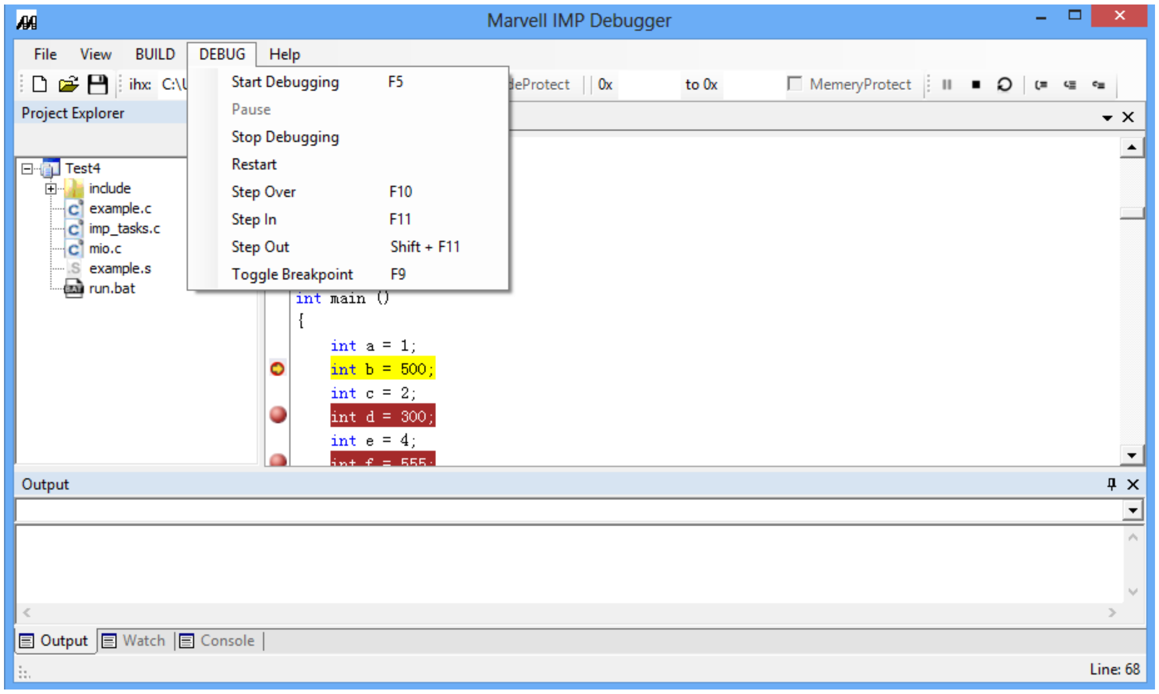
Build

```
Load succeeded
```

PIC 26

3.3.6 Software breakpoints

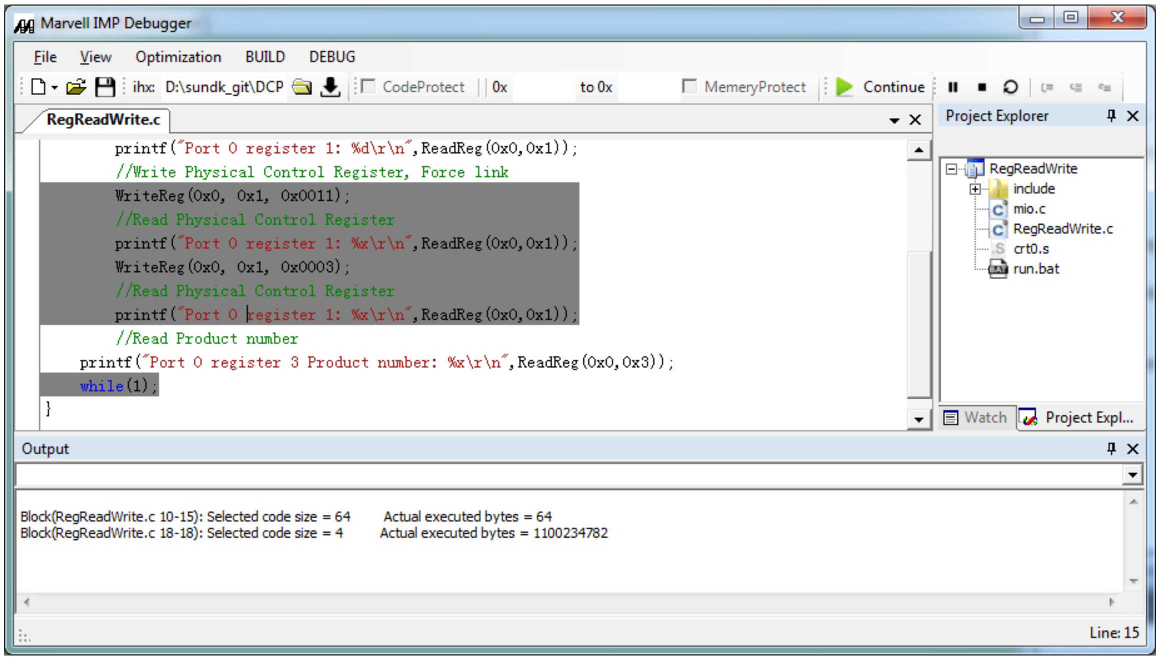
This debugger will provide infinite software breakpoints, you can click the front of the line or press hotkey "F9" to toggle and remove breakpoints. See PIC27.



```
int main ()
{
    int a = 1;
    int b = 500;
    int c = 2;
    int d = 300;
    int e = 4;
    int f = 555;
```

PIC 27

And the GUI support to set one to three range breakpoints as PIC28



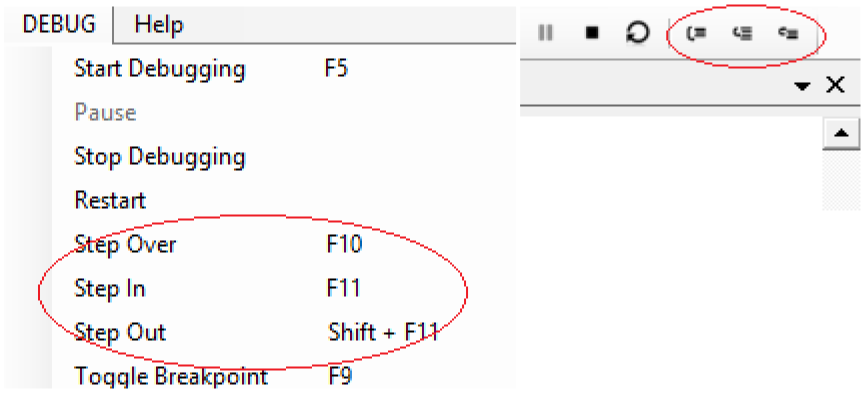
```
        printf("Port 0 register 1: %d\r\n",ReadReg(0x0,0x1));
        //Write Physical Control Register, Force link
        WriteReg(0x0, 0x1, 0x0011);
        //Read Physical Control Register
        printf("Port 0 register 1: %x\r\n",ReadReg(0x0,0x1));
        WriteReg(0x0, 0x1, 0x0003);
        //Read Physical Control Register
        printf("Port 0 register 1: %x\r\n",ReadReg(0x0,0x1));
        //Read Product number
    printf("Port 0 register 3 Product number: %x\r\n",ReadReg(0x0,0x3));
    while(1);
}
```

```
Block(RegReadWrite.c 10-15): Selected code size = 64     Actual executed bytes = 64
Block(RegReadWrite.c 18-18): Selected code size = 4     Actual executed bytes = 1100234782
```

