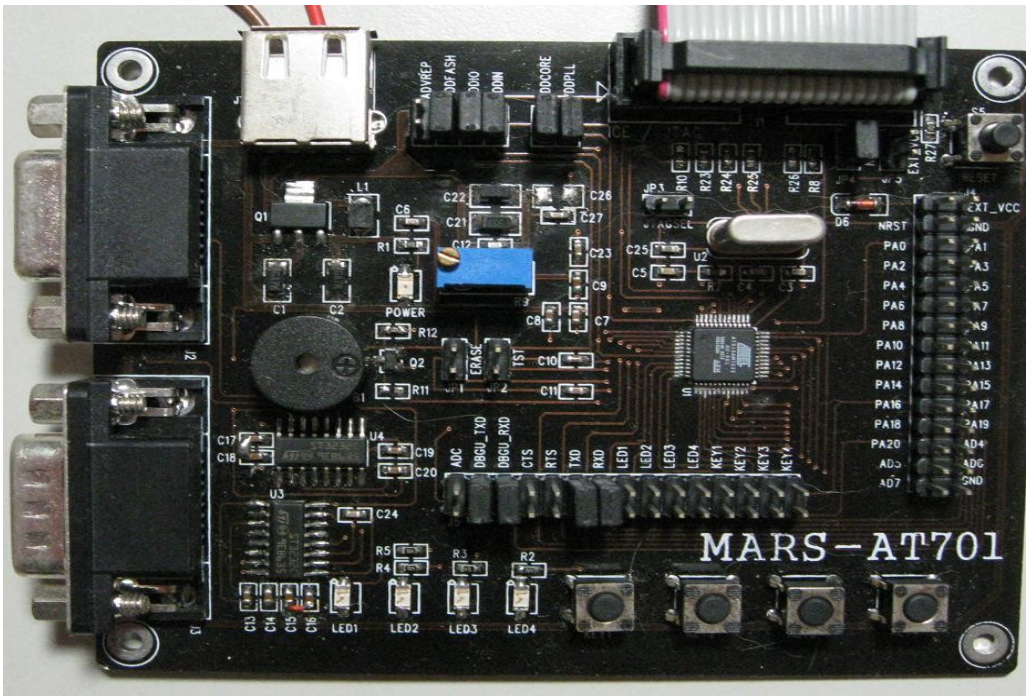
MARS-AT701 快速開發以及使用手冊



貓咪工作室

2007.9.1

序:

本產品以 Open Source 的工具來完成軟體開發，讓使用者在開發的過程中，可以節省軟體開發費用，也可以對嵌入式系統的軟體開發有更進一步的了解·

Ch1-1: MARS-AT701 軟體開發系統

古人有句話說的好"工欲善其事 必先利其器"，所以我們先來了解開發時所需要用到的軟體工具，並在稍後的章節再來一步步的介紹工具的安裝過程與操作步驟!

表 1-1 以下是所需要的軟體列表

工具名稱	描述
NOTEPAD++ (http://notepad-plus.sourceforge.net/tw/site.htm)	文字編輯器
Cygwin(http://www.cygwin.com/)	編譯的過程，需要使用到他的 shell 以及一些工具程式(如 make , grep....)
GNU ARM **Cross Tool** Chain (http://www.yagarto.de/)	各位需要用到的編譯工具組 包含編譯器(GCC)等等...
Open OCD (http://www.yagarto.de/)	DEBUG 所需要的軟體工具·
H-JTAG (http://www.hjtag.com/)	燒錄 Image 到 flash 的工具

Cross Tool 說明: 我們在建構編譯器時，是以 Build **系統**/ Host **系統**/ Target **系統** 這三個系統的不同點來作爲編譯器的分類點，因爲編譯器從建構到使用的過程中，可以在不同的系統上執行· **Build** 指的是建構編譯器的系統，**Host** 則是編譯器被執行的系統，最後的 **Target** 則是指機器碼(編譯器所產生出來的機器碼)被執行的地方 例如 X86 PC 或者是 ARM 開發板·

表 1-2 編譯器類型

Build	Host	Target	編譯器類型	結果
x86	x86	x86	Native	在 x86 上建構和執行，爲 x86 產生執行檔
x86	x86	ARM	Cross	在 x86 上建構和執行，爲 ARM 產生執行檔

MARS-AT701 的軟體開發流程(如圖 1-1)，是先在 WINDOWS 的編輯器上編寫完程式碼(註解 1)，再把寫好的程式碼放置在 Cygwin 的工作目錄下(先在 Cygwin 的 root 下建立好自己的工作目錄，這樣比較好方便做事)，最後在 Cygwin 的 command mode 下(如圖 1-2)，執行 GNU Tool chain 的各個工具，直到產生出最後的 image， 再透過 H-JTAG 把 image 燒錄到板子上的 flash(內建在 ATMEL SAM7SXXX 的晶片內)內，關開電源即可驗證成果!

