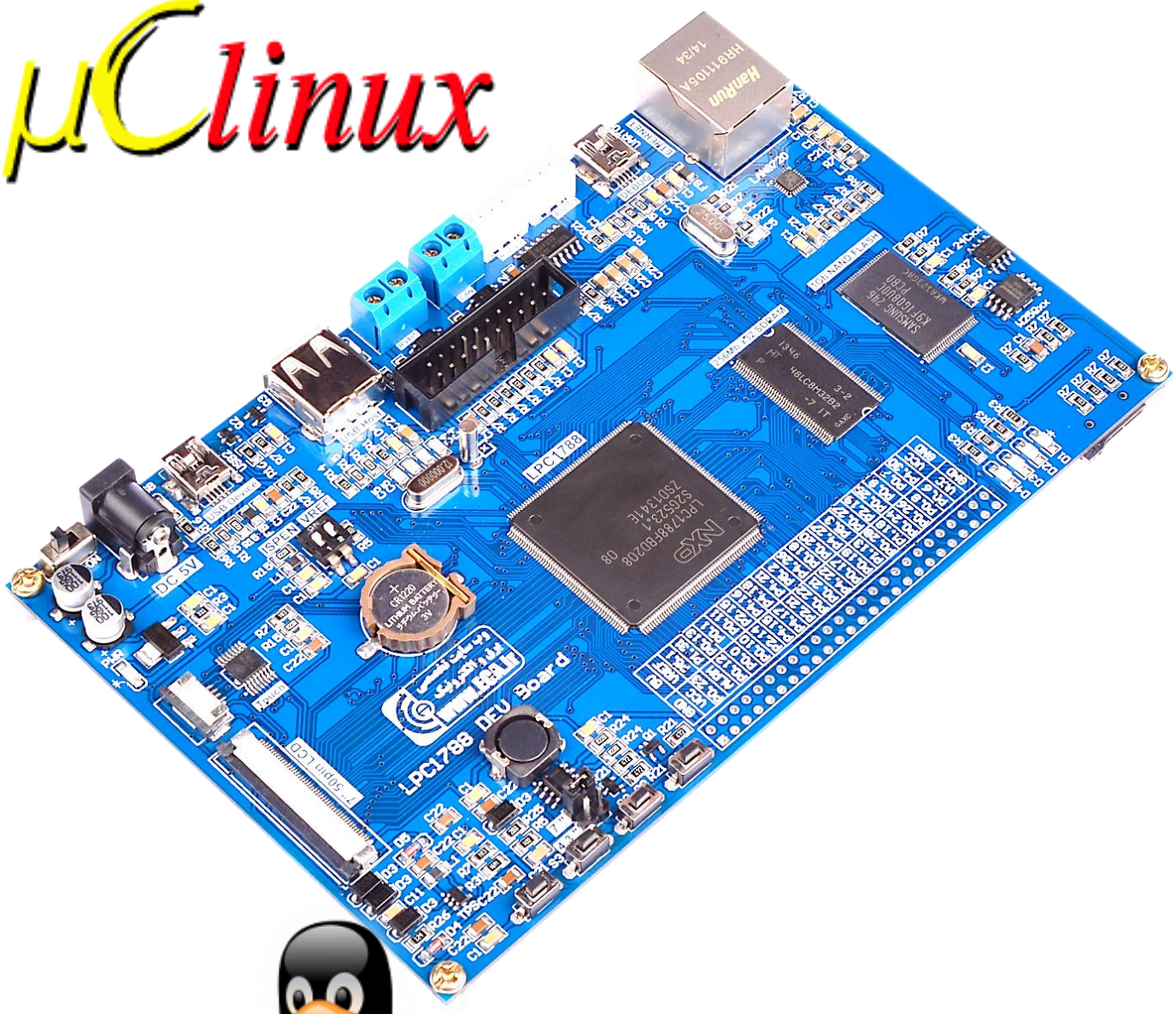
لینوکس Cortex-M

راهنمای بسته پشتیبانی برد (BSP) برای

برد کاربردی-حرفه ای ECA LPC1788



Linux

فهرست مطالب

1. بررسی اجمالی

این مطلب راهنمای بسته پیشتیبانی برد (BSP) سیستم عامل یو سی لینوکس برای برد کاربردی–صنعتی LPC1788 شرکت ECA می‌باشد.