PIC 28

3.3.7 Step, step in and step out

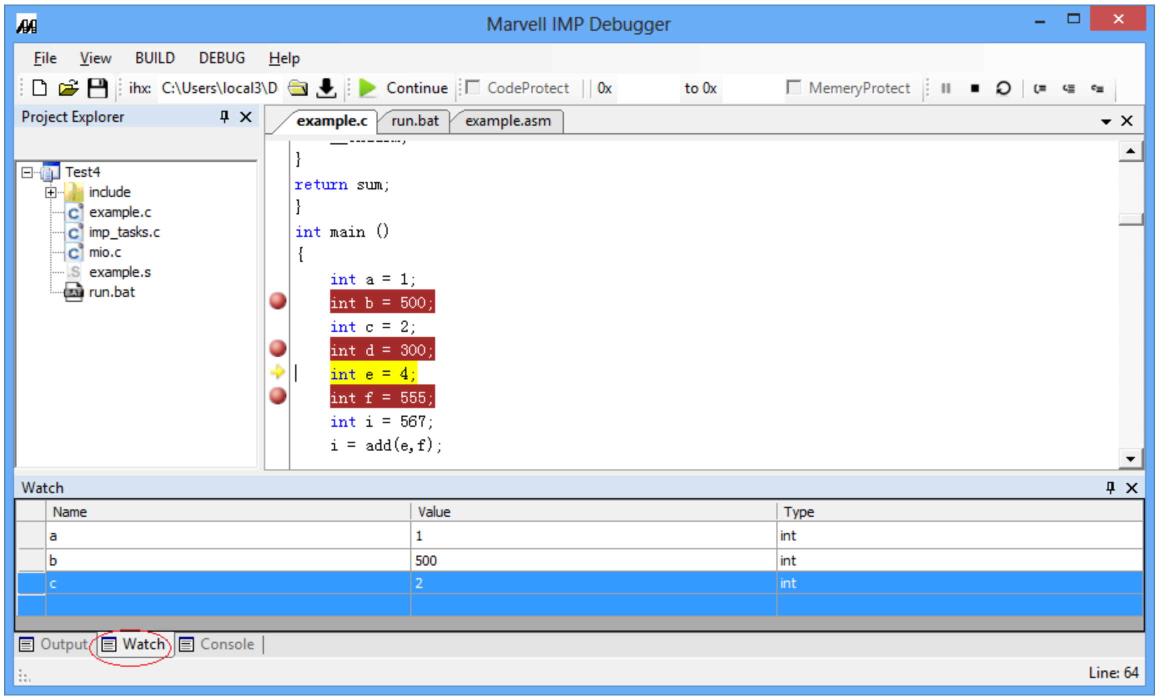
The functions “Step”, “Step in” and “Step out” is the same as visual studio. See PIC29.



PIC 29

3.3.8 Watch variables

In C code source level debugger, you can add variables to watch, so far, it only support “int”, “unsigned int”, “char”, “unsigned char”, “ float”, “short”, “unsigned short”, “long32” and “unsigned long32” variables. Watch console will display the variables real time. See PIC30.



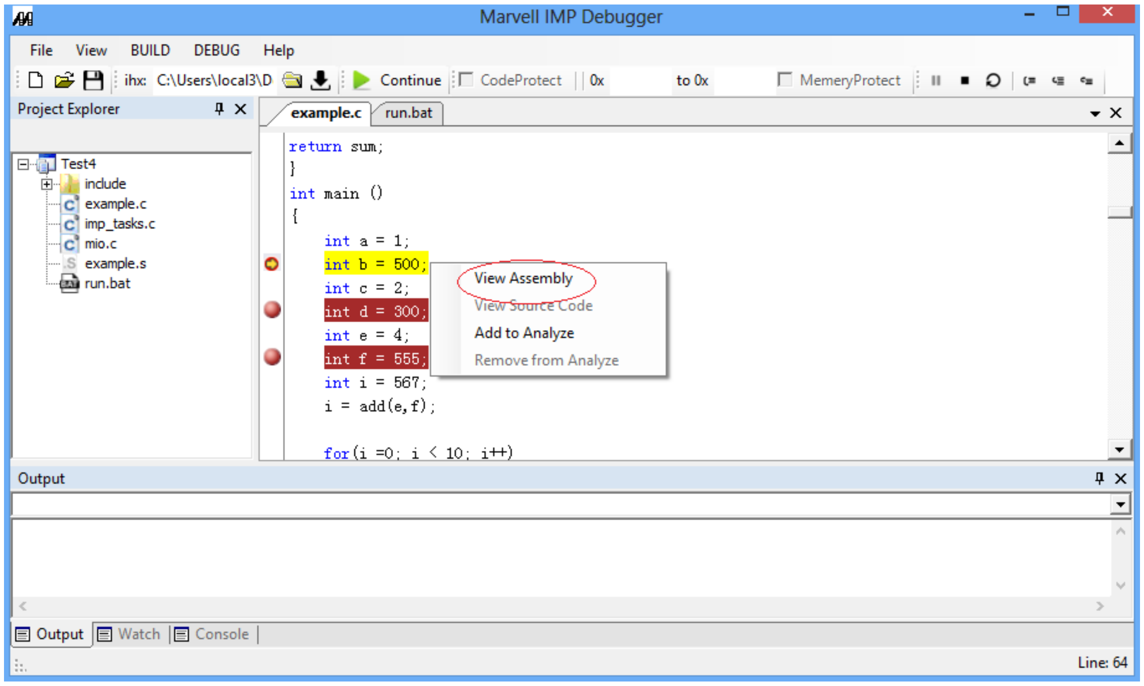
```
}
return sum;
}
int main ()
{
    int a = 1;
    int b = 500;
    int c = 2;
    int d = 300;
    int e = 4;
    int f = 555;
    int i = 567;
    i = add(e,f);
```

Name	Value	Type
a	1	int
b	500	int
c	2	int

PIC30

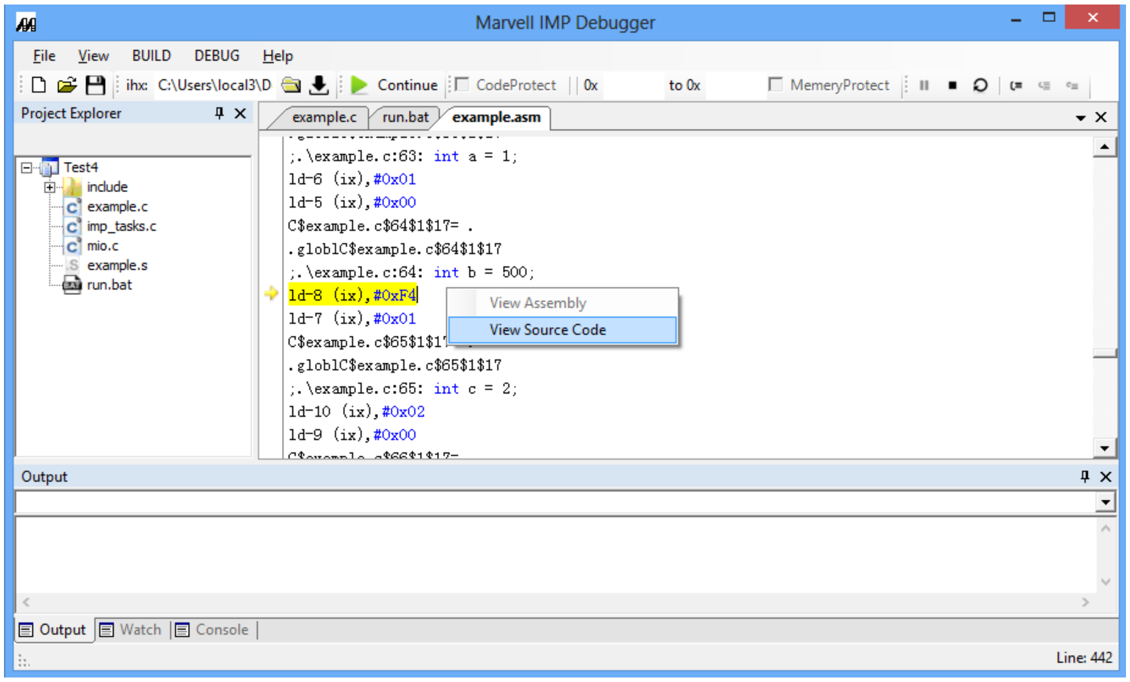
3.3.9 Assembly files source level debugger

In C code zone (Only supporting from C to assembly), right mouse click → “View Assembly”, it will switch to assembly file. In assembly file, you can use “Step” only, right mouse click →”View Source Code” back to C code. See PIC31 and PIC32.



```
return sum;
}
int main ()
{
    int a = 1;
    int b = 500;
    int c = 2;
    int d = 300;
    int e = 4;
    int f = 555;
    int i = 567;
    i = add(e,f);

    for(i =0; i < 10; i++)
```

PIC 31



```
;.\example.c:63: int a = 1;
ld-6 (ix),#0x01
ld-5 (ix),#0x00
C$example.c$64$1$17= .
.globlC$example.c$64$1$17
;.\example.c:64: int b = 500;
ld-8 (ix),#0xF4
ld-7 (ix),#0x01
C$example.c$65$1$17
.globlC$example.c$65$1$17
;.\example.c:65: int c = 2;
ld-10 (ix),#0x02
ld-9 (ix),#0x00
```