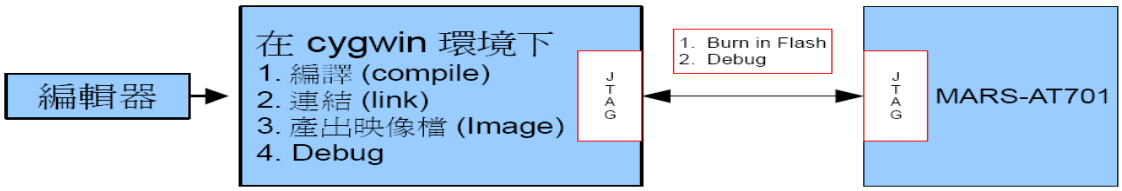


圖 1-1

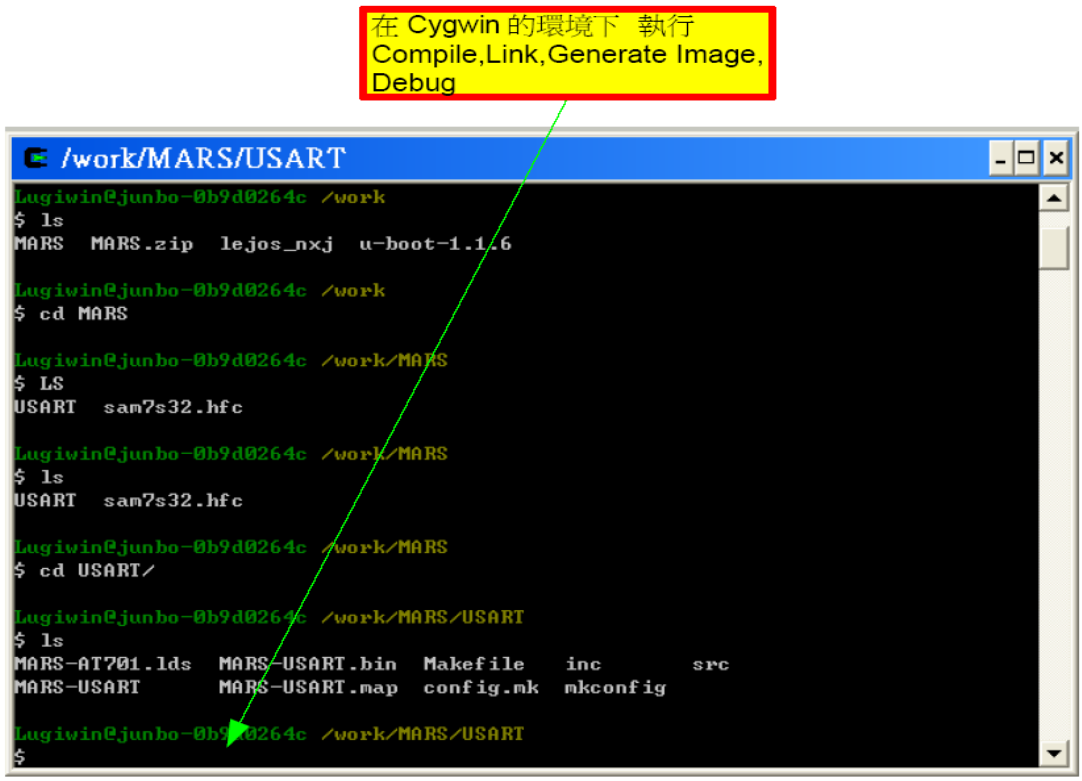


圖 1-2

MARS-AT701 軟體開發上所使用到的工具，來自於自由軟體 GNU(http://www.gnu.org/) 所開發的眾多軟體計畫中的其中幾個，使用者如果想要徹底發揮工具的威力，可以線上瀏覽開發工具的相關文件檔·

GNU Tool chain for ARM 所編譯出來的程式，其執行的效能和程式碼密度已經非常接近商業化軟體，如 ARM 公司出的軟體 Real view，或者是 KEIL ，IAR 等等· 如果使用者想使用 IDE 介面的工具組，可以使用 Eclipse IDE 這套自由軟體(以後會介紹)，所以沒有必要去弄套盜版軟體來開發·

註解 1: 微軟的文字檔中會以 **CRLF** 當作換行字元(0xD,0xA)，而 GNU 採用的 POSIX 的系統是以 **LF** 當作換行字元(0xA)，所以在編譯的過程中如果出現換行字元的 error，請先在 project 的目錄下(例如 /work/MATS/USART)Key '**make dos2unix**'，按下 ENTER 由 Makefile 去處理格式轉換的動作·

PS:

GNU 這個計畫的創始者 Richard M.Stallman 對自由軟體的精神有一套定義·

自由軟體 並不代表他是免錢的，指的是程式設計師可以**自由的去使用** 甚至**更改**成更適合自己使用的軟體，也可以**自由的散佈**當初下載的自由軟體或者是**自己更改**後的軟體，使得其他人可以蒙受自由軟體的所帶來的利益(軟體更加穩定 功能更加強大)·

Ch1-2 MARS-AT701 開發流程

這個章節說明 MARS 軟體開發流程的架構及目的，如同上一節所講的我們先透過文字編輯器，把 C 程式碼(*.c) 組合語言程式碼 (*.s) 以及 Include Files(*.h)，還有在 Link 期間所需要用到的 linker script(用來控制 Linker (arm-elf-ld.exe))建立好，最後再加上個 Make file ，讓程式碼可以被自動的從編譯到連結出可在 MARS-AT701 開發板上被執行的 Binary 檔·

Make 可以完成軟體自動化的建構(總不可能從編譯到連結，其間的每個步驟都得自己親手下 command 吧!)，而 Make file 則是控制 Make 所需要用到的 Script·

總結:如圖 1-3 先透過文字編輯器把所需要的 *.c ,*.s,*.h,Makefile,linker script 準備好，這樣才可以進行下一步·

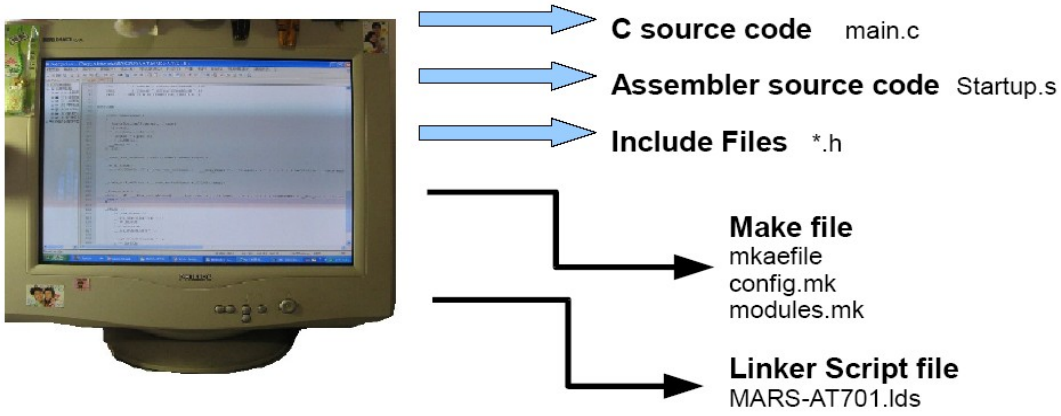


圖 1-3

Ch1-2.1 軟體開發流程

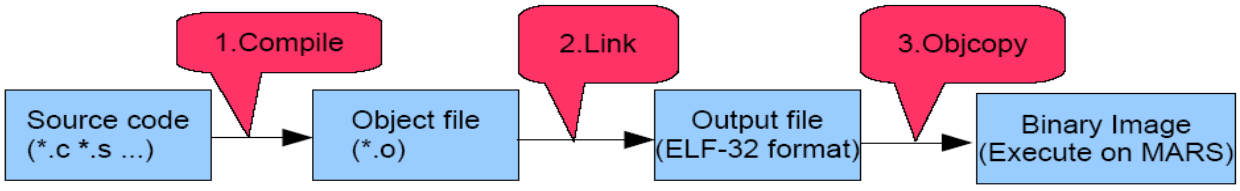


圖 1-4

從圖 1-4 中可以了解到 MARS 映像檔(Binary image)的建構流程，先說明第一道手續 **Compile**，透過 GNU ARM Tool Chain 的 GCC(GNU C Compiler)來編譯 MARS AT701 上的範例程式，其輸出檔爲 Obj file ，這個輸出檔已經很接近最後的 ARM 機器碼，但是其中的 function symbol 或者是 data symbol 還未決定好**被使用**時所存取的位址·

如圖 1-5 所示透過 GCC 把所有的 Source code 編譯成 obj 檔(未決定好 symbol 的位址)，symbol 所存放的位址，要到下一個步驟 Link 時才做決定。