بسته پشتیبانی برد، یک محیط توسعه نرم افزار برای آموزش و توسعه لینوکس بر روی پردازنده Cortex-M3 میکروکنترلر LPC1788 با استفاده از برد کاربردی–صنعتی ECA به عنوان پلتفرم سخت افزاری را فراهم می‌کند.

2. فایل های موجود در بسته

فایل های لیست شده در زیر به عنوان محتویات این بسته می‌باشند که از طریق وب سایت شرکت ECA قابل تهیه می‌باشند.

- eca-u-boot-1788.hex : فایل اجرایی از پیش آماده شده بوت لودر برای نصب بر روی حافظه Flash داخلی میکروکنترلر LPC177
- hello.uImage : فایل ایمیج پروژه نمونه تولید شده برای تست
- linux-ECA-LPC1788-bspg.pdf : راهنمای بسته پشتیبانی برد
- linux-ECA-1788.tar.bz2 : محیط توسعه نرم افزاری Linux برای LPC17xx شامل :

ا) هسته لینوکس

ب) busybox و سایر اجزای مورد نیاز

پ) محیط توسعه متقابل مبتنی بر لینوکس

ت) سکوی توسعه پروژه های مبتنی بر لینوکس (کاربردهای نهفته) برای شروع سریع شامل برنامه نمونه

ث) کامپایلر پردازنده های ARM برای لینوکس

3. ویژگی های نرم افزاری

در لیست زیر ویژگی ها و ظرفیت های لینوکس LPC17XX جمع بندی شده است

3.1 فریمور U-Boot

- U-Boot نسخه v2010.03
- اجرا از حافظه فلش و SRAM داخلی میکروکنترلر بدون نیاز به حافظه خارجی
- کنسول سریال
- درایور شبکه برای آپلود ایمیج روی برد
- درایور سریال برای آپلود ایمیج به برد
- درایور فلش داخلی میکروکنترلر برای قابلیت های خود ارتقایی
- درایور برای ذخیره پیکربندی روی حافظه فلش سریال خارجی

- قابلیت بوت خودکار برای بوت شدن ایمیج سیستم عامل بدون دخالت خارجی

3.2 لینوکس

- هسته uClinux v2.6.33
- امکان بوت از ایمیج فشرده و غیر فشرده
- قالبیت اجرای کد هسته بحرانی از روی حافظه Flash داخلی LPC17XX
- درایور سریال و کنسول Linux
- درایور اترنت و شبکه بندی
- Busybox v1.17
- پشتیبانی از استاندارد POSIX pthreads
- حافظت پروسس به هسته و پروسس به پرسس با استفاده از واحد MPU هسته LPC17XX
- قابلیت لود ماژول های هسته
- شل ایمن ssh
- وب سرور
- پارتیشن بندی مبتنی بر MTD برای فلش های خارجی
- درایور برای رابط DMA
- درایور برای واحد USB Host
- درایور برای Framebuffer
- درایور برای کارت حافظه SD
- پشتیبانی از RS485 در درایور سریال

4. راه اندازی سخت افزار

برد کاربردی–صنعتی LPC1788 شرکت ECA پلتفرم سخت افزاری مورد نیاز برای پیاده سازی، توسعه و آموزش لینوکس روی هسته Cortex M3 میکروکنترلر LPC1788 را فراهم می‌کند. در این بخش نحوه آماده سازی برد کاربردی–صنعتی LPC1788 برای اجرای سیستم عامل لینوکس شرح داده می شود.