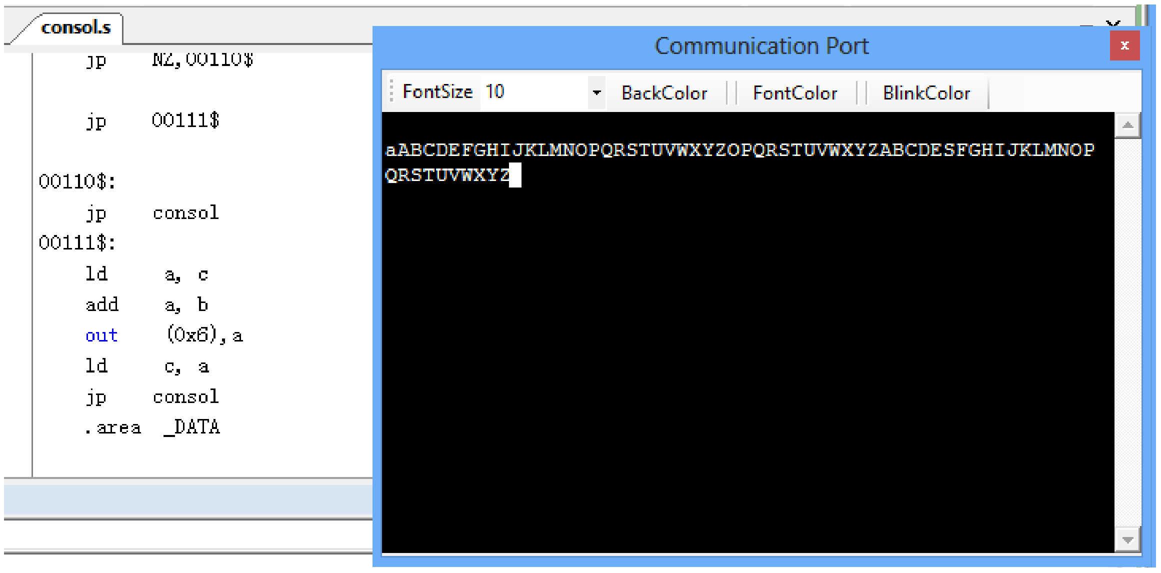
PIC 32

3.3.10 Communicate port console

This feature is designed by software, so for supporting this feature, you should add the related code (Refer to sample project "ConsoleIOAsm"). It is basically a terminal emulation mode; it supports communicating with Z80, the console will display the output of the z80 and key hits in this console will go to Z80.

➢ Console display

PIC33 shows the console will display Z80 output when CPU is running (Start running from "RT Debug" or "IMP" page). For detailed information, please refer to sample "ConsoleIOAsm".



```
    jp      NZ,00110$

    jp      00111$

00110$:
    jp      consol
00111$:
    ld       a, c
    add      a, b
    out      (0x6),a
    ld       c, a
    jp      consol
    .area   _DATA
```

```
aABCDEFGHIJKLMNOPQRSTUVWXYZOPQRSTUVWXYZABCDESFGHIJKLMNOP
QRSTUVWXYZ
```

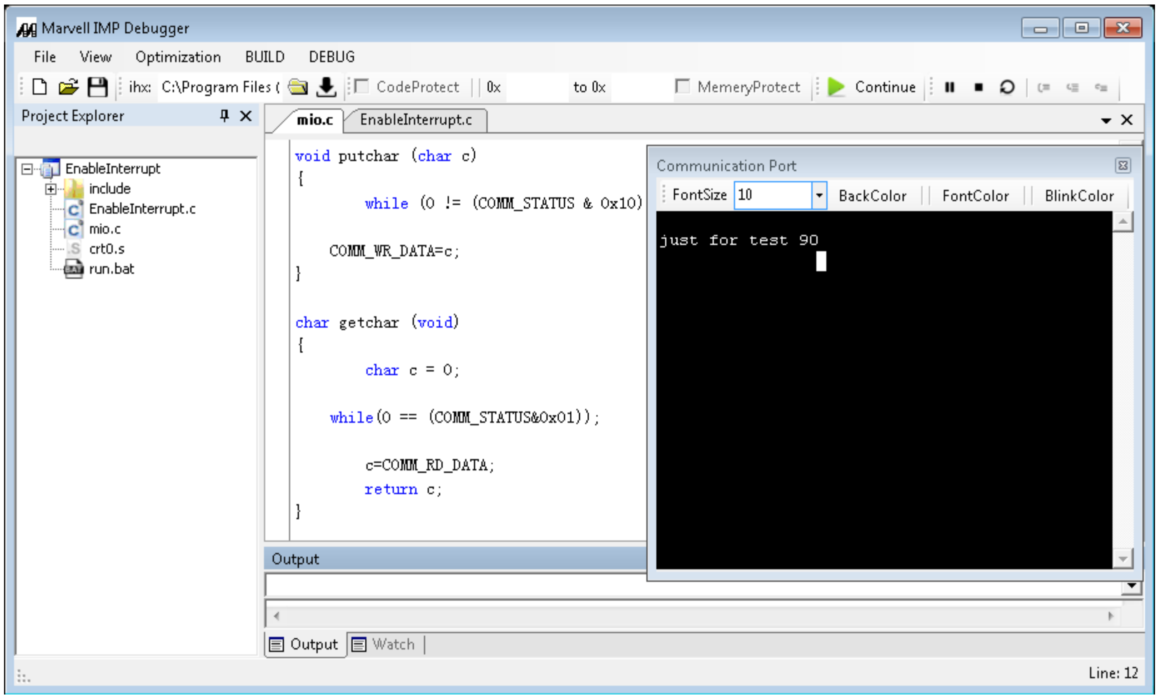
PIC 33

➢ Console input

Following are the related codes for supporting this feature:

```
void putchar (char c)
{
        while (0 != (COMM_STATUS & 0x10));

  COMM_WR_DATA=c;
}

char getchar (void)
{
        char c = 0;

  while(0 == (COMM_STATUS&0x01));

        c=COMM_RD_DATA;
        return c;
}
```

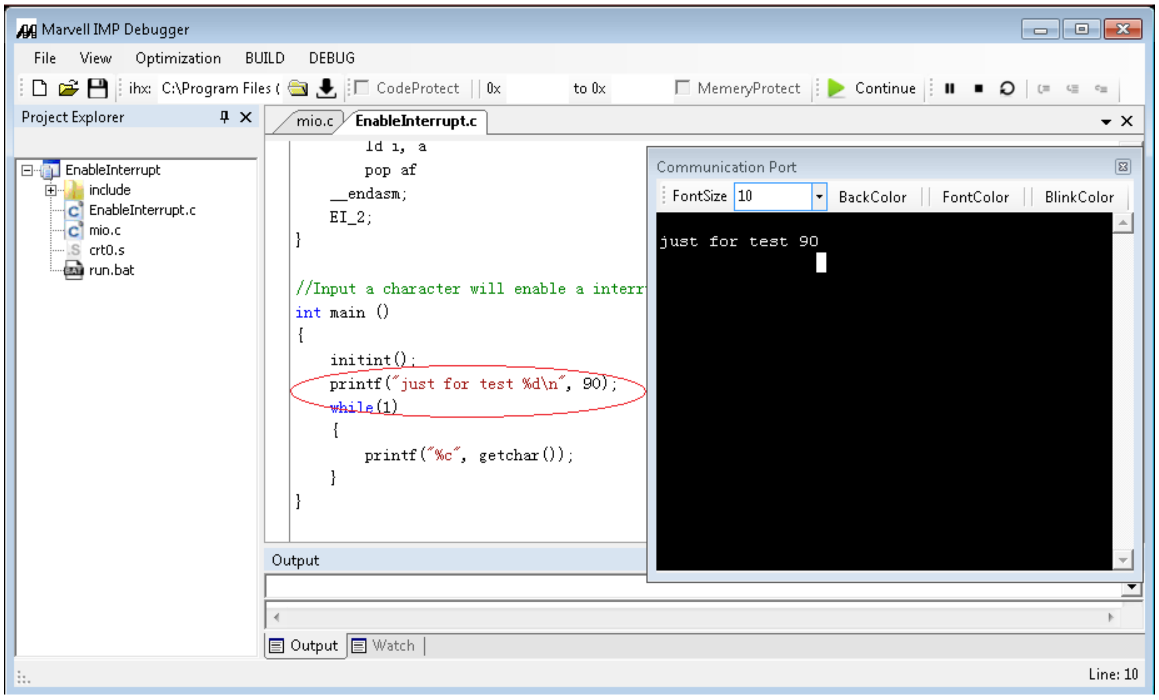
This part codes have been implemented in sample project “EnableInterrupt”, and the console can drag to anywhere. See PIC34 and PIC35.



```
void putchar (char c)
{
        while (0 != (COMM_STATUS & 0x10)

    COMM_WR_DATA=c;
}

char getchar (void)
{
        char c = 0;

    while(0 == (COMM_STATUS&0x01));

        c=COMM_RD_DATA;
        return c;
}
```

```
just for test 90
```

PIC 34



```
        ld l, a
        pop af
    __endasm;
    EI_2;
}

//Input a character will enable a interr
int main ()
{
    initint();
    printf("just for test %d\n", 90);
    while(1)
    {
        printf("%c", getchar());
    }
}
```

```
just for test 90
```

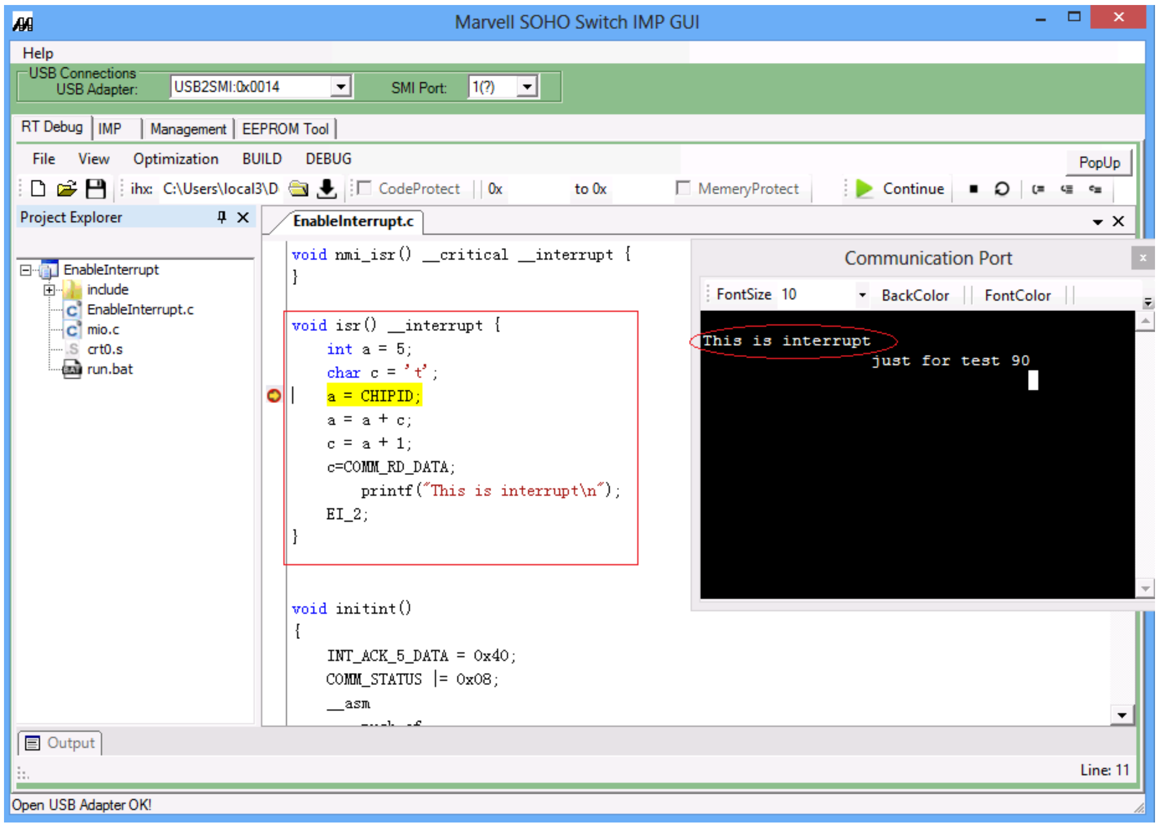
PIC 35

➢ Interrupt

Following interrupt code has been included in sample project “EnableInterrupt”

```
void isr() __interrupt {
   int a = 5;
   char c = 't';
   a = CHIPID;
   a = a + c;
   c = a + 1;
   c=COMM_RD_DATA;
          printf("This is interrupt\n");
   EI_2;
}
```

When you run this project as PIC35, if you input a character into console window, the message will be printed as PIC36.



USB Adapter: USB2SMI:0x0014 SMI Port: 1(?)

ihx: C:\Users\local3\D 0x to 0x

FontSize 10

```
void nmi_isr() __critical __interrupt {
}

void isr() __interrupt {
    int a = 5;
    char c = 't';
    a = CHIPID;
    a = a + c;
    c = a + 1;
    c=COMM_RD_DATA;
        printf("This is interrupt\n");
    EI_2;
}


void initint()
{
    INT_ACK_5_DATA = 0x40;
    COMM_STATUS |= 0x08;
    __asm
```

```
This is interrupt
                    just for test 90
```

Line: 11

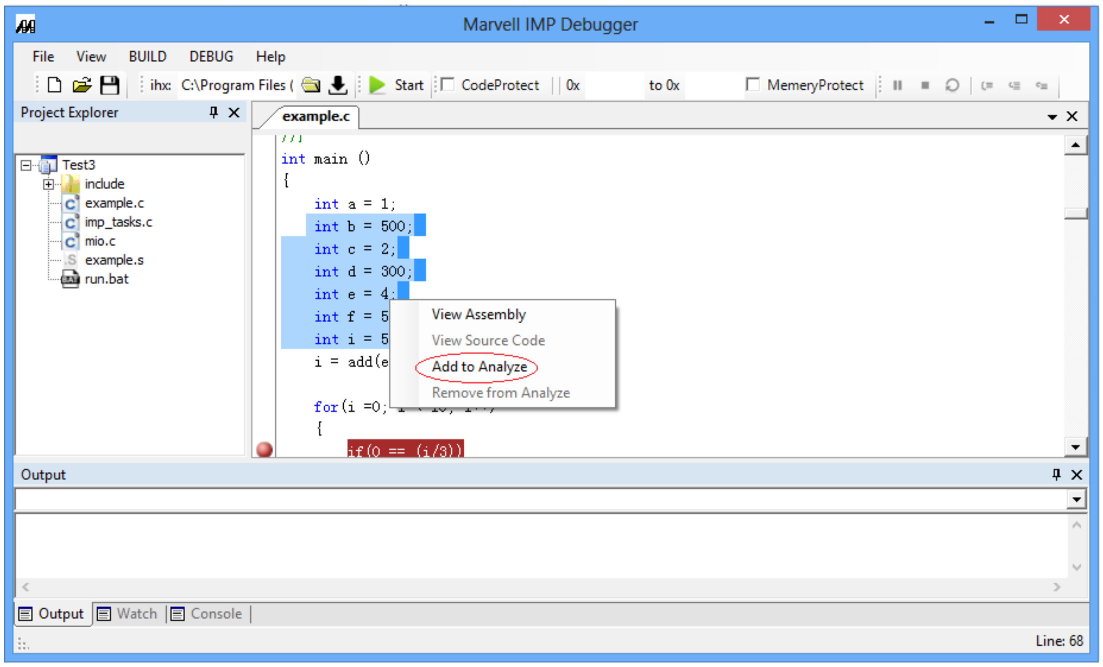
Open USB Adapter OK!

PIC 36

3.3.11 Code-running-counter

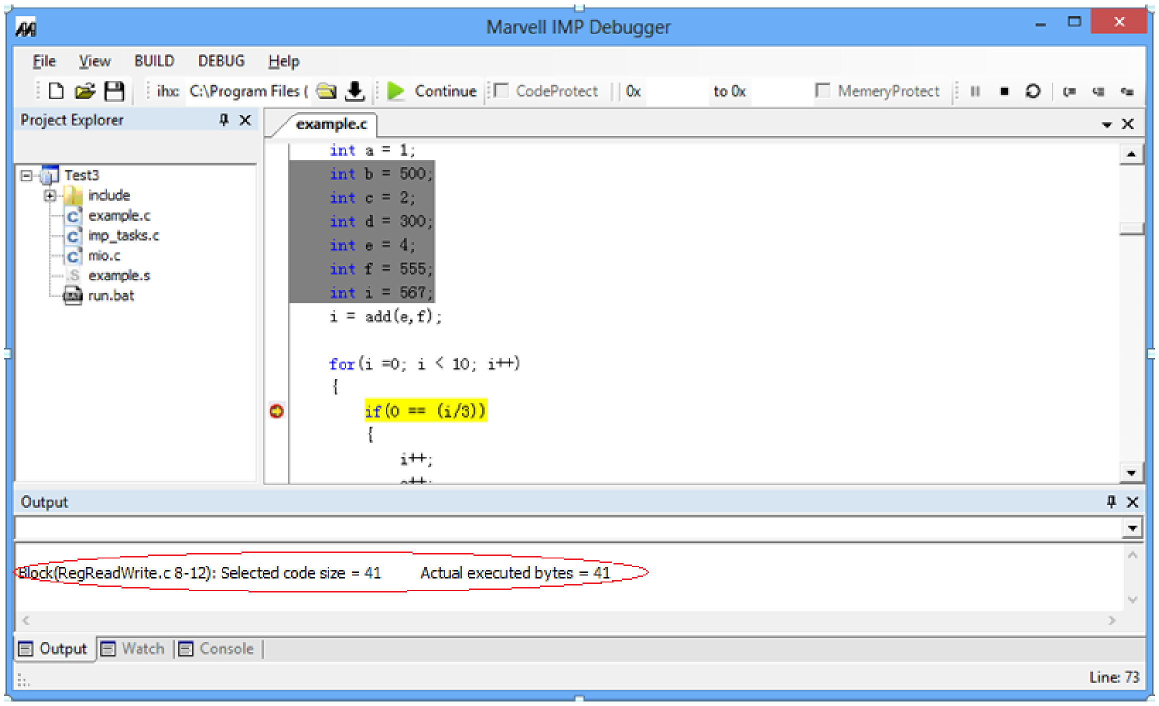
This feature based on range breakpoints, the breakpoint doesn’t monitor writing or reading. You can select a range of C codes and right mouse click, add to analyze, and then click “Run”, you can see messages in “output console”, the messages only display when CPU is pausing or stopping at breakpoints. See PIC37 and PIC38.

Selected code size in PIC38 means the C code range according to assembly code range, and count means assembly step.



```
int main ()
{
    int a = 1;
    int b = 500;
    int c = 2;
    int d = 300;
    int e = 4;
    int f = 5
    int i = 5
    i = add(e

    for(i =0;
    {
        if(0 == (i/3))
```

PIC 37



```
    int a = 1;
    int b = 500;
    int c = 2;
    int d = 300;
    int e = 4;
    int f = 555;
    int i = 567;
    i = add(e,f);

    for(i =0; i < 10; i++)
    {
        if(0 == (i/3))
        {
            i++;
```

```
Block(RegReadWrite.c 8-12): Selected code size = 41     Actual executed bytes = 41
```

PIC 38

3.3.12 Memory protect

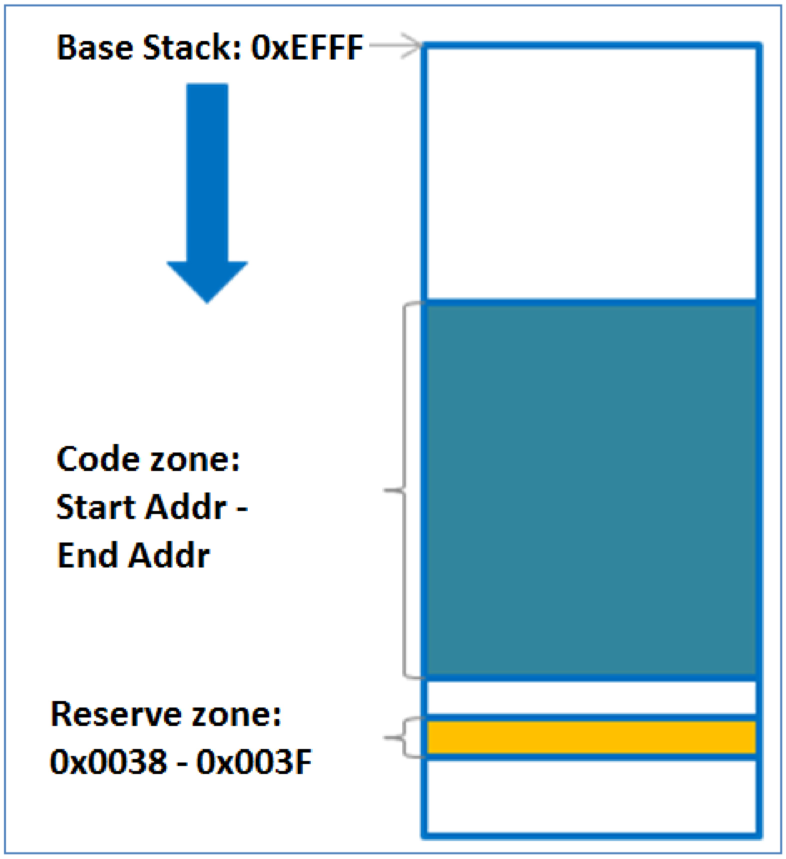
This feature also based on range breakpoints, the breakpoint monitor writing only. There is two kinds of memory protect: a. Code Protect, if you select this feature, it will protect your whole code zone in memory; b.Memory protect, you should input the start address and end address before enable this feature, it will protect the memory range you have input. Of course, you can select the both feature at the same time.

Because both memories protect and code-running-counter will take hardware breakpoints, so those features will influence each other, please see table1.

Table 1

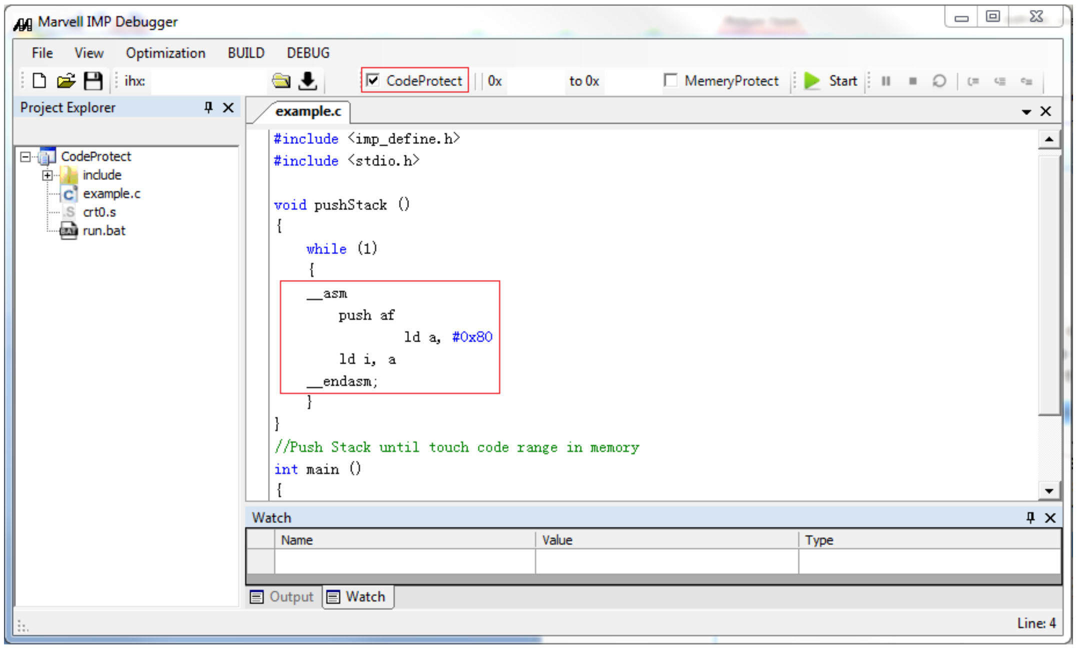
Code Protect	Memory Protect	Code-running-counter numbers
×	×	3
×	√	2
√	×	2
√	√	1

PIC39 will show the memory map.



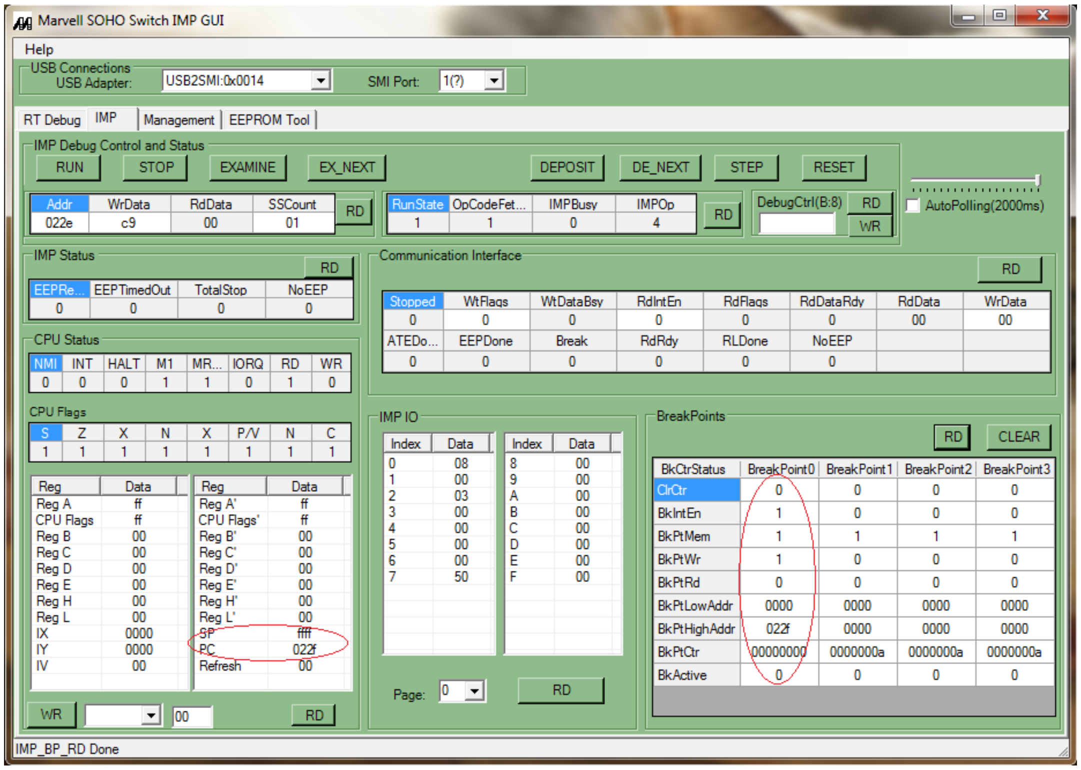
PIC 39

Sample project "CodeProtect" will push stack continue, it will touch code zone, and the CPU will stop. PIC40 shows load image successfully and select **CodeProtect**, then switch to IMP page PIC41, we can see the breakpoint have been set in hardware.



```
#include <imp_define.h>
#include <stdio.h>

void pushStack ()
{
    while (1)
    {
    __asm
        push af
                ld a, #0x80
        ld i, a
    __endasm;
    }
}
//Push Stack until touch code range in memory
int main ()
{
```

PIC 40



Addr	WrData	RdData	SSCount
022e	c9	00	01

RunState	OpCodeFet...	IMPBusy	IMPOp
1	1	0	4

IMP Status

EEPRe...	EEPTimedOut	TotalStop	NoEEP
0	0	0	0

CPU Status

NMI	INT	HALT	M1	MR...	IORQ	RD	WR
0	0	0	1	1	0	1	0

CPU Flags

S	Z	X	N	X	P/V	N	C
1	1	1	1	1	1	1	1

Reg	Data
Reg A	ff
CPU Flags	ff
Reg B	00
Reg C	00
Reg D	00
Reg E	00
Reg H	00
Reg L	00
IX	0000
IY	0000
IV	00

Reg	Data
Reg A'	ff
CPU Flags'	ff
Reg B'	00
Reg C'	00
Reg D'	00
Reg E'	00
Reg H'	00
Reg L'	00
SP	ffff
PC	022f
Refresh	00

Communication Interface

Stopped	WtFlags	WtDataBsy	RdIntEn	RdFlags	RdDataRdy	RdData	WrData
0	0	0	0	0	0	00	00
ATEDo...	EEPDone	Break	RdRdy	RLDone	NoEEP		
0	0	0	0	0	0		

IMP IO

Index	Data
0	08
1	00
2	03
3	00
4	00
5	00
6	00
7	50

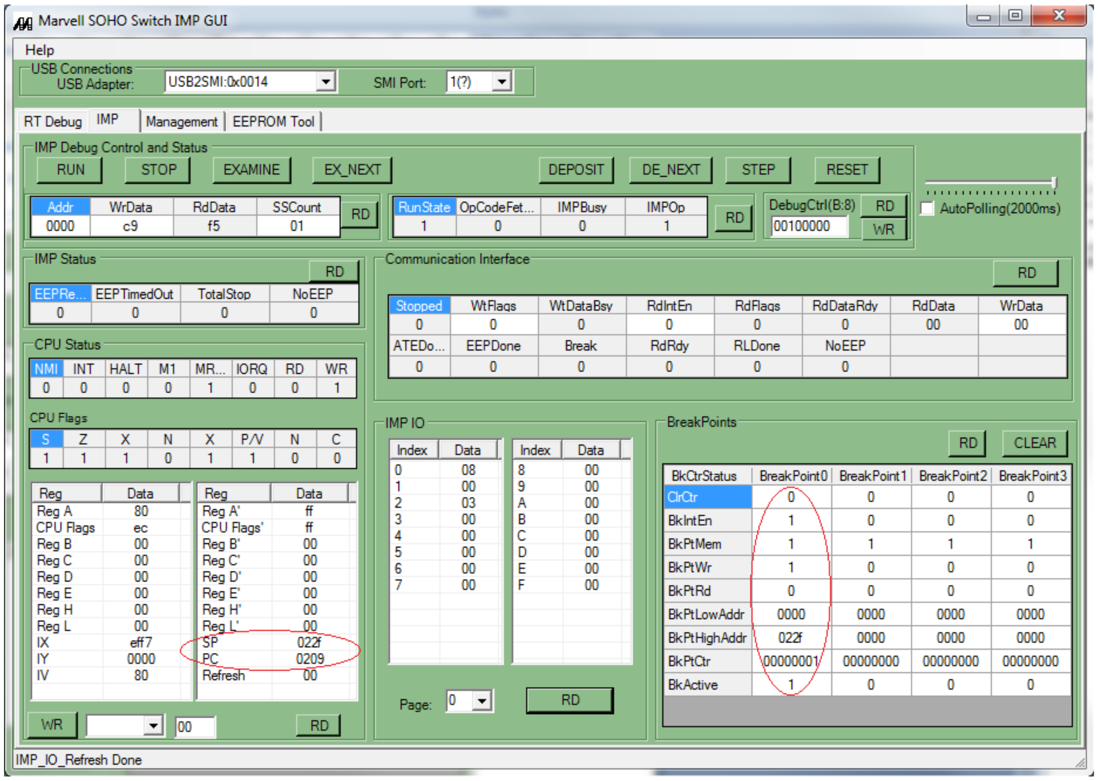
Index	Data
8	00
9	00
A	00
B	00
C	00
D	00
E	00
F	00

BreakPoints

BkCtrStatus	BreakPoint0	BreakPoint1	BreakPoint2	BreakPoint3
ClrCtr	0	0	0	0
BkIntEn	1	0	0	0
BkPtMem	1	1	1	1
BkPtWr	1	0	0	0
BkPtRd	0	0	0	0
BkPtLowAddr	0000	0000	0000	0000
BkPtHighAddr	022f	0000	0000	0000
BkPtCtr	00000000	0000000a	0000000a	0000000a
BkActive	0	0	0	0

PIC 41

Returan RT Debug tab and click “Start” button, after a while, CPU will be stopped as PIC42.



USB Connections
USB Adapter: USB2SMI:0x0014 SMI Port: 1(?)

IMP Debug Control and Status

Addr	WrData	RdData	SSCount
0000	c9	f5	01

RunState	OpCodeFet...	IMPBusy	IMPOp
1	0	0	1

DebugCtrl(B:8) 00100000

AutoPolling(2000ms)

IMP Status

EEPRe...	EEPTimedOut	TotalStop	NoEEP
0	0	0	0

Communication Interface

Stopped	WtFlags	WtDataBsy	RdIntEn	RdFlags	RdDataRdy	RdData	WrData
0	0	0	0	0	0	00	00
ATEDo...	EEPDone	Break	RdRdy	RLDone	NoEEP		
0	0	0	0	0	0		

CPU Status

NMI	INT	HALT	M1	MR...	IORQ	RD	WR
0	0	0	0	1	0	0	1

CPU Flags

S	Z	X	N	X	P/V	N	C
1	1	1	0	1	1	0	0

Reg	Data
Reg A	80
CPU Flags	ec
Reg B	00
Reg C	00
Reg D	00
Reg E	00
Reg H	00
Reg L	00
IX	eff7
IY	0000
IV	80

Reg	Data
Reg A'	ff
CPU Flags'	ff
Reg B'	00
Reg C'	00
Reg D'	00
Reg E'	00
Reg H'	00
Reg L'	00
SP	022f
PC	0209
Refresh	00

WR 00 RD

IMP IO

Index	Data
0	08
1	00
2	03
3	00
4	00
5	00
6	00
7	00

Index	Data
8	00
9	00
A	00
B	00
C	00
D	00
E	00
F	00

Page: 0

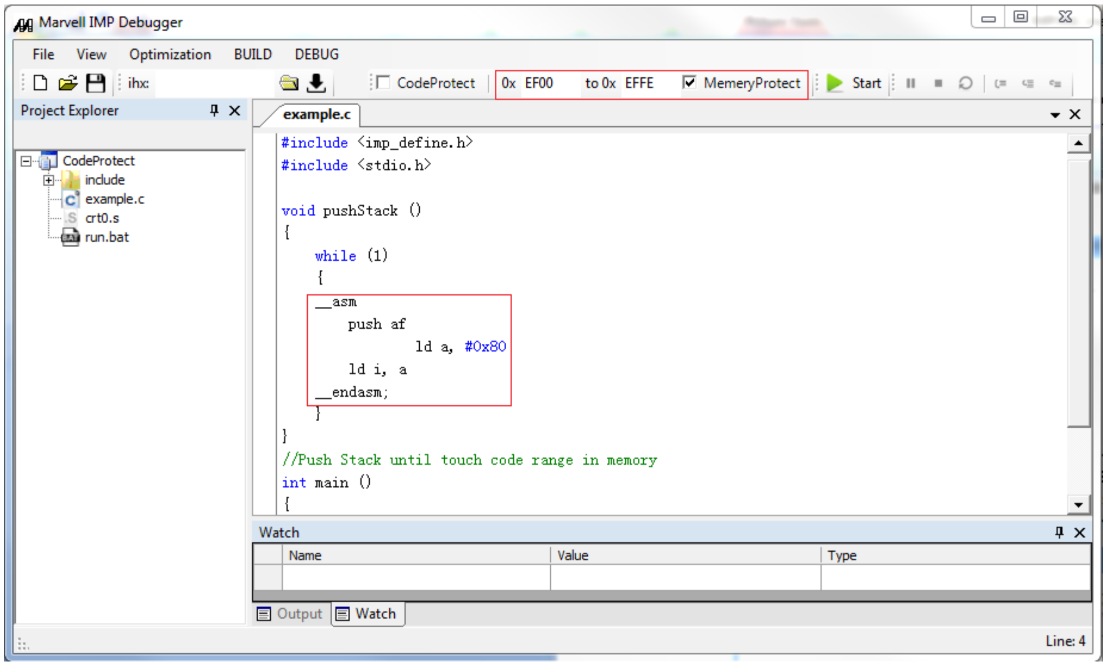
BreakPoints

BkCtrStatus	BreakPoint0	BreakPoint1	BreakPoint2	BreakPoint3
ClrCtr	0	0	0	0
BkIntEn	1	0	0	0
BkPtMem	1	1	1	1
BkPtWr	1	0	0	0
BkPtRd	0	0	0	0
BkPtLowAddr	0000	0000	0000	0000
BkPtHighAddr	022f	0000	0000	0000
BkPtCtr	00000001	00000000	00000000	00000000
BkActive	1	0	0	0

IMP_IO_Refresh Done

PIC 42

<u>MemoryProtect</u> sample just the same as code protect. Following pictures will show this case.



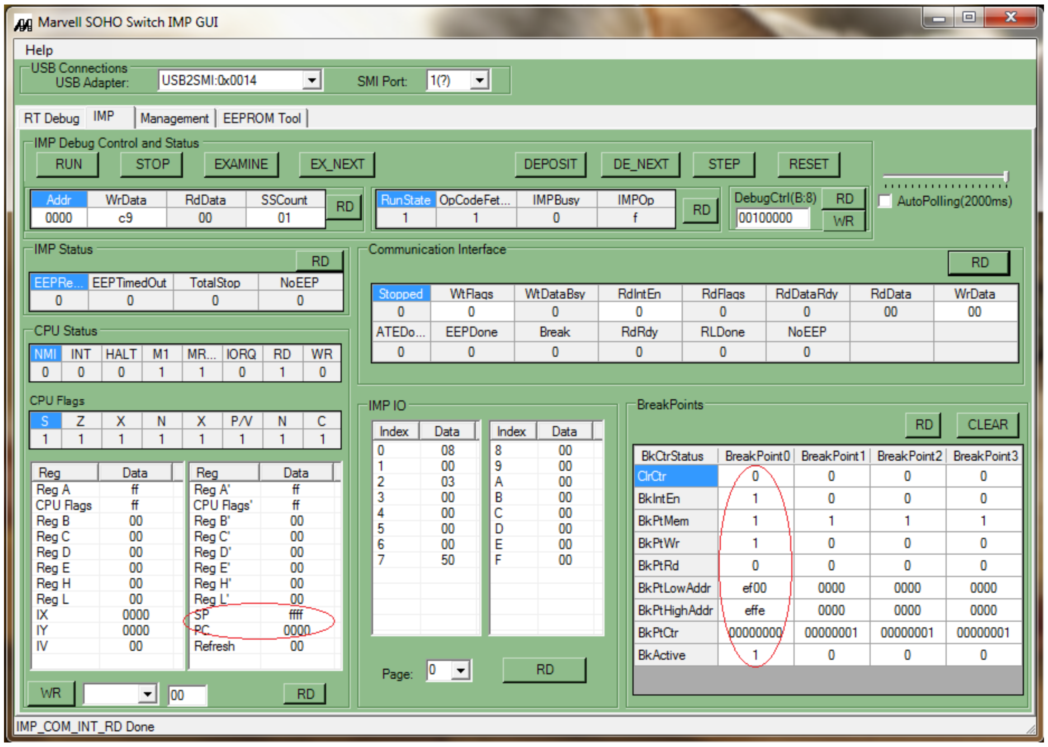
CodeProtect 0x EF00 to 0x EFFE ☑ MemeryProtect

```
#include <imp_define.h>
#include <stdio.h>

void pushStack ()
{
    while (1)
    {
    __asm
        push af
            ld a, #0x80
        ld i, a
    __endasm;
    }
}
//Push Stack until touch code range in memory
int main ()
{
```

Watch

Name	Value	Type

PIC 43



USB Connections
USB Adapter: USB2SMI:0x0014 SMI Port: 1(?)

IMP Debug Control and Status

Addr	WrData	RdData	SSCount
0000	c9	00	01

RunState	OpCodeFet...	IMPBusy	IMPOp
1	1	0	f

DebugCtrl(B:8) 00100000
AutoPolling(2000ms)

IMP Status

EEPRe...	EEPTimedOut	TotalStop	NoEEP
0	0	0	0

CPU Status

NMI	INT	HALT	M1	MR...	IORQ	RD	WR
0	0	0	1	1	0	1	0

CPU Flags

S	Z	X	N	X	P/V	N	C
1	1	1	1	1	1	1	1

Reg	Data	Reg	Data
Reg A	ff	Reg A'	ff
CPU Flags	ff	CPU Flags'	ff
Reg B	00	Reg B'	00
Reg C	00	Reg C'	00
Reg D	00	Reg D'	00
Reg E	00	Reg E'	00
Reg H	00	Reg H'	00
Reg L	00	Reg L'	00
IX	0000	SP	ffff
IY	0000	PC	0000
IV	00	Refresh	00

WR 00

Communication Interface

Stopped	WtFlags	WtDataBsy	RdIntEn	RdFlags	RdDataRdy	RdData	WrData
0	0	0	0	0	0	00	00
ATEDo...	EEPDone	Break	RdRdy	RLDone	NoEEP		
0	0	0	0	0	0		

IMP IO

Index	Data	Index	Data
0	08	8	00
1	00	9	00
2	03	A	00
3	00	B	00
4	00	C	00
5	00	D	00
6	00	E	00
7	50	F	00

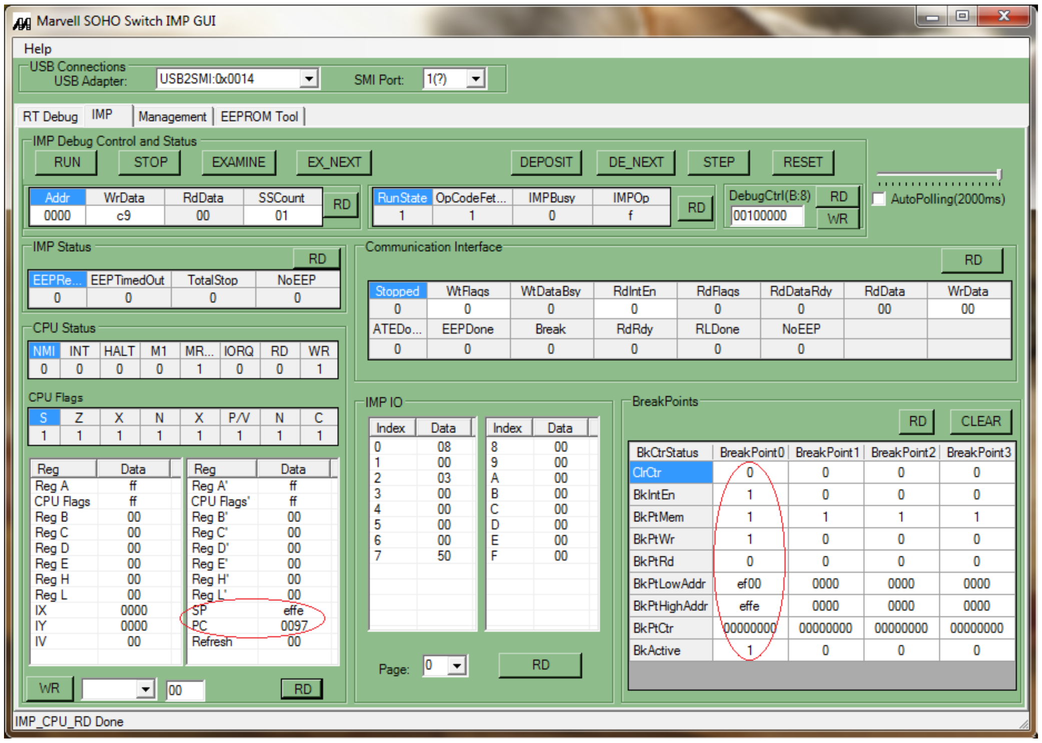
Page: 0

BreakPoints

BkCtrStatus	BreakPoint0	BreakPoint1	BreakPoint2	BreakPoint3
ClrCtr	0	0	0	0
BkIntEn	1	0	0	0
BkPtMem	1	1	1	1
BkPtWr	1	0	0	0
BkPtRd	0	0	0	0
BkPtLowAddr	ef00	0000	0000	0000
BkPtHighAddr	effe	0000	0000	0000
BkPtCtr	00000000	00000001	00000001	00000001
BkActive	1	0	0	0

IMP_COM_INT_RD Done

PIC 44



USB Connections
USB Adapter: USB2SMI:0x0014 SMI Port: 1(?)

IMP Debug Control and Status

Addr	WrData	RdData	SSCount
0000	c9	00	01

RunState	OpCodeFet...	IMPBusy	IMPOp
1	1	0	f

DebugCtrl(B:8) 00100000
AutoPolling(2000ms)

IMP Status

EEPRe...	EEPTimedOut	TotalStop	NoEEP
0	0	0	0

CPU Status

NMI	INT	HALT	M1	MR...	IORQ	RD	WR
0	0	0	0	1	0	0	1

CPU Flags

S	Z	X	N	X	P/V	N	C
1	1	1	1	1	1	1	1

Reg	Data	Reg	Data
Reg A	ff	Reg A'	ff
CPU Flags	ff	CPU Flags'	ff
Reg B	00	Reg B'	00
Reg C	00	Reg C'	00
Reg D	00	Reg D'	00
Reg E	00	Reg E'	00
Reg H	00	Reg H'	00
Reg L	00	Reg L'	00
IX	0000	SP	effe
IY	0000	PC	0097
IV	00	Refresh	00

WR 00

Communication Interface

Stopped	WtFlags	WtDataBsy	RdIntEn	RdFlags	RdDataRdy	RdData	WrData
0	0	0	0	0	0	00	00
ATEDo...	EEPDone	Break	RdRdy	RLDone	NoEEP		
0	0	0	0	0	0		

IMP IO

Index	Data	Index	Data
0	08	8	00
1	00	9	00
2	03	A	00
3	00	B	00
4	00	C	00
5	00	D	00
6	00	E	00
7	50	F	00

Page: 0

BreakPoints

BkCtrStatus	BreakPoint0	BreakPoint1	BreakPoint2	BreakPoint3
ClrCtr	0	0	0	0
BkIntEn	1	0	0	0
BkPtMem	1	1	1	1
BkPtWr	1	0	0	0
BkPtRd	0	0	0	0
BkPtLowAddr	ef00	0000	0000	0000
BkPtHighAddr	effe	0000	0000	0000
BkPtCtr	00000000	00000000	00000000	00000000
BkActive	1	0	0	0

IMP_CPU_RD Done

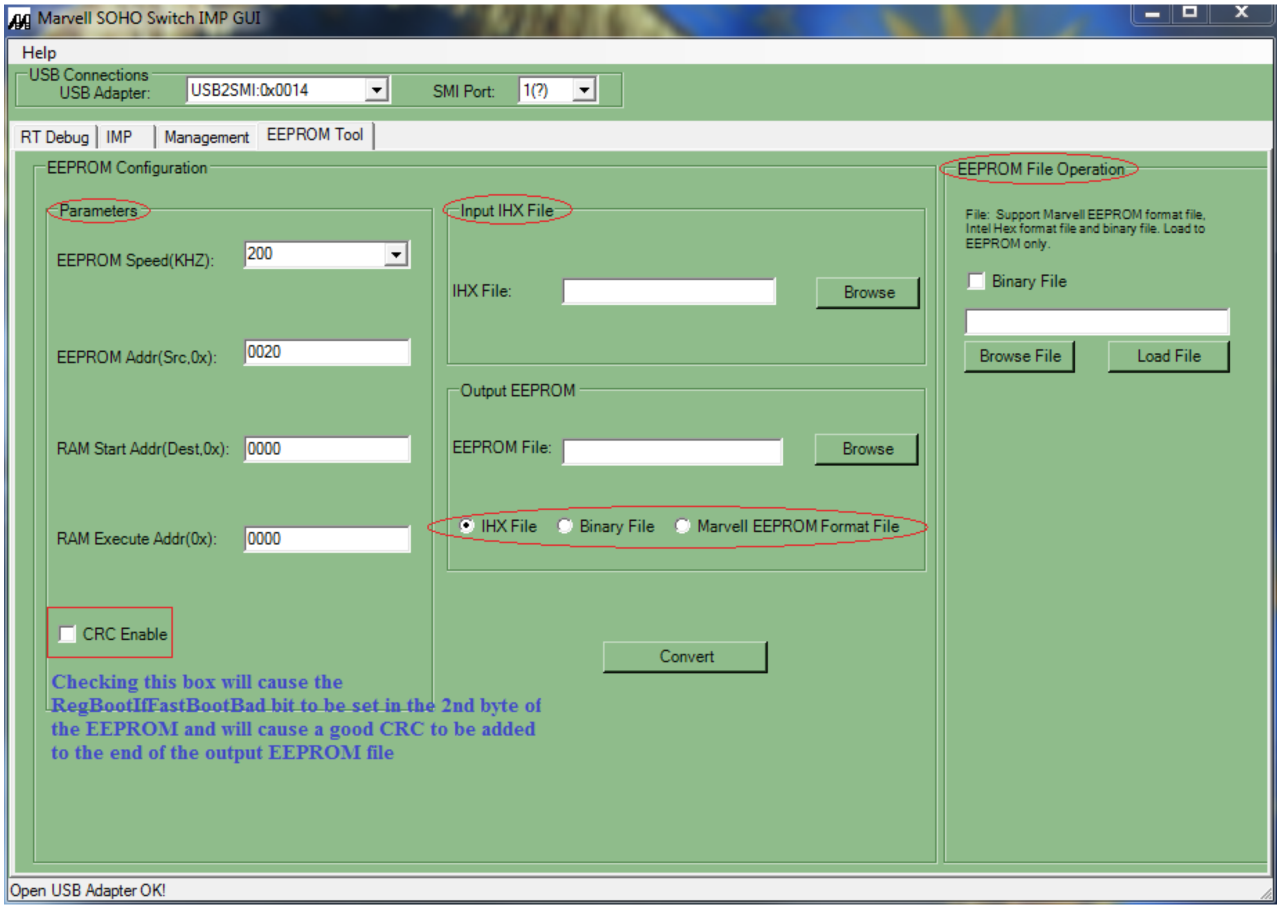
PIC 45

3.4 EEPROM Tool Page

As PIC46 shows, you can see "Parameters", "Input IHX File", "Output EEPROM File" and "EEPRO File Operation". The "Parameters" list all entries with the default values, and you can check "CRC Enable" checkbox (**PCI46 introduce this checkbox's function as well**) if you want include corresponding bytes into the generated file. Before you click "Convert" button, you must make sure all the above items have been configured, otherwise, the converting will be failed.

So far, the EEPROM tool can support "Marvell EEPROM Format File", "Binary File" and "IHX File".

"EEPRO File Operation" support load hex or binary file to EEPROM



USB Connections
USB Adapter: USB2SMI:0x0014
SMI Port: 1(?)

EEPROM Speed(KHZ): 200
EEPROM Addr(Src,0x): 0020
RAM Start Addr(Dest,0x): 0000
RAM Execute Addr(0x): 0000
[ ] CRC Enable

Checking this box will cause the RegBootIfFastBootBad bit to be set in the 2nd byte of the EEPROM and will cause a good CRC to be added to the end of the output EEPROM file

IHX File:
EEPROM File:
[✓ IHX File] Binary File Marvell EEPROM Format File

File: Support Marvell EEPROM format file, Intel Hex format file and binary file. Load to EEPROM only.
[ ] Binary File

Open USB Adapter OK!

PIC 46

4. KNOWN ISSUES

1. The GUI will be hung when set breakpoint for C code line “printf(…)” and step in it.
2. For pure asm project, not support "pause" feature.