圖 1-5

第二道手續 Link，如圖 1-6 GNU Linker 負責把上一步所產生 obj 檔，再加上其他的 obj 模組從 library 拉出來，再把這些 obj 檔全部組合起來，一併解決 symbol 的位址問題(只有在建構板子上所執行的 Binary 檔時，才可以設定各個 symbol 所存放的位址)，最後產生指定格式的 output file(MARS 是採用 ELF32 little ending 的格式)，其內含很多資訊(如 debug information)，所以 linker 所產生出來的輸出檔無法立即在板子上執行，還需要再經過下一道處理，才能淬取出可以燒到板子上的映像檔(***.bin**)，Linker 所產生的 output file 在往後的 debug 過程中是個必需品(Debug 軟體需要透過它得知各個 symbol 的資訊，以及讓使用者可以方便參照 C source code，而所需要的程式碼存放路徑)。

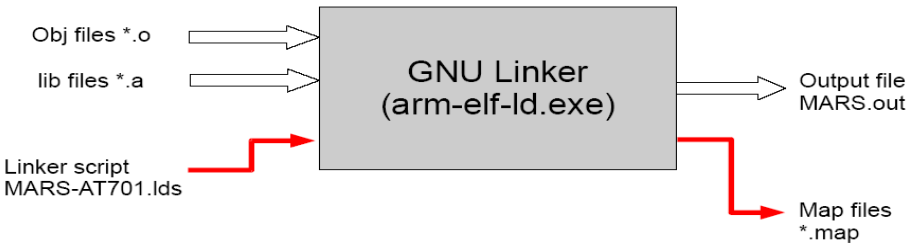


圖 1-6

第三道手續 Objcopy 則是爲了要從上一步的輸出檔中，取出可以被 MARS 上的 ATMEL SAM7SXXX 所執行的 Binary 檔(***.bin**)，而這個輸出檔需要透過 H-JTAG 經由 JTAG 介面，燒錄到 ATMEL SAM7SXXX 內部的 Flash 上。

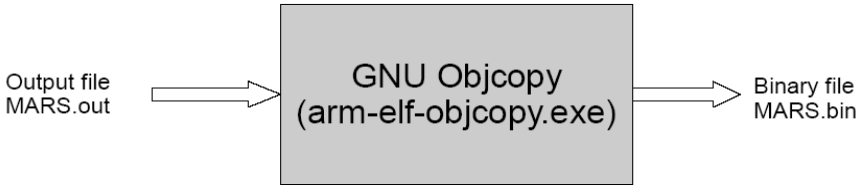


圖 1-7

在稍後的章節我們會詳細的介紹開發和安裝的流程，現在就先以圖示來了解基本的 build code 過程，以便對剛剛所講過的流程有一定的概念·

我們以 USART 這個 Sample Project 來做範例，本產品的軟體開發是在 Cygwin 的環境下所完成的，如圖 1-8 在 shell 下執行 Make 這支程式，讓 USART 這個 Project 的所有程式碼可以自動轉換成在 MARS AT701 上被執行的 Binary 映像檔，輸入 make 後可以發現 Cygwin 的 shell 吐出一些訊息(這些訊息是在 Makefile 內所定義的動作)，其中 all 就是 build 出 Binary 映像檔所要選的選項，接下來如圖 1-9 輸入 **make all** ，圖 1-10 可以發現已經在原來的檔案下，，產生出 MARS-USART 這個 ELF 格式的輸出檔(圖 1-4 **Link** 後所產生的檔案)，以及 MARS-USART.bin(圖 1-4 **Objcopy** 後所產生的檔案) 這支 Binary 映像檔 ，接下來的章節就介紹如何下載 MARS-USART.bin 到開發板上，以及透過 Debug 軟體參考 MARS-USART 後對系統的 Debug 流程·

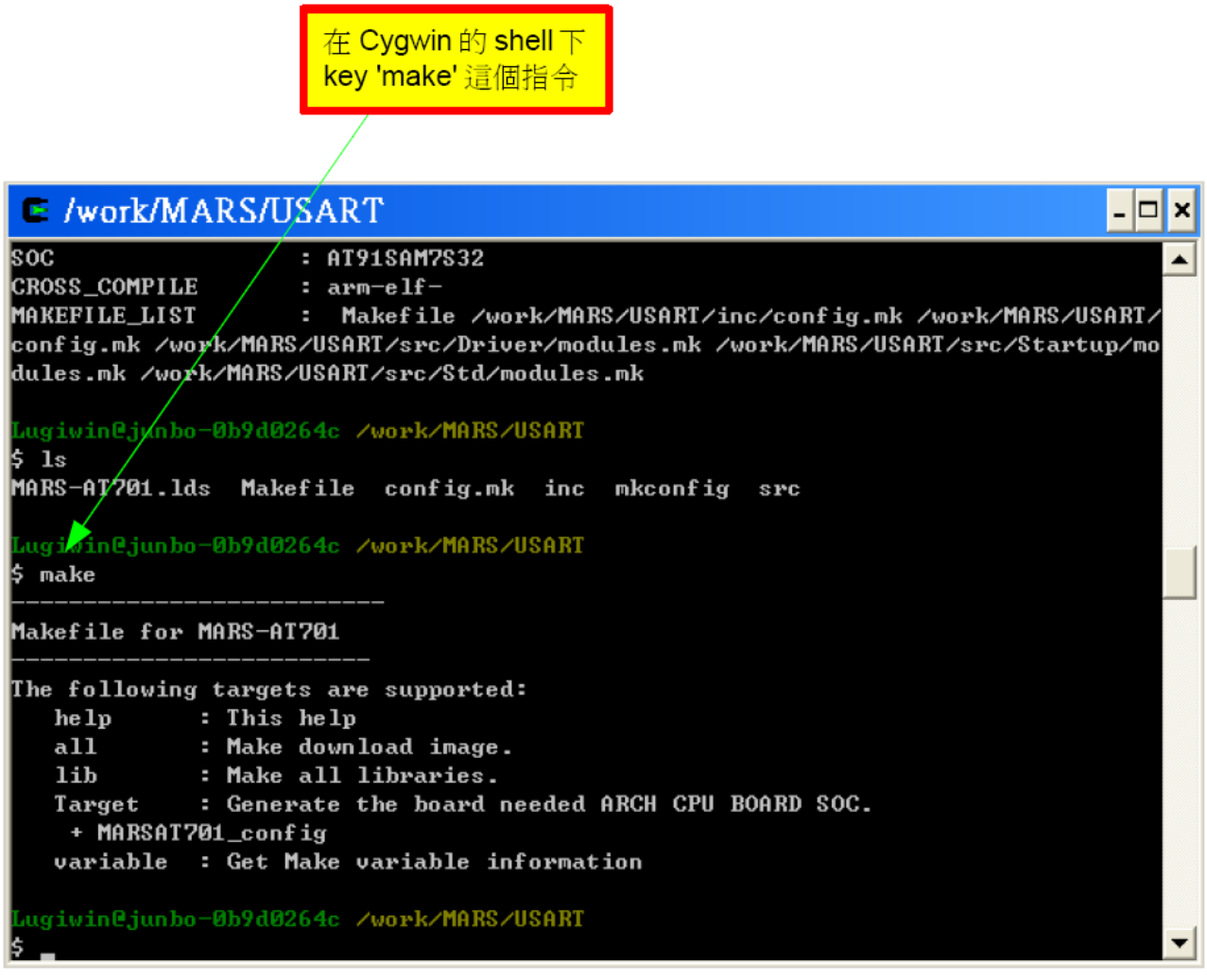


圖 1-8

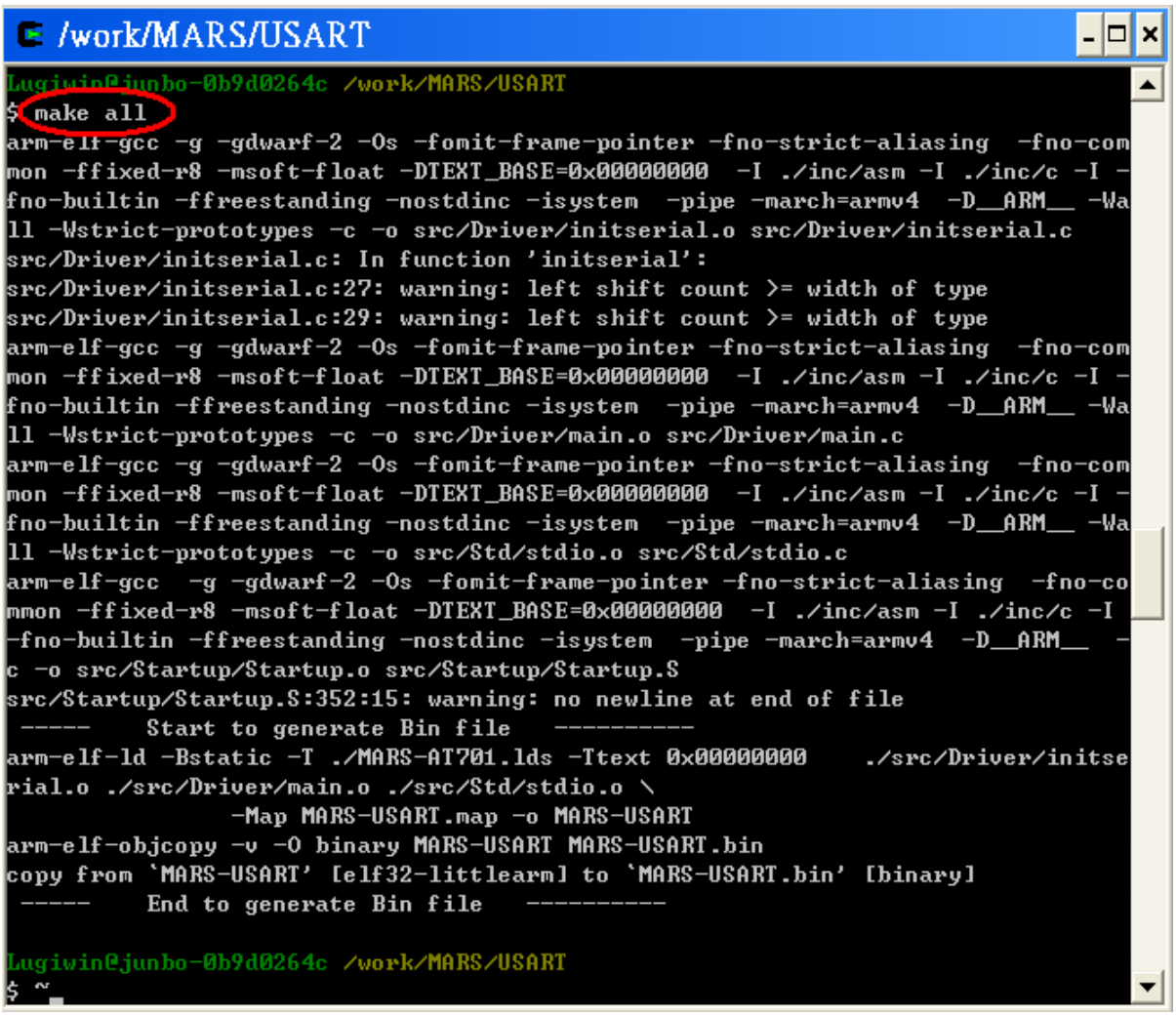

/work/MARS/USART
Lugiwin@junbo-0b9d0264c /work/MARS/USART
$ make all
arm-elf-gcc -g -gdwarf-2 -Os -fomit-frame-pointer -fno-strict-aliasing -fno-com
mon -ffixed-r8 -msoft-float -DTEXT_BASE=0x00000000 -I ./inc/asm -I ./inc/c -I -
fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -Wa
ll -Wstrict-prototypes -c -o src/Driver/initserial.o src/Driver/initserial.c
src/Driver/initserial.c: In function 'initserial':
src/Driver/initserial.c:27: warning: left shift count >= width of type
src/Driver/initserial.c:29: warning: left shift count >= width of type
arm-elf-gcc -g -gdwarf-2 -Os -fomit-frame-pointer -fno-strict-aliasing -fno-com
mon -ffixed-r8 -msoft-float -DTEXT_BASE=0x00000000 -I ./inc/asm -I ./inc/c -I -
fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -Wa
ll -Wstrict-prototypes -c -o src/Driver/main.o src/Driver/main.c
arm-elf-gcc -g -gdwarf-2 -Os -fomit-frame-pointer -fno-strict-aliasing -fno-com
mon -ffixed-r8 -msoft-float -DTEXT_BASE=0x00000000 -I ./inc/asm -I ./inc/c -I -
fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -Wa
ll -Wstrict-prototypes -c -o src/Std/stdio.o src/Std/stdio.c
arm-elf-gcc -g -gdwarf-2 -Os -fomit-frame-pointer -fno-strict-aliasing -fno-co
mmon -ffixed-r8 -msoft-float -DTEXT_BASE=0x00000000 -I ./inc/asm -I ./inc/c -I
-fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -
c -o src/Startup/Startup.o src/Startup/Startup.S
src/Startup/Startup.S:352:15: warning: no newline at end of file
----- Start to generate Bin file ----------
arm-elf-ld -Bstatic -T ./MARS-AT701.lds -Ttext 0x00000000 ./src/Driver/initse
rial.o ./src/Driver/main.o ./src/Std/stdio.o \
-Map MARS-USART.map -o MARS-USART
arm-elf-objcopy -v -O binary MARS-USART MARS-USART.bin
copy from `MARS-USART' [elf32-littlearm] to `MARS-USART.bin' [binary]
----- End to generate Bin file ----------
Lugiwin@junbo-0b9d0264c /work/MARS/USART
$ ~

圖 1-9

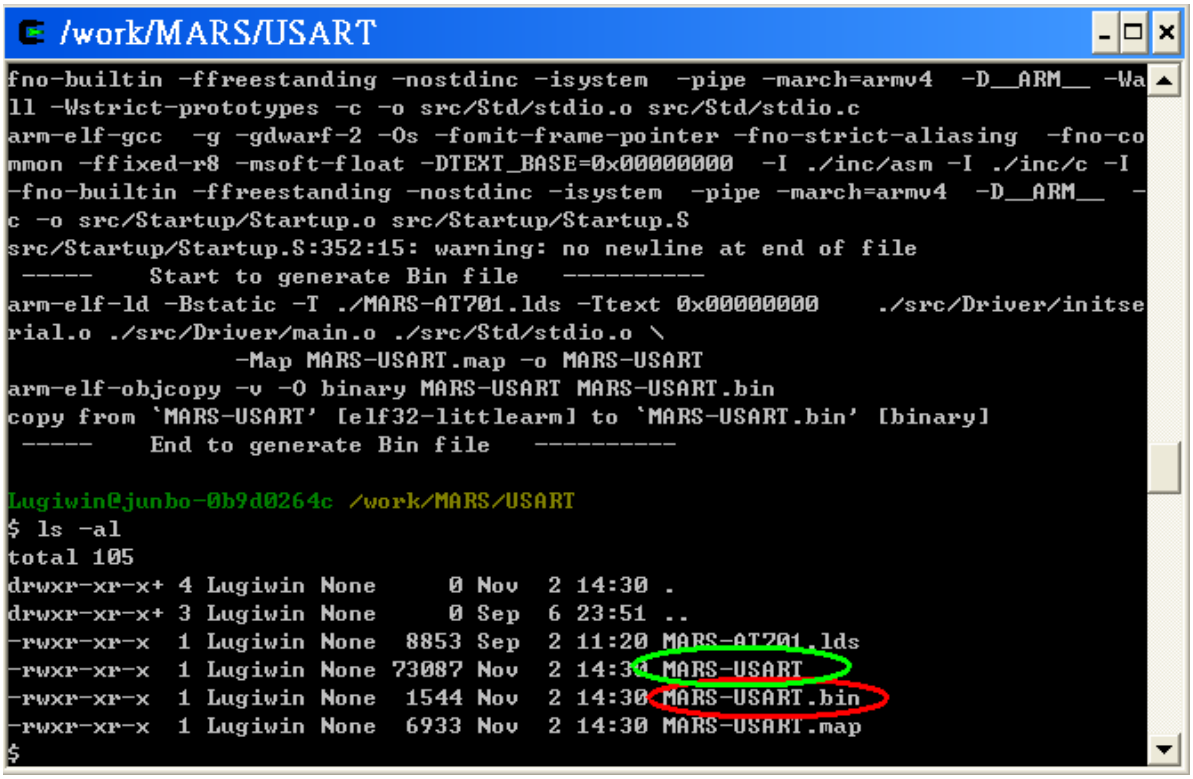

/work/MARS/USART
fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -Wa
ll -Wstrict-prototypes -c -o src/Std/stdio.o src/Std/stdio.c
arm-elf-gcc -g -gdwarf-2 -Os -fomit-frame-pointer -fno-strict-aliasing -fno-co
mmon -ffixed-r8 -msoft-float -DTEXT_BASE=0x00000000 -I ./inc/asm -I ./inc/c -I
-fno-builtin -ffreestanding -nostdinc -isystem -pipe -march=armv4 -D__ARM__ -
c -o src/Startup/Startup.o src/Startup/Startup.S
src/Startup/Startup.S:352:15: warning: no newline at end of file
----- Start to generate Bin file ----------
arm-elf-ld -Bstatic -T ./MARS-AT701.lds -Ttext 0x00000000 ./src/Driver/initse
rial.o ./src/Driver/main.o ./src/Std/stdio.o \
-Map MARS-USART.map -o MARS-USART
arm-elf-objcopy -v -O binary MARS-USART MARS-USART.bin
copy from `MARS-USART' [elf32-littlearm] to `MARS-USART.bin' [binary]
----- End to generate Bin file ----------
Lugiwin@junbo-0b9d0264c /work/MARS/USART
$ ls -al
total 105
drwxr-xr-x+ 4 Lugiwin None 0 Nov 2 14:30 .
drwxr-xr-x+ 3 Lugiwin None 0 Sep 6 23:51 ..
-rwxr-xr-x 1 Lugiwin None 8853 Sep 2 11:20 MARS-AT701.lds
-rwxr-xr-x 1 Lugiwin None 73087 Nov 2 14:30 MARS-USART
-rwxr-xr-x 1 Lugiwin None 1544 Nov 2 14:30 MARS-USART.bin
-rwxr-xr-x 1 Lugiwin None 6933 Nov 2 14:30 MARS-USART.map
$

圖 1-10

Ch1-2.2 燒錄以及軟體 Debug 流程

這章節講述燒錄以及 Debug 的流程，先以燒錄開始，燒錄 Image 的軟體工具，在這裡採用 H-FLASH,Open-OCD 這兩套燒錄軟體，其中 H-FLASH 有比較漂亮的使用者介面，如圖 1-11 先把 Wiggler JTAG 小板和開發板以及 PC 上的 PrintPort 串接好，再透過 H-JTAG 這支程式辨認到 MARS AT701 上的 ATMEL Microprocessor，辨識的方法是經由 H-JTAG 這支程式，透過 JTAG 的介面傳送要求 Device ID 的 command 到 Microprocessor 內部的 Tap controller，接下來 Tap controller 把 Device 的 ID 透過 JTAG 的 TDO 回傳到 PC 上，H-JTAG 得到後再經由 UI 讓使用者知道，ATMEL Microprocessor 已經偵測到且 JTAG 的連線也無問題·

偵測到 Microprocessor 後再開啓 H-FLASH 如圖 1-12 透過 JTAG 把 Image 燒錄到 Microprocessor 內部的 flash，燒錄的方式是透過 TAP Controller 選擇好 Boundary scan Chain 的 ScanChain0，讓這個 chain 的輸入輸出對應到 TDI TDO 上，經由這個包圍在內部核心的 Boundary scan Chain，控制 ARM Core 讓 ARM Core 執行燒錄 Image 的動作·

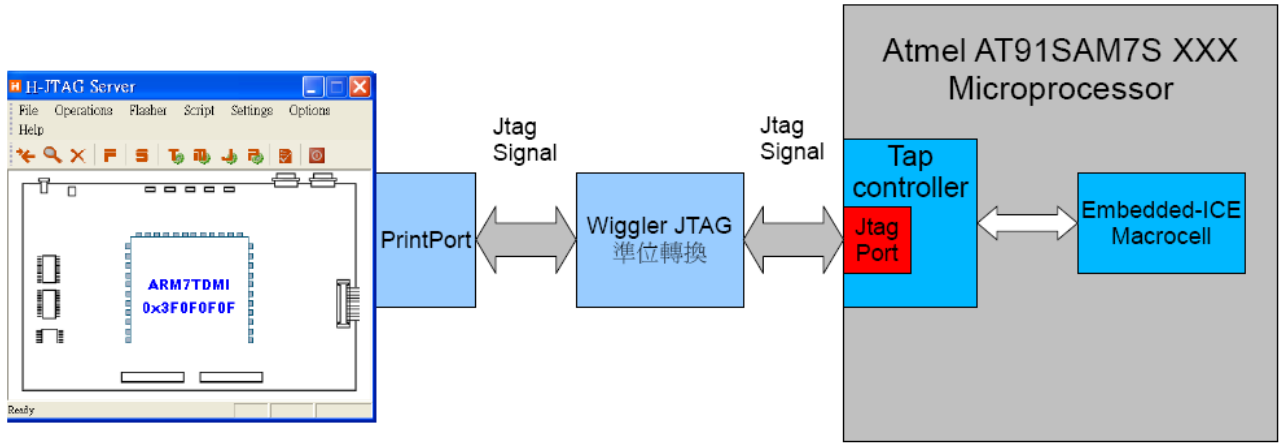


圖 1-11

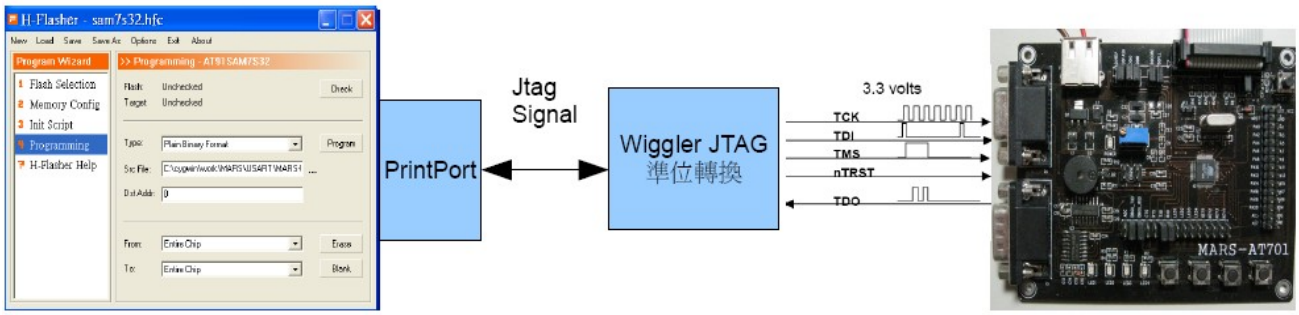


圖 1-12

Debug 的方式和燒錄的方式有點不太一樣，在軟體部分則需要用到 Yagarto 的 arm-elf-insight.exe 作 source code 層級的 debug, 以及 OpenOCD 的 openocd-pp.exe 來控制 JTAG 動作，在硬體方面則是需要再多用到另一個 Boundary scan Chain，控制 Microprocessor 內部的 Embedded ICE，實作出 HW Breakpoint & Watchpoint·

圖 1-13 顯示出 MARS AT701 Debug 的架構圖，ARM-ELF-INSIGHT.EXE 是個圖形化介面的工具如圖 1-14，Debug 的時候透過這個介面達到 Trace source code 的功能，是個 source code level 的 debug 工具，Open-OCD-PP.exe 則是屬於 Debug Server 導向的工具，他負責處理 JTAG 的訊號以及回應 Debug Client 的請求，其中 Client 和 Server 的溝通則是透過 TCP/IP 的 Protocol 來完成·

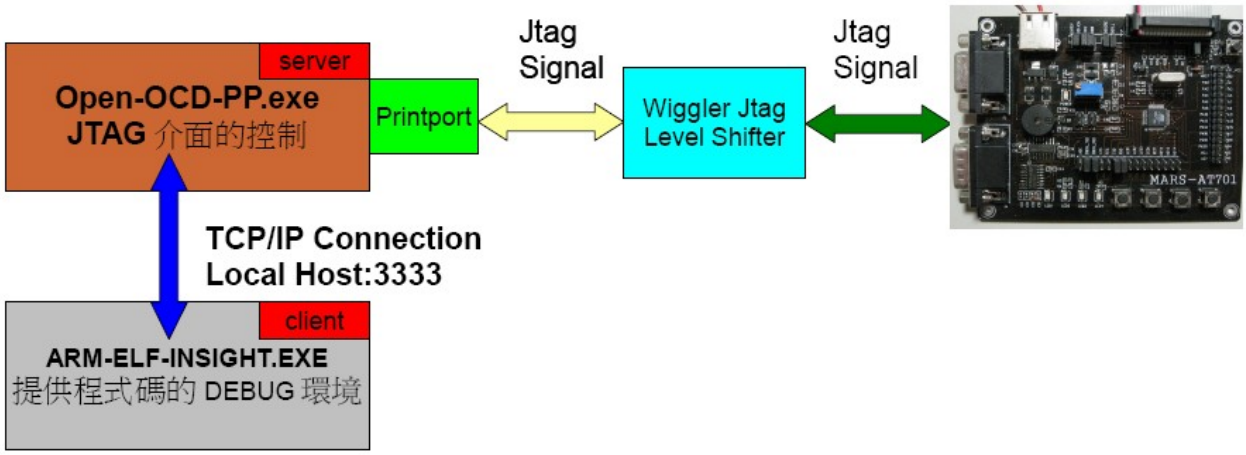


圖 1-13

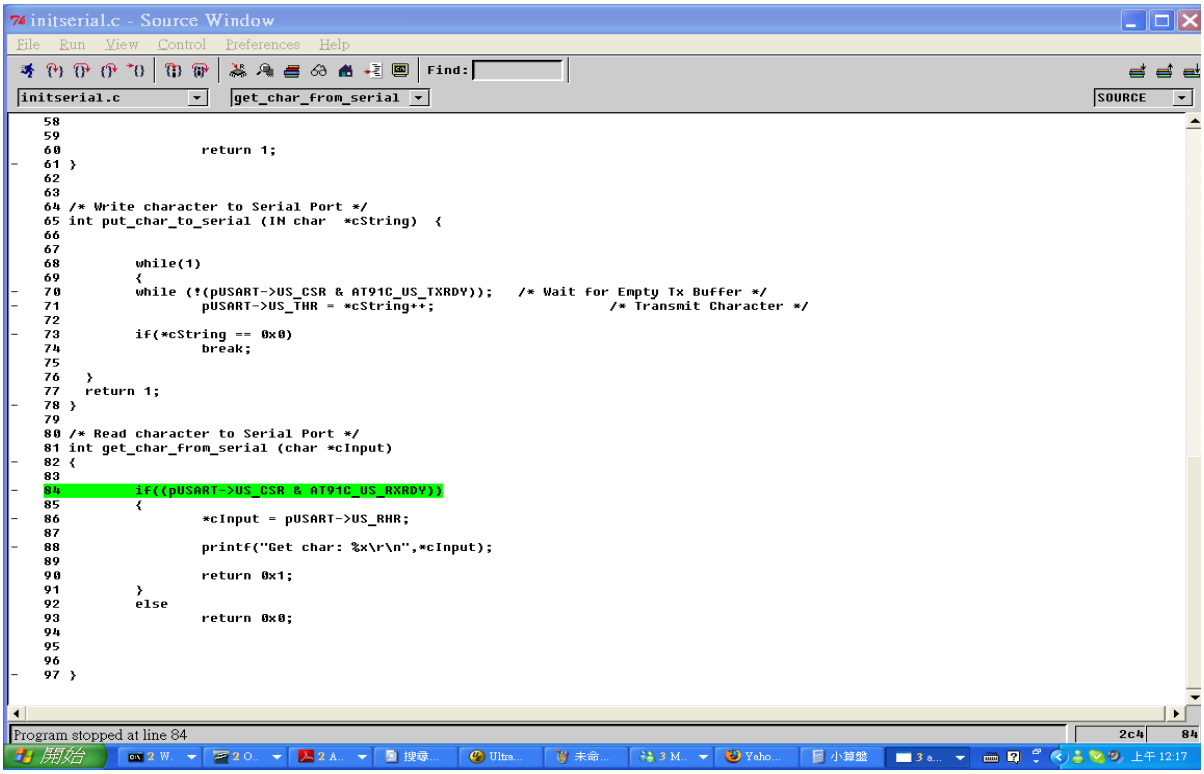


圖 1-14

ARM-ELF-INSIGHT.EXE 的核心是 GNU Debugger(**GDB**)，和 GDB 不同的地方就是他有個很好使用的圖形化使用者介面如圖 1-15·

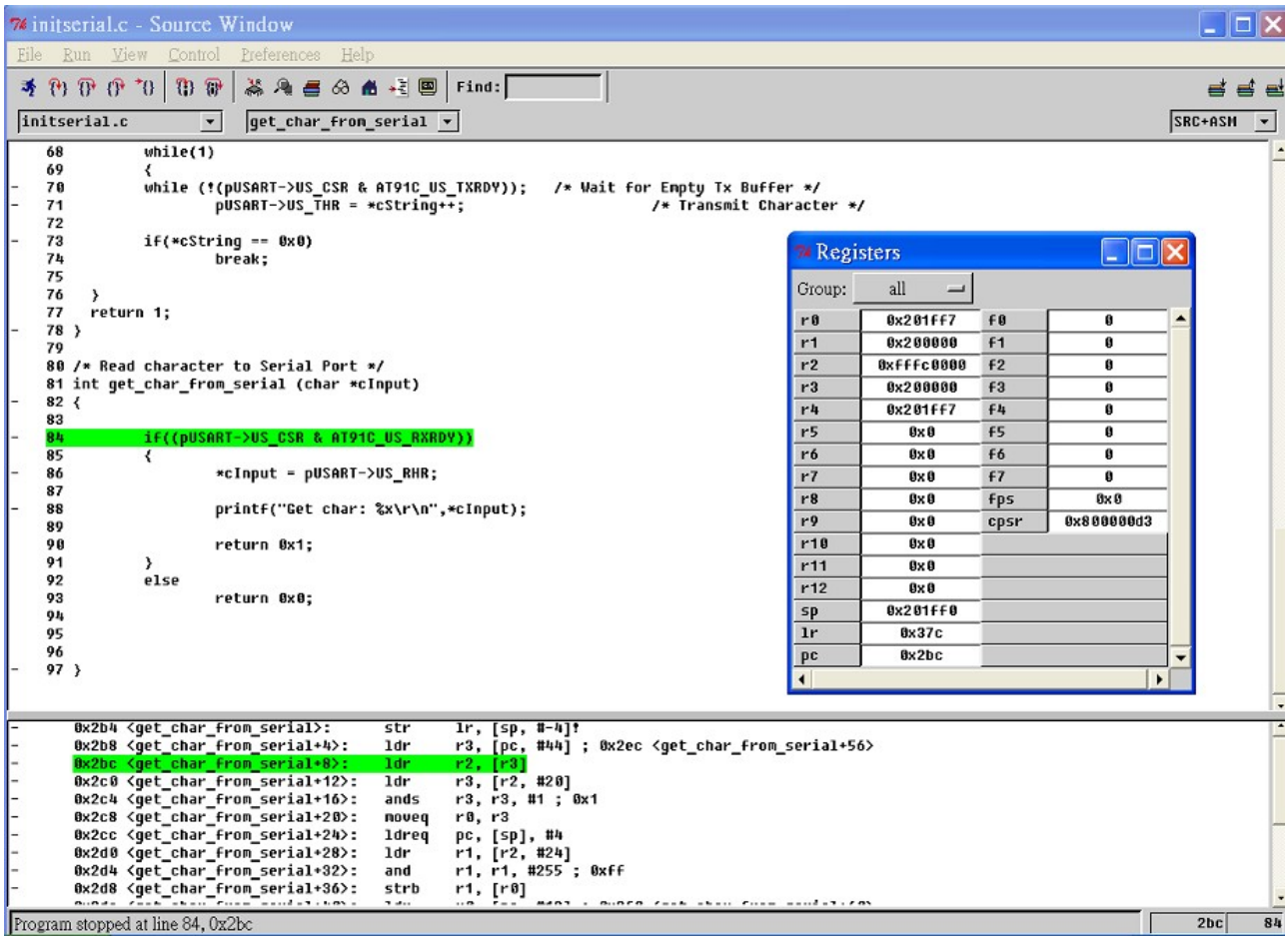


圖 1-15

而在開始 debug 的時候需要參考 Linker 所產生的輸出檔(MARS-USART: 如圖 1-10 make all 後所產生的輸出檔)，這個輸出檔不僅包含著機器碼也包含著 symbol 的資訊，例如 function 被執行的位址或者是所在的程式碼檔案，有了這些資訊可以讓 debug 時不會因爲看到一堆機器碼而煩惱，因爲 ARM-ELF-INSIGHT.EXE 可以參照這些資訊，讓使用者參考到正確的 C Source code，如圖 1-15 我們可以看到上半部的視窗是 C Source code，下半部則全是機器碼，綠色的 bar 就是 ARM 正要執行程式碼·

在 Debug 的過程中會按照 GDB RSP(Remote Serial Protocol) ，傳送 Debug 的 request 到 Server 端(Open-OCD-PP.EXE)，如圖 1-16 當 Client 要從 0x200 的位置讀取 8byte 的資料時，就會透過 TCP 傳送一段文字給 Server(如圖 1-16 所示 $0x200, 8 #cs)，Server 收到時也會按照

RSP 透過 TCP 回傳給 GDB Client(ARM-ELF-INSIGHT.EXE)·

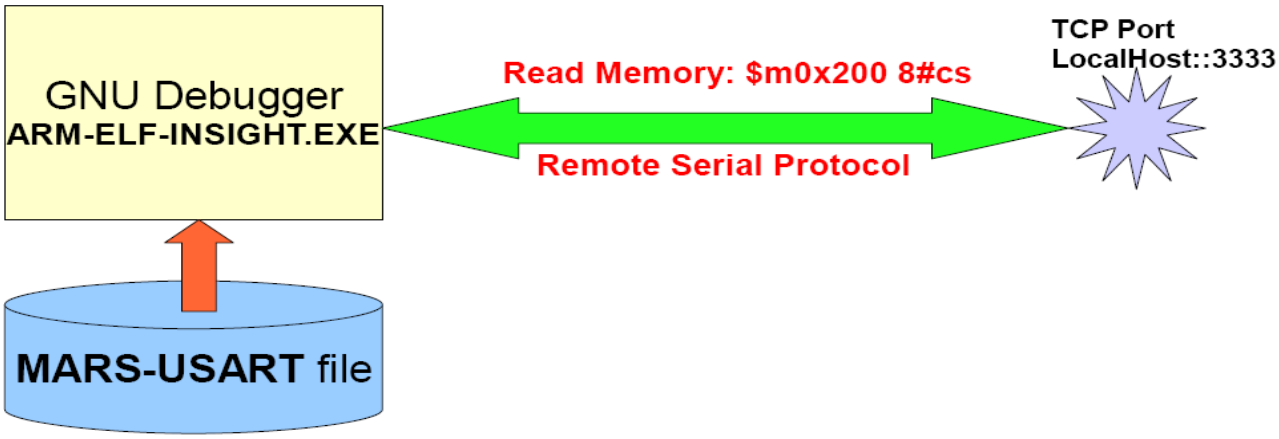


圖 1-16

圖 1-17 就是使用者所面對的開發板，Wiggle JTAG 透過排線和開發板連接在一起，開發的時候 Wiggler 是透過印表機 Cable 和電腦連結在一起，而板子離 USB 較遠的 RS232 插槽則透過 RS232 Cable 和電腦的 RS232 連結埠連結在一起·

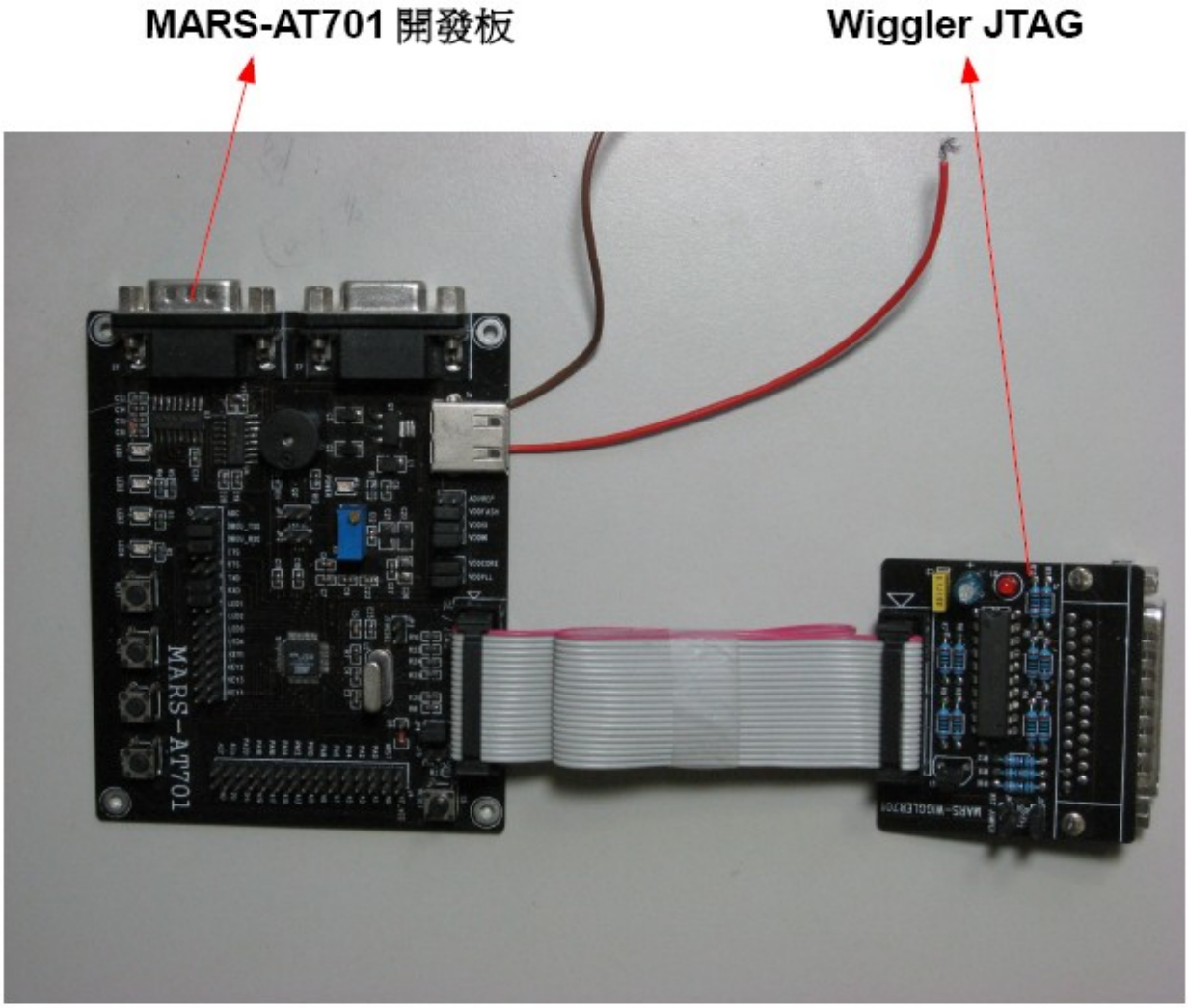


圖 1-17 MARS-AT701 開發板 以及 Wiggler JTAG 實體圖