برای اجرای سیستم عامل لینوکس از حافظه SDRAM خارجی به ظرفیت 32 مگابایت و برای ذخیره ایمیج هسته لینوکس از حافظه فلش سریال W25Q32 استفاده می شود. ارتباط با بوت لودر و شل لینوکس نیز از طریق پورت سریال میکروکنترلر و مبدل USB به سریال روی برد انجام می شود که با برچسب DEBUG بر روی برد مشخص شده است. همچنین درایور های مبدل USB به سریال PL2303 نیز بایستی بر روی سیستم نصب باشند. وضعیت دیپ سوئیچ برد نیز باید به صورت زیر باشد:

دیپ سوئیچ	وضعیت	شرح
ISPEN	OFF	فقط برای ریختن بوت لودر با فلش مجیک روشن شده و دوباره قطع شود
VREF	ON	-

5. راه اندازی نرم افزار

5.1 نصب بوت لودر U-Boot

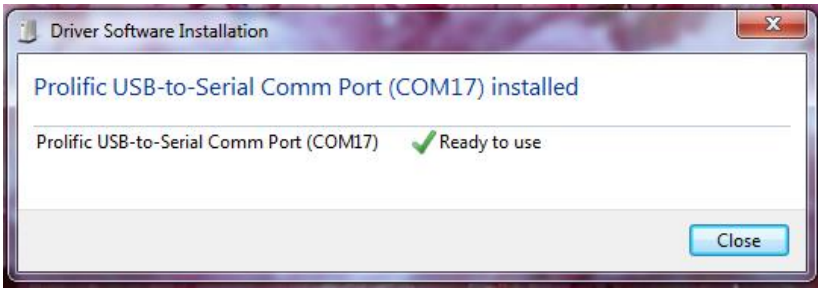
برای نصب بوت لودر بر روی حافظه فلش داخلی می‌توان از دو روش استفاده از پروگرامر J-Link و یا بوت لودر خود شرکت و روش ISP استفاده کرد که هر دو روش در زیر توضیح داده می شوند.

5.1.1 نصب از طریق بوت لودر و نرم افزار Flash Magic

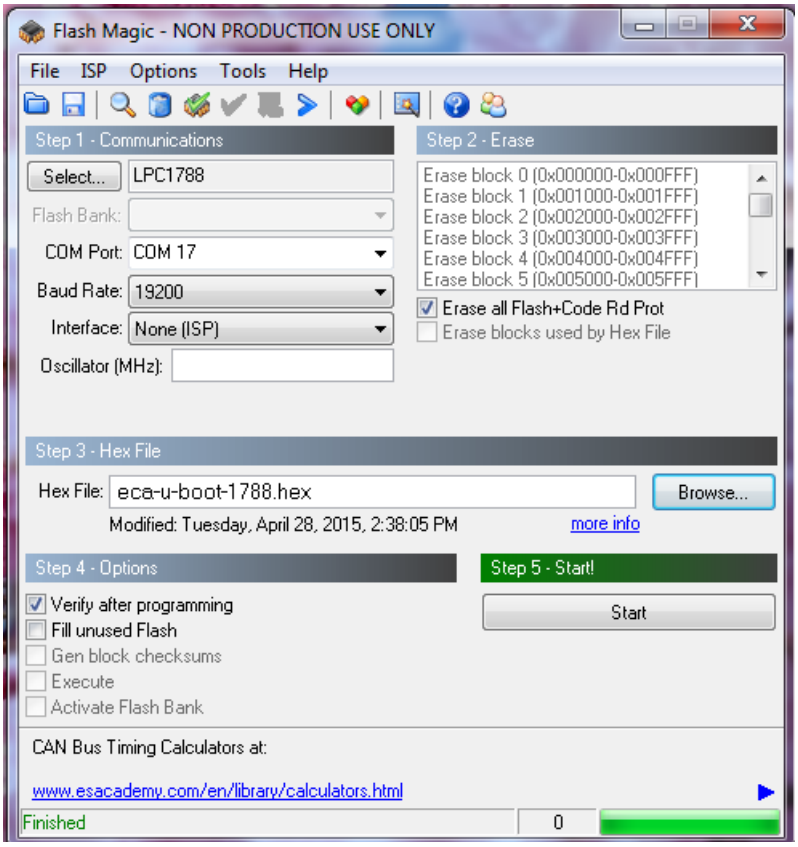
- آخرین نسخه نرم افزار FlashMagic را از دیسک همراه محصول یا وب سایت زیر تهیه و نصب نمائید.

http://www.flashmagictool.com/download.html&d=FlashMagic.exe

- فایل درایور PL2303_Prolific_DriverInstaller را نصب نمائید.
- دیپ سوئیچ ISPEN را فعال نموده و تغذیه برد را متصل نمائید.
- پورت USB بخش DEBUG (مبدل USB به سریال) را به کامپیوتر متصل نمائید. سیستم عامل میبایست دستگاه جدید را به عنوان پورت سریال شناسایی نماید.



- نرم افزار FlashMagic را اجرا کرده و مطابق شکل زیر ابتدا نوع میکروکنترلر و پورت اختصاص داده شده برای مبدل USB به سریال را مشخص نمایید.
- برای اطمینان از شماره پورت اختصاص یافته به بخش Device Manager مراجعه کنید.



- مسیر فایل eca-u-boot-1788.hex را با فشردن کلید Browse.. مشخص نمائید و برای بازبینی پروگرام صحیح میکروکنترلر تیک گزینه Verify after programming را بزنید.
- کلید Start را فشار دهید تا عملیات انتقال فایل شروع شود. پس از چند ثانیه پیغام سبز رنگ Finished نشان میدهد که عملیات به خوبی انجام شده است.

5.1.2 نحوه پروگرام کردن میکرو با استفاده از پروگرامر JLink

- آخرین نسخه نرم افزار JLink را از سایت Segger و یا دیسک همراه برد تهیه و نصب نمائید.
- برد آموزشی را به پروگرامر JLink متصل نموده و نرم افزار J-Flash را اجرا نمائید.
- از بخش Device میکروکنترلر LPC1788 را انتخاب کنید و کلاک را روی حالت Auto تنظیم نمائید.
- در حالیکه تغذیه برد را وصل کرده اید از تب Target گزینه Connect را بزنید. در صورتیکه عملیات اتصال به درستی انجام شود پیغام Connected successfully در بخش LOG نمایش داده می شود.
- حال میتوانید از بخش File فایل eca-u-boot-1788.hex را انتخاب کرده و با گزینه Program یا فشردن کلید F5 آن را بر روی میکروکنترلر پروگرام نمائید.

5.2 محیط U-Boot

بعد از پروگرام بوت لودر روی برد و ریست میکروکنترلر، U-Boot از روی فلش داخلی میکروکنترلر با ارسال خروجی زیر بر روی پورت سریال DEBUG (115200 bps) بالا می‌آید :

```
U-Boot 2010.03 (Aug 05 2015 - 07:04:57)

CPU  : LPC178x/7x series (Cortex-M3)
Freqs: SYSTICK=108MHz,EMCCLK=54MHz,PCLK=54MHz
Board: ECA LPC1788 Development Board rev 1
ECA R&D Team (Jafarpour@outlook.com)
www.ECA.ir
www.forum.ECA.ir
DRAM:  32 MB
Flash:  0 kB
In:    serial
Out:   serial
Err:   serial
Net:   LPC178X_MAC
Hit any key to stop autoboot:  0
ECA-DEV1788>
```

در صورتیکه با خطای "Bad CRC" در ترمینال مواجه شدید به این دلیل می‌باشد که U-Boot در حالت ذخیره تنظیمات بر روی فلش سریال پیکربندی شده است و چون اولین باری می‌باشد که U-Boot بالا می آید این تنظیمات بر روی حافظه فلش وجود ندارد. با یک بار ذخیره تنظیمