CANTUS™ 에서 JTAG 디버깅 하기

Ver 1.2

문서 내용 변경 사항

1.0 (2010-11-16)

first release

1.1(2010-11-19)

Jtag Mode 설정pin이 소프트웨어적으로 사용 될 경우에 대한 대처 법 추가(3.2)

1.2(2011-01-13)

2. GDB(Insight)와 CANTUS 연결을 위한 설정 수정

5) Start Debugger 수정

목 차

1. CANTUS 에서 JTAG 디버깅을 위한 장비 및 프로그램

CANTUS에서 Jtag 디버깅를 사용하기 위해서는 필요한 장비 및 프로그램이 필요하다.

- ✓ E-CON (USB to JTAG 변환 장비)
- ✓ EConMan.exe (PC 용 E-CON 제어 프로그램, EISC Studio 3 설치 시 /econ에 설치됨) : version 0.9.8 버전 이상
- ✓ AE32000 용 Source level debugger 인 GDB (ae32000-elf-gdb 또는 ae32000-elf-insight EISC Studio 3 설치 시 포함)
  - ■ Gdb 관련 문서는 http://sources.redhat.com/gdb/ 에서 찾아 볼 수 있다.
- ✓ Support platform: MS Windows XP SP3 Later

*** Cantus 에 내장된 Hardware breakpoint 개수는 총 8개이며 1개는 debugger(Gdb) 가 제어용으로 사용하게 되므로 사용자는 총 7개의 hardware breakpoint 를 사용 할 수 있다.**

*** Cantus 는 내장 flash memory 에 프로그램이 로딩 되므로 software breakpoint 를 사용할 수 없다.**

*** 이 문서는 source level debugger 인 gdb 와 insight 구분 없이 GDB로 명칭을 통일하여 작성되었다.**

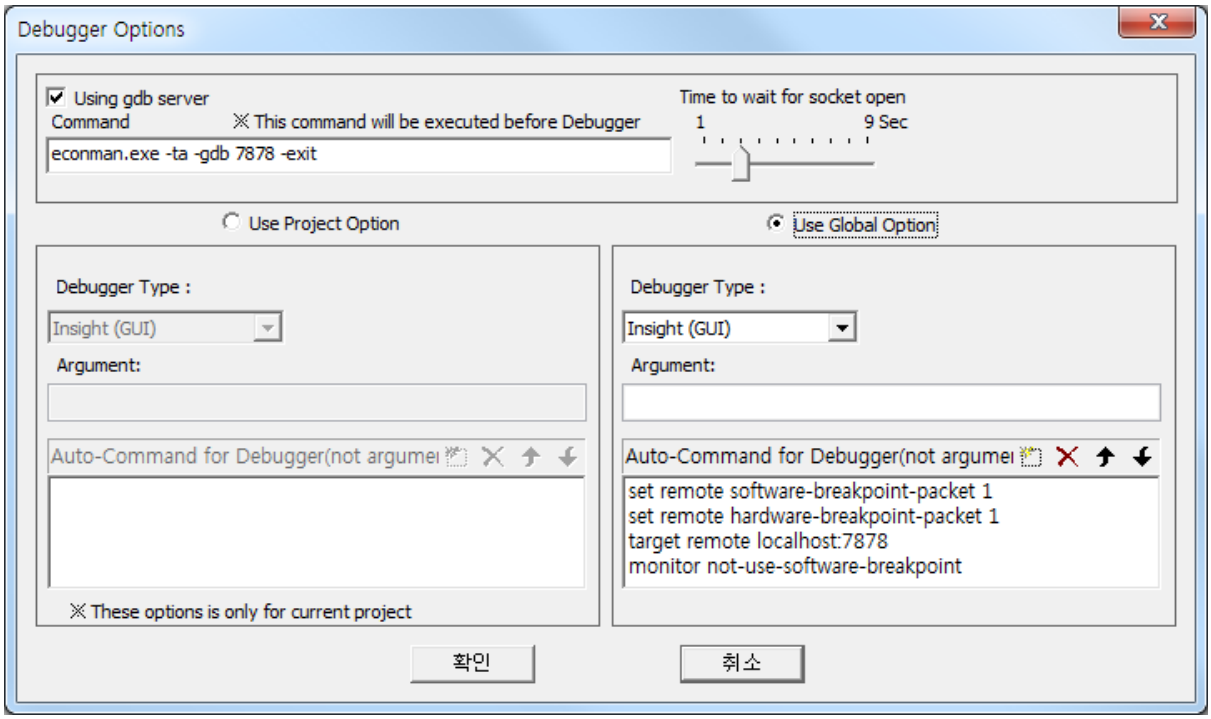
2. GDB(Insight) 와 CANTUS 연결을 위한 설정

CANTUS를 원격 디버깅 하기 위해서는 GDB 초기화 과정이 필요 하다.
초기화 과정에서 사용하는 GDB명령은 다음과 같다. monitor 명령어는 반드시 target 명령어 이후에 해야 한다.

```
set remote hardware-breakpoint-packet 1
set remote software-breakpoint-packet 1
target remote :7878
monitor not-use-software-breakpoint
```

"target remote :7878" : EconMan에서 "gdbserver"명령을 통해 Open한 Port번호로 접속 한다.
"monitor not-use-software-breakpoint" : 모든 breakpoint 명령어를 hardware로 처리하도록 설정 명령어이다.

이상을 EISC Studio 3 Menu에서 "Debug" → "Debug Options"에서 아래와 같이 설정 한다.



Using gdb server : Debugger가 실행할 때 가장 먼저 실행되는 command로, EConMan을 사용하여, target에 연결하고, Port 7878로 gdbserver를 열어 둔다. 기본적으로 설정되어 있으며, 별도 수정은 필요는 없다.

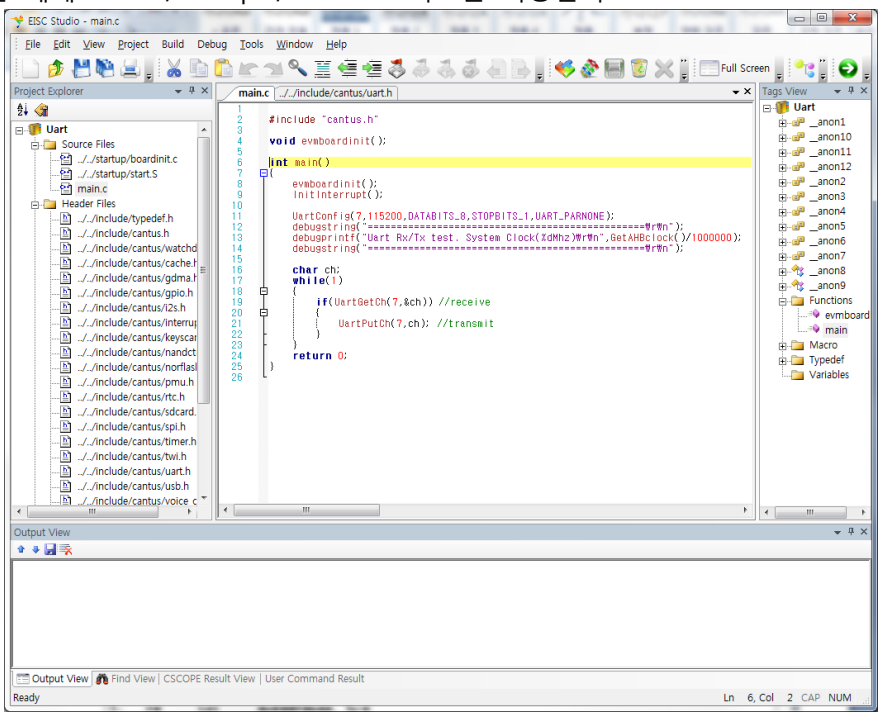
Use Project Option : 현재 Project에서만 유효한 Debugger Type과 Argument를 설정한다.

Use Global Option : 모든 Project에서 유효한 Debugger Type과 Argument를 설정한다. 기본적으로 Argument가 채워져 있다. 별도 수정은 필요는 없다.

3. GDB 와 CANTUS 연결하기

EISC Studio3 에서 디버깅할 프로젝트를 Open 한다.

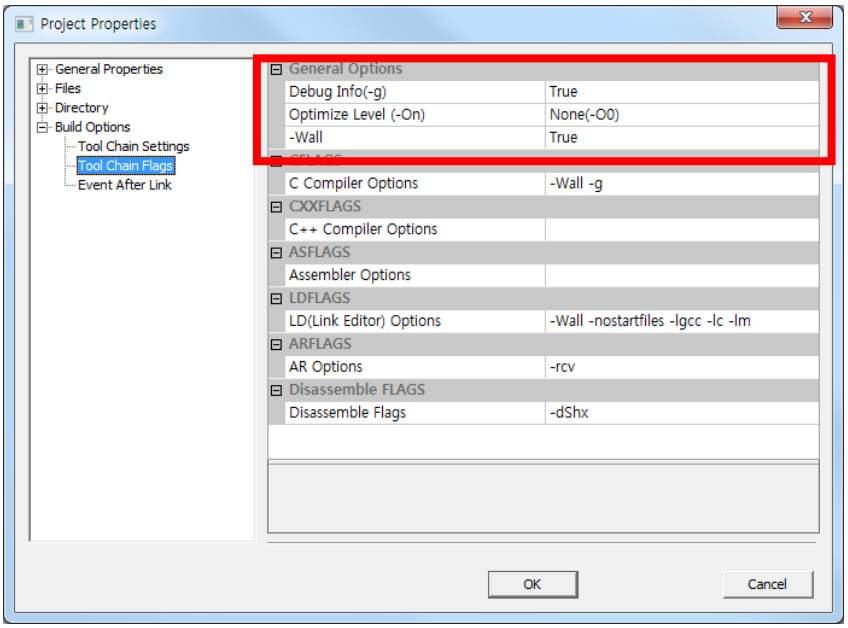
이 문서에서는 예제로 SDK/Example/Uart 프로젝트를 이용한다.



1) 프로젝트 빌드

디버깅을 위해서는 프로그램을 컴파일할 때 반드시 디버깅 정보를 생성하도록 해야 한다.

프로젝트 설정에서 "Debug Info(-g)" 를 True 로 설정 하고 Optimize Level 은 None 으로 해야 소스라인과 disassemble 내의 assemble 정보가 더 정확해 진다. 최적화 옵션을 사용하면 최적화 됨에 따라서 일치 하지 않을 수 있다.



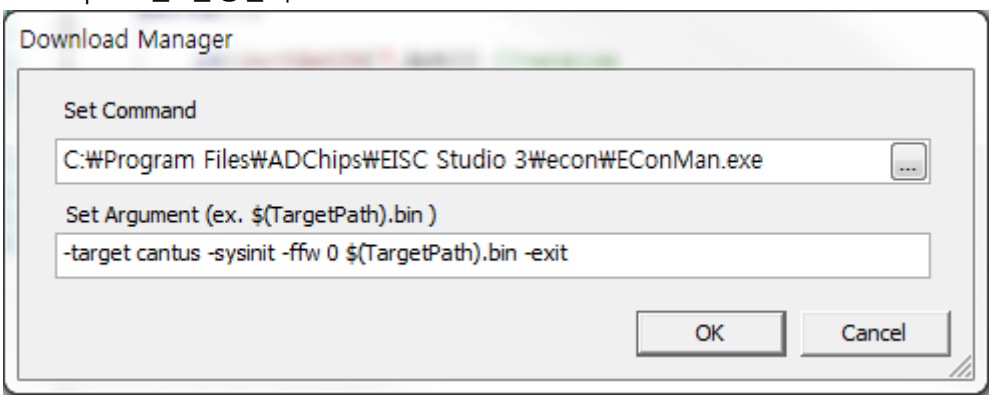
2) Jtag Mode

CANTUS를 JTAG Mode로 진입한다. JTAG Mode를 위한 nJTAGSEL Pin은 P0.7/A7/D7/A15와 공유된 PIN 이다. 따라서 만약 프로그램이 이 Pin을 사용하는 경우라면 Jtag Mode로 진입 후 Pin의 입력 신호를 풀어 P0.7/A7/D7/A15로 동작 할 수 있도록 하여야 한다.

3) Binary Download

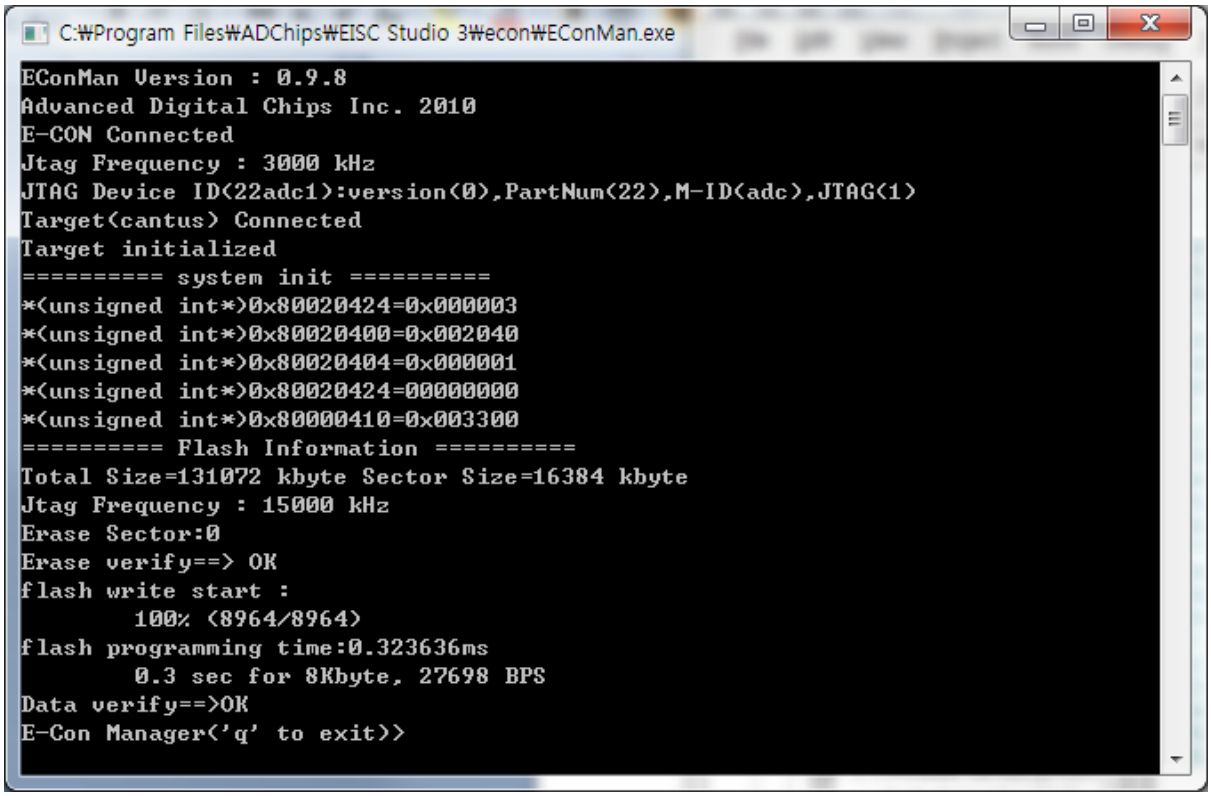
새로 만들어진 Binary를 CANTUS에 다운로드 한다

E-Con 을 이용하여 다운로드 할 경우 아래와 같이 Menu에서 "Build" → "Download Option"을 통해 Download Option을 설정한다.



설정 완료 후 Menu에서 "Build" → "Download to Target"을 실행 하면 프로젝트에서 생성된 binary 파일이 다운로드 된다. 다운로드 완료 후 CANTUS를 Reset 한다.

Windows Vista 나 Windows 7 의 경우 관리자 권한으로 실행하지 않을 경우 설정을 저장하지 못하므로 반드시 관리자 권한으로 실행해야 한다.



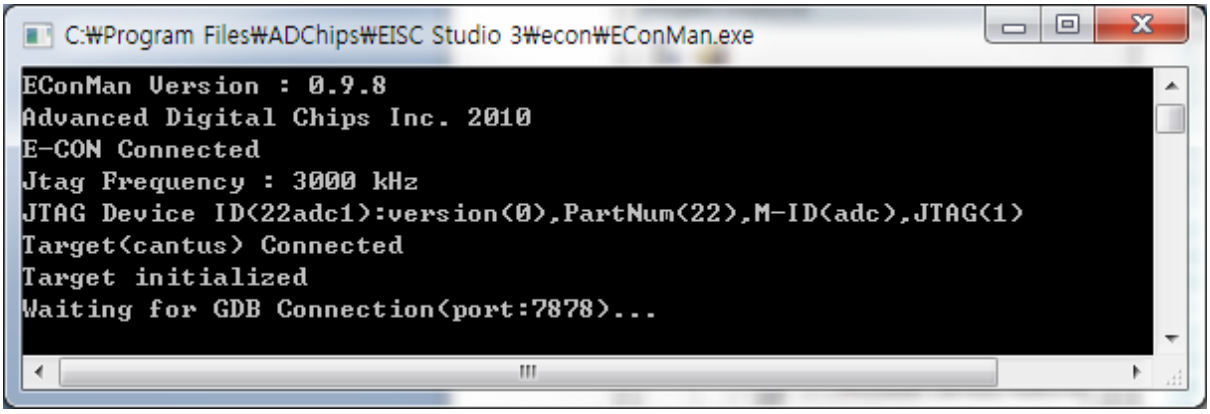
4) Debug Option

"2. GDB(Insight)와 CANTUS 연결을 위한 설정"과 같이 Menu에서 "Debug" → "Debug Options"을 설정한다.

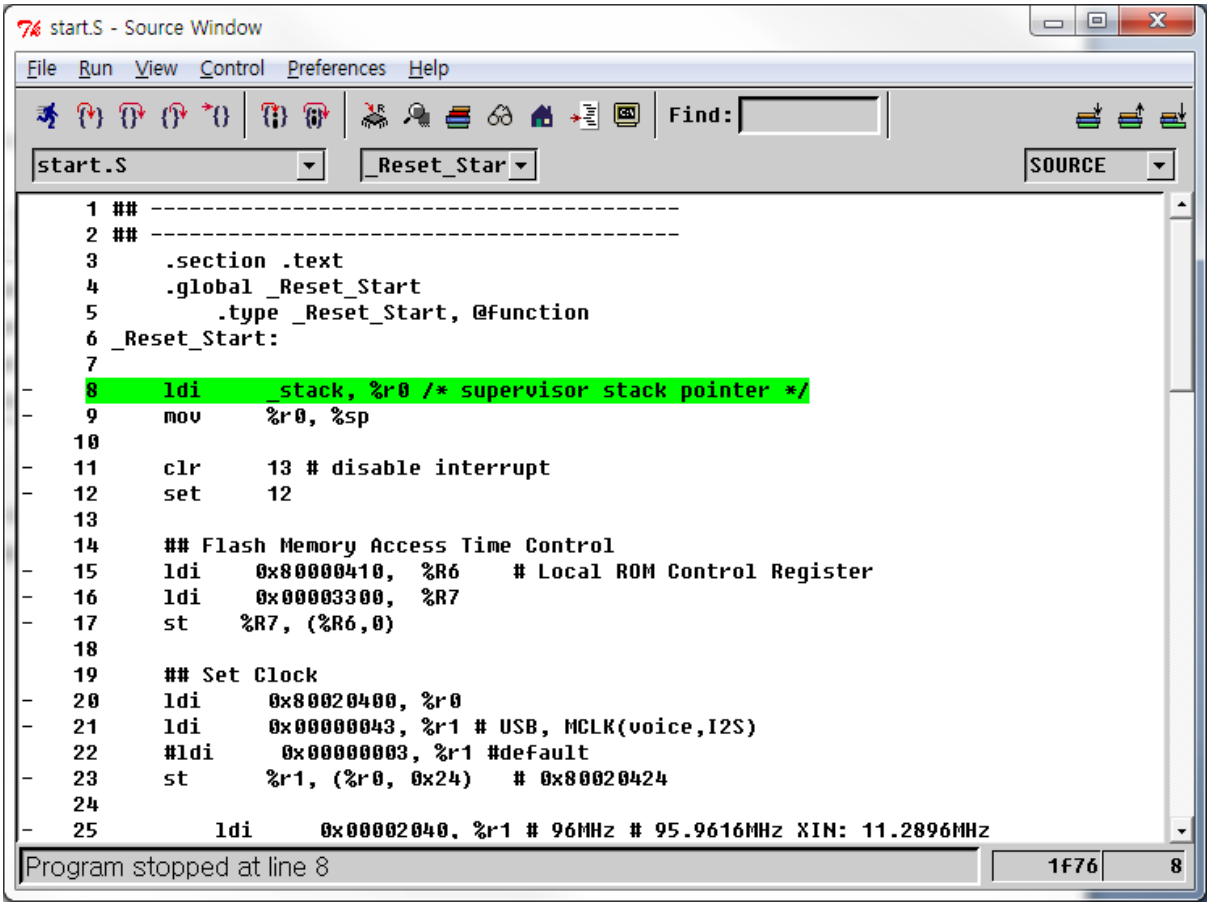
5) Start Debugger

Menu에서 "Debug" → "Start Debugger"를 실행한다.

아래와 같이 EConMan이 실행되어, target을 설정하고, Port 7878로 GDB Server를 Open한다.

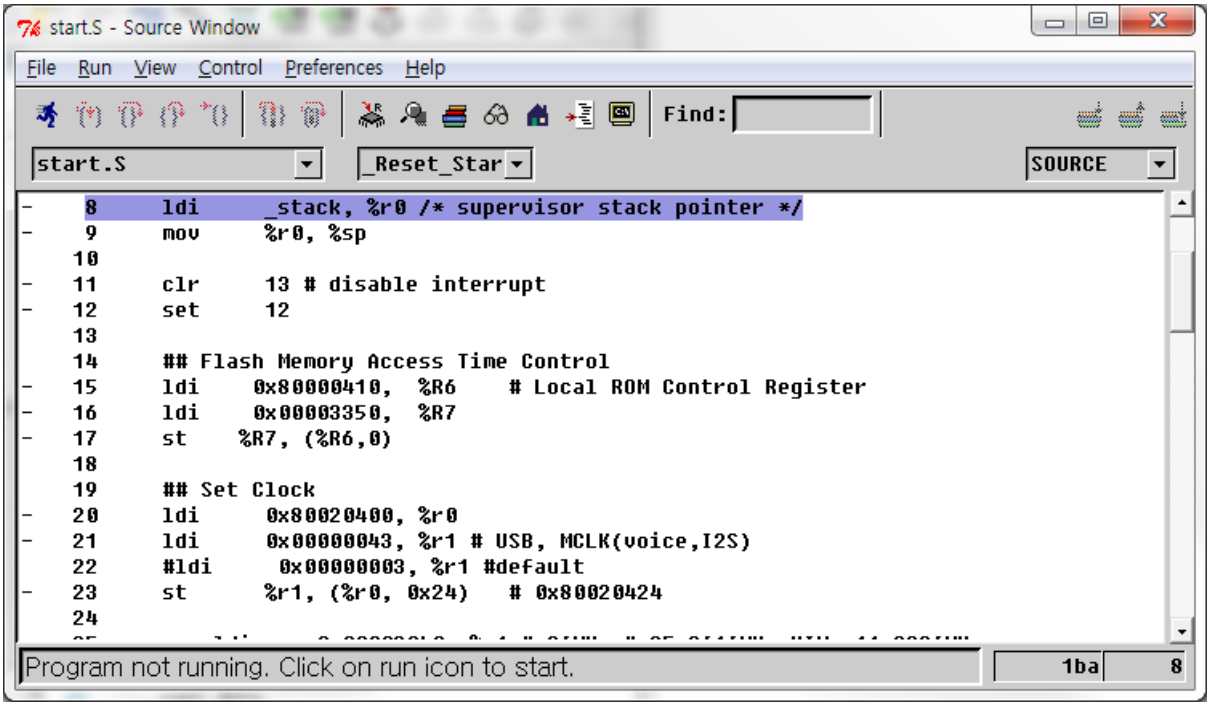


GDB가 실행 되면서 EConMan 에서 Open한 Socket으로 접속하여 프로세서가 다음에 실행할 소스라인을 연두색으로 표시한다.



"Select function name to disassemble"이라고 나타나면 위 start.S부분을 click하여 start.S를 선택한다.

이 상태에서 원하시는 지점에 breakpoint 를 설정하고 continue 실행하거나 Step, 또는 Next 명령어를 이용하여 디버깅 하면 된다.



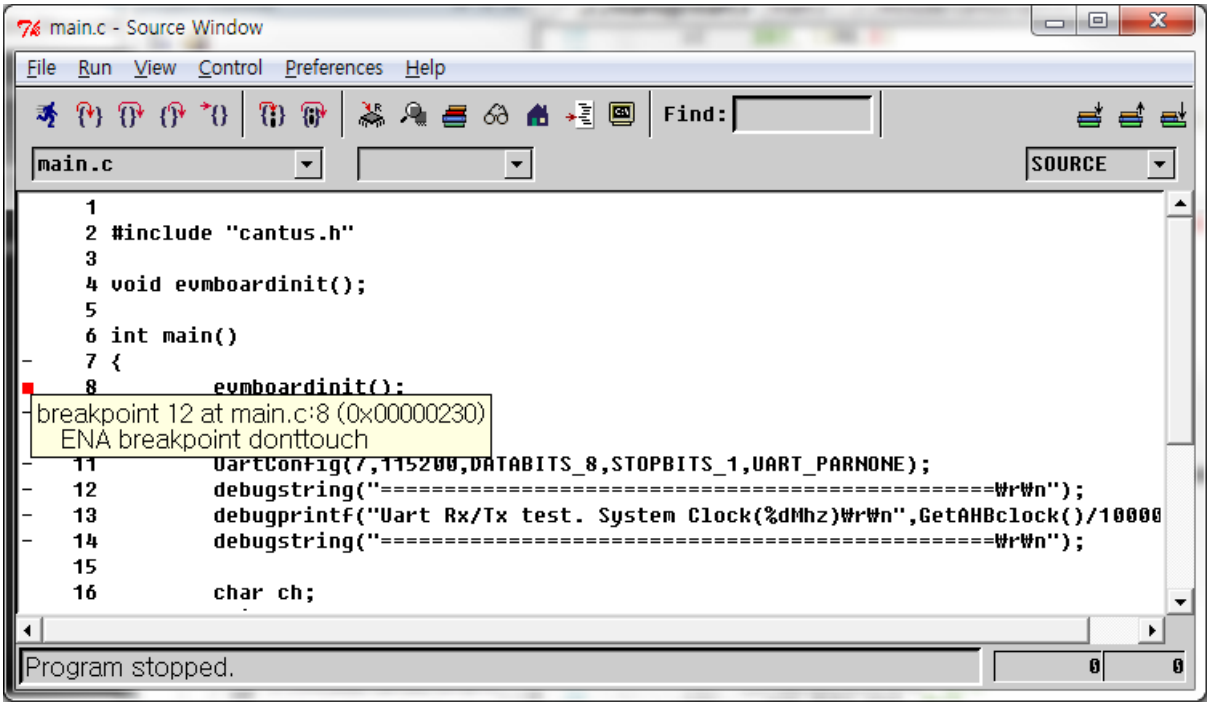
위와 같이 보라색으로 표시된다면 EConMan 과 연결이 되지 않았다는 의미이다.

이 경우 EConMan 이 정상적으로 실행되었는지 확인해보거나 Port 번호가 맞는지 확인해야 된다.

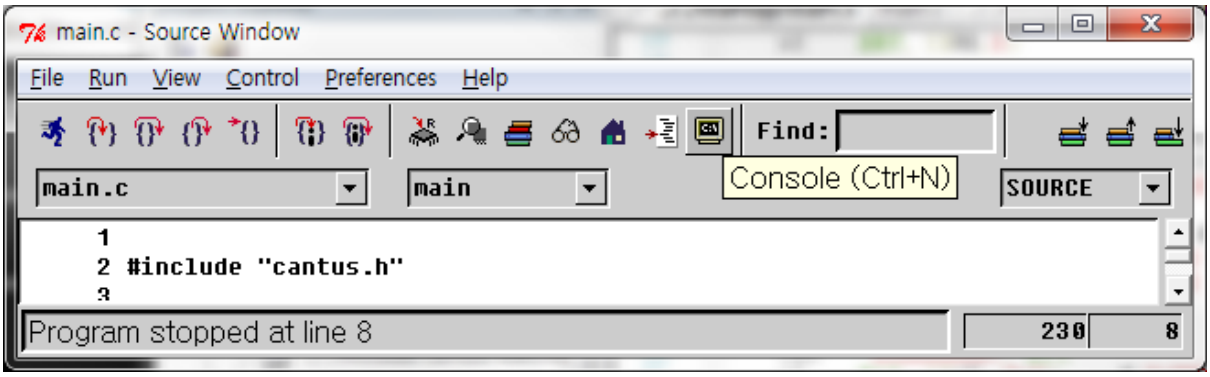
7) Breakpoint 설정

main 함수에 breakpoint 를 설정하고 continue 를 실행 해보자.

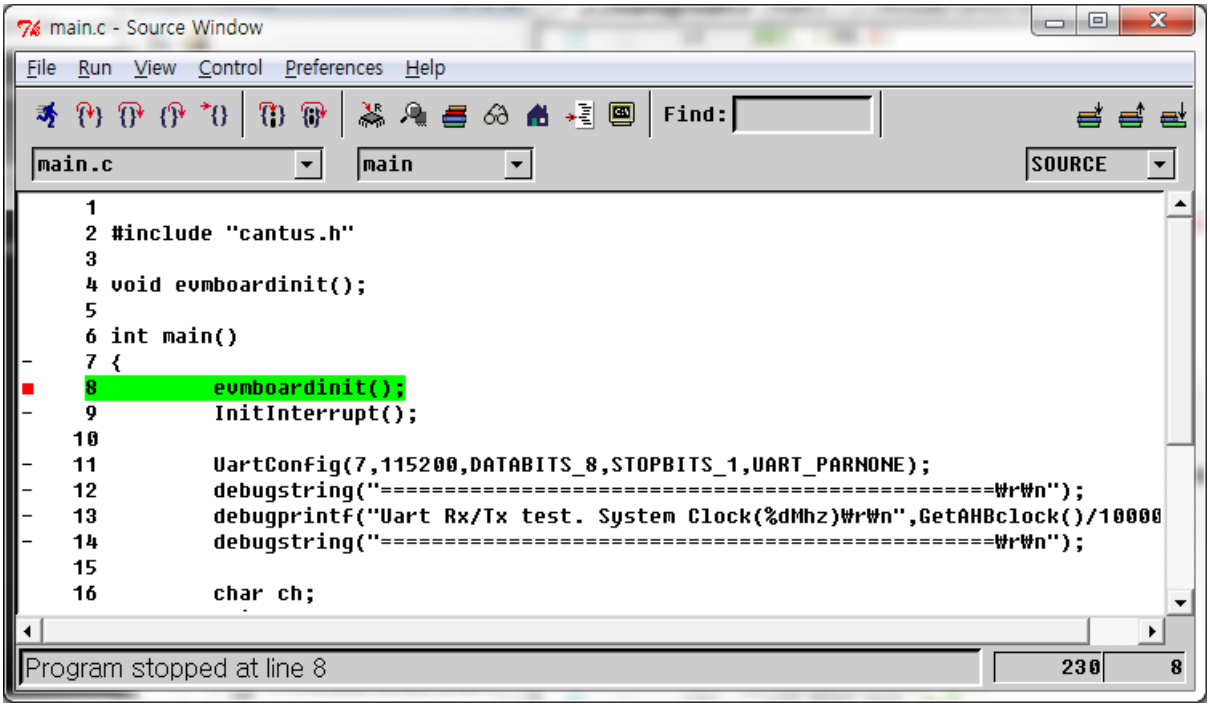
현재 창에서 왼쪽 상단의 소스파일명을 나타내는 콤보박스를 클릭하여 원하는 소스파일을 연 다음 소스라인넘버 왼쪽을 클릭하면 breakpoint 가 설정된다.



이 상태에서 키보드의 'c' 를 눌르거나 command 창에서 'c' 를 입력하면 다음 breakpoint 가 설정된 위치까지 실행하게 된다.("Run" 명령어가 아님을 주의)



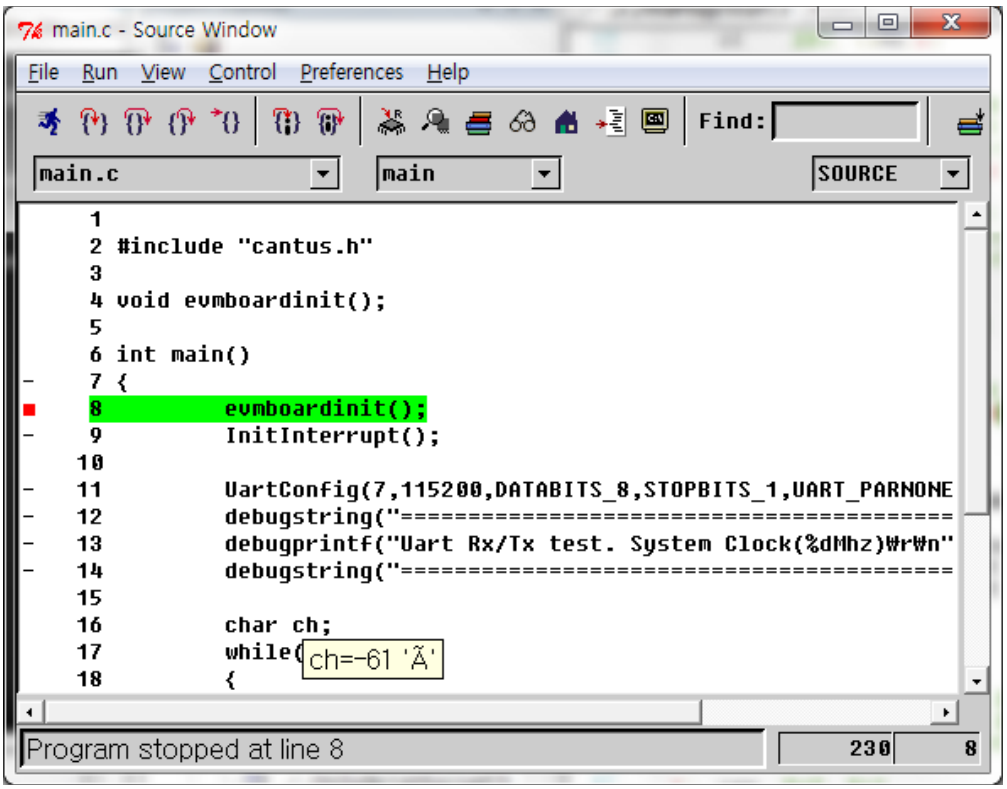
Command 창은 메뉴아이콘 중에서 console 을 실행 하면 별도의 창이 실행되면서 사용자가 직접 명령어를 입력 할 수 있다. Gdb 명령어는 그 수가 너무 많기 때문에 일부 명령어(continue, step, next, finish,..) 를 제외하고는 사용자가 직접 입력해야만 한다.



위 그림은 main 함수까지 실행 한 이후의 화면이다.

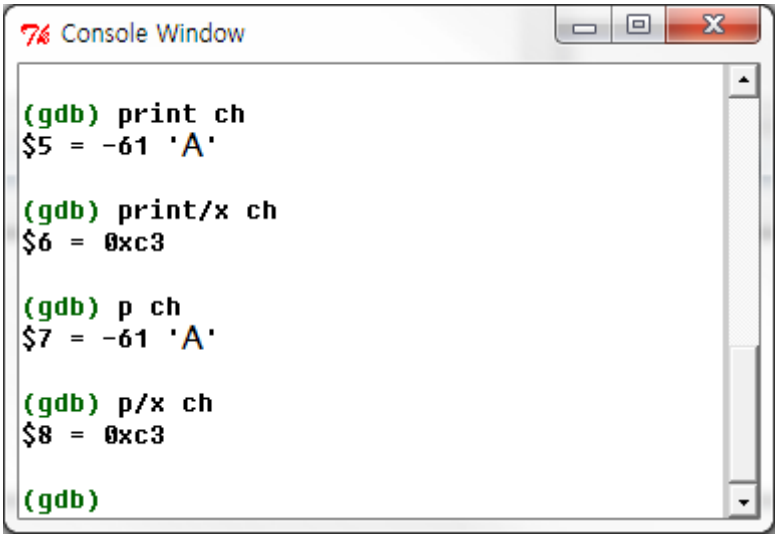
8) 변수 값 출력

원하는 변수 값이나 함수의 번지를 알고 싶다면 해당 변수나 함수에 마우스 커서를 위치 하면 현재 상태 값을 나타낸다.



위 그림은 지역변수 char ch 에 커서를 위치하여 변수 값을 출력한 모습니다.

Console 에서 사용자가 직접 입력하여 그 값을 출력 할 수도 있다.



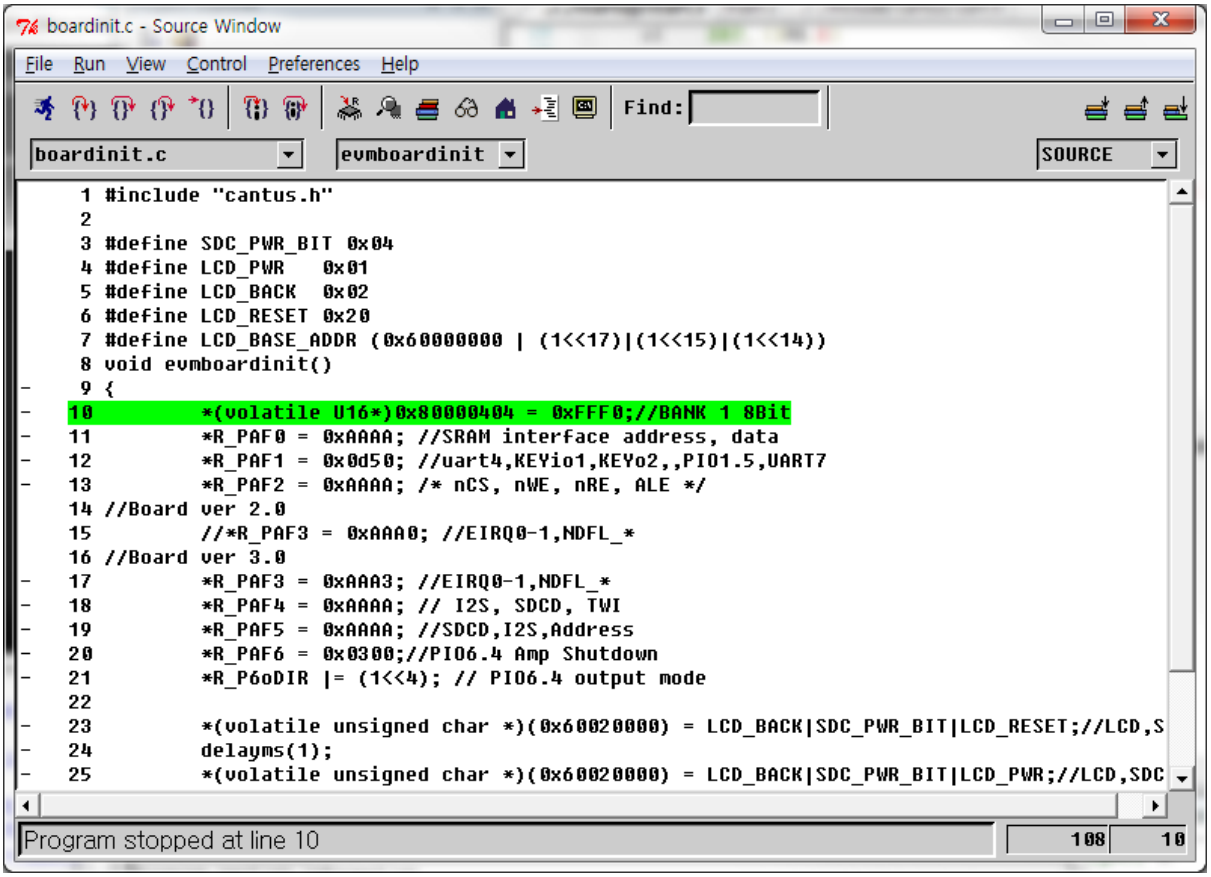
"print"("p"라고 입력해도 된다) 명령어는 변수 값을 출력하는 명령어이다.

*변수 ch 는 초기화 값이 없기 때문에 변수 값이 다를 수 있다.

9) 함수 안으로 들어가기

현재 evmboardinit 이라는 함수는 호출 하기 전 단계에 위치 해 있다. 이 함수 안으로 들어가보도록 하겠다.

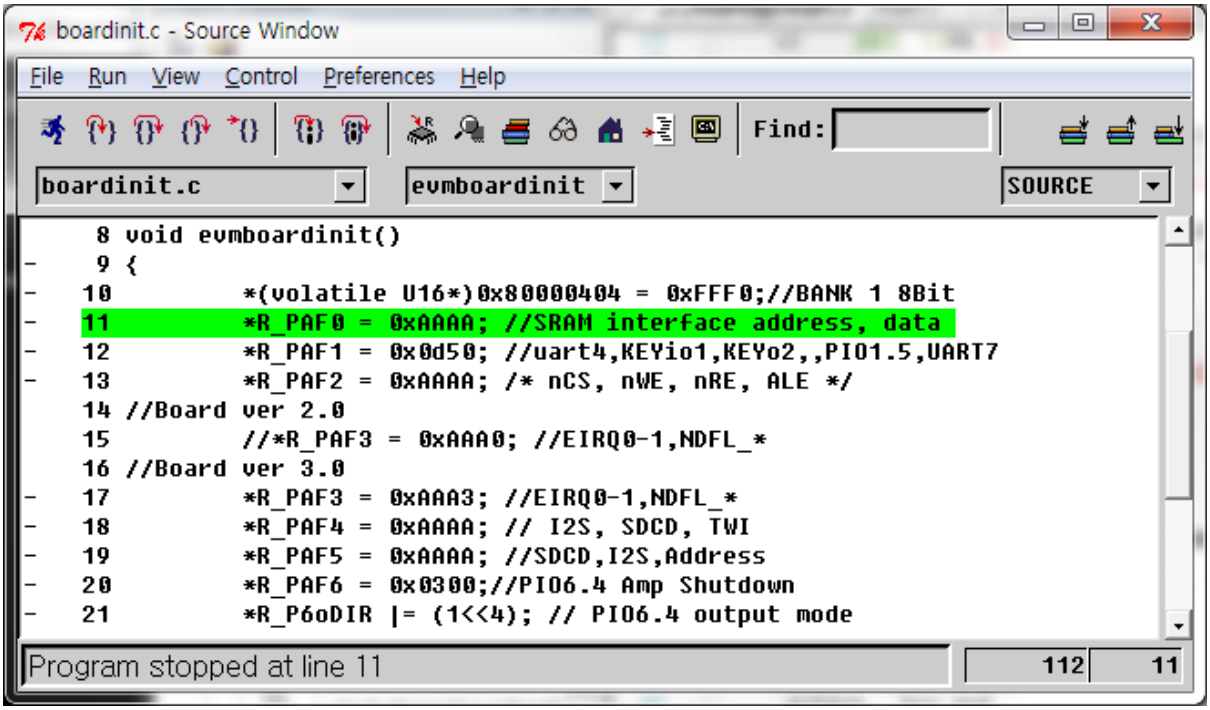
Console 창에서 "step" 이라는 명령어를 실행하거나 소스창이 활성이 되어 있는 상태에서 's' 키를 누르면 된다.



위 그림은 emvboardinit() 함수 안으로 들어온 화면이다.

10) 소스라인 하나씩 실행

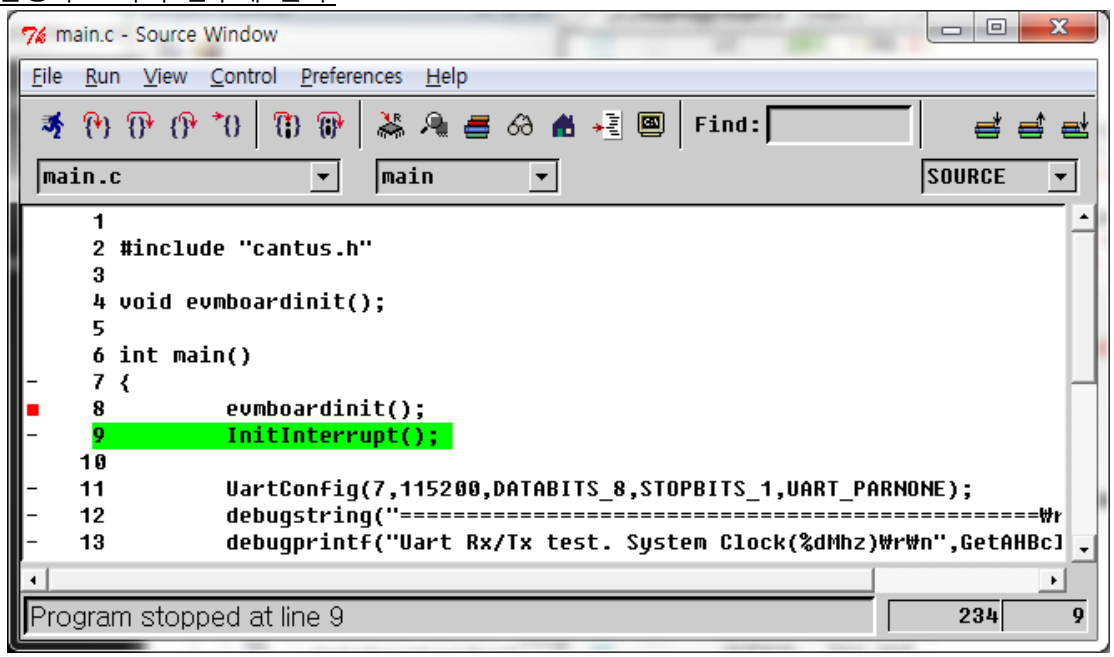
현재 상태에서 소스라인 하나씩 실행 하기 위해서는 키보드 'n' 를 누르거나 console 창에서 "next"('n'만 입력해도 됨) 를 입력 하면 된다.



11) 함수 밖으로 나가기

현재 실행중인 함수 밖으로 나가기 위해서는 키보드 'f'를 누르거나 console 창에서 "finish"("fin" 만 입력해도 된다)입력해도 된다.

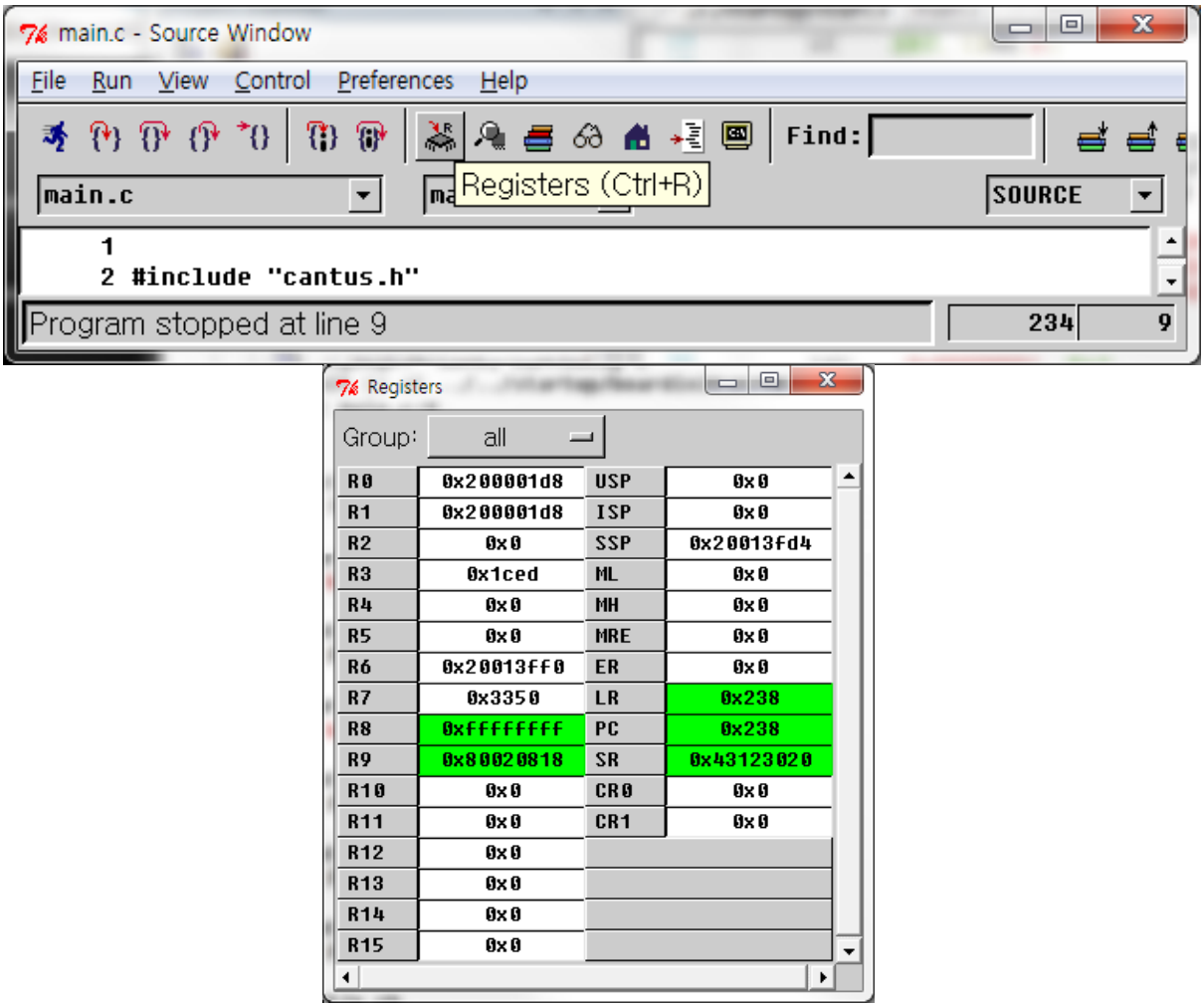
<u>이 명령어는 현재 라인 이후 명령어를 무시하고 함수 밖으로 나가는 것이 아니라 현재 함수를 모드 실행하고 나서 멈추게 된다.</u>



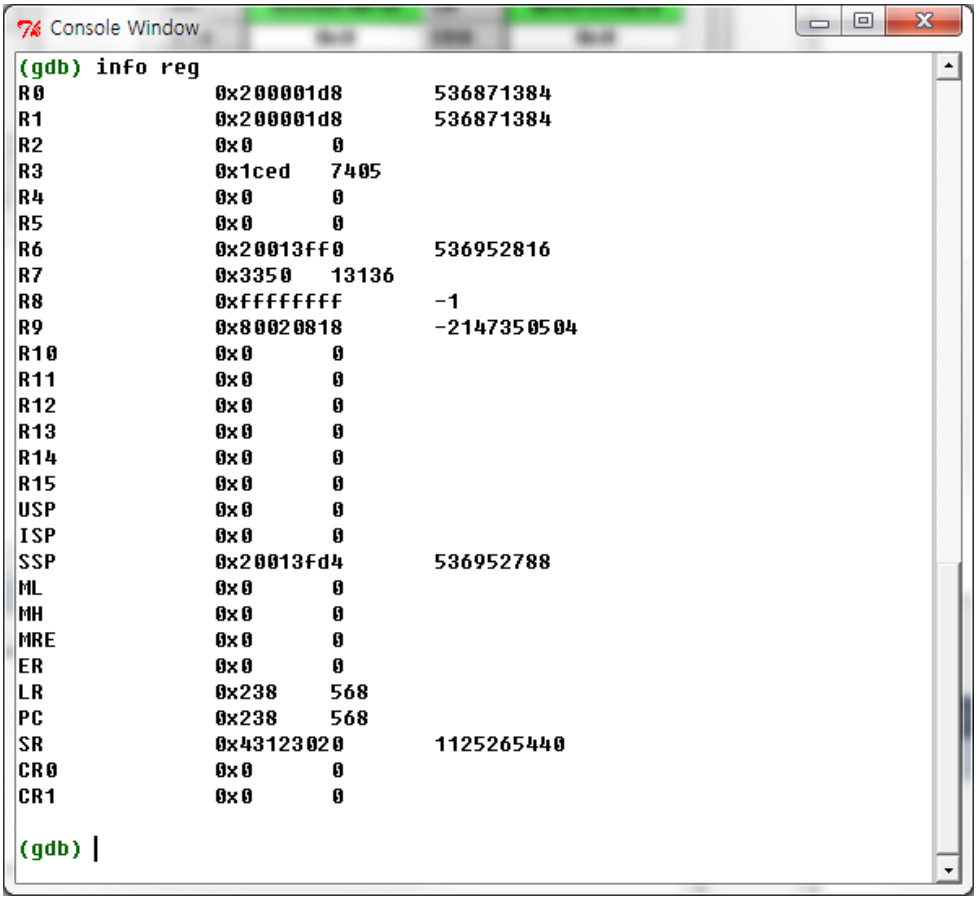
위 그림과 같이 evmboardinit 를 호출 한 이후에 멈추게 된다.

12) CPU register값 출력

현재 상태의 CPU(ae32000) 의 register 값을 출력하고 싶다면 아래 그림과 같이 register view 창을 실행 하면 된다. Register 가 변경 될 때 마다 자동으로 수정 된 값을 보여준다



또는 console 창에서 "info reg"("i reg")를 입력 해도 된다.



13) Console 창에서 명령어 반복 실행

Console 창에서 마지막에 실행 했던 명령어를 다시 실행 하기 위해서는 단순히 ENTER 키만 입력하면 가장 최근에 입력했던 명령어가 실행된다.

프로그램 실행 제어 명령어인 next 나 step 명령어가 가장 빈번이 사용되는데 이 경우 유용하다.

14) GDB 종료

소스 창을 종료하거나 Console 창에서 “quit”또는 “q”를 입력하면 종료한다.

4. JTAG debug 예외사항

1. Hardware Watchpoint 는 지원하지 않는다.
2. pop pc 명령어에서 single step 을 하면 2개를 명령어를 실행하고 멈춘다.

예를 들어서

```
int f1()
{
}               /* 1번 */
int f2()
{               /* 3번 */
}
int main()
{
            f1();
            f2();   /* 2번 */
            return 0;
}
================================================================
int f1()
{
..........................
36:         40 b5               pop         %pc                 /* 1번 */
}
int f2()
{
  38:       20 b4               push        %lr                 /* 3번 */
  3a:       60 b0               push        %R5,%R6
  3c:       d6 e1               lea         (%SP),%R6
0000003e <.LM4>:
  3e:       c6 e1               lea         (%R6),%SP
  40:       60 b1               pop         %R5,%R6
  42:       40 b5               pop         %pc
00000044 <_main>:
main():
}
int main()
{
...........................
            f1();
  4c:       ef df               jal         2c <_f1>
            f2();
  4e:       f4 df               jal         38 <_f2>            /* 2번 */
```

위 같은 소스파일은 디버깅 할 때 1번에 정지된 상태에서 single 스텝을 하면 2번에 멈추는 것이 아니라 3번에서 멈추게 된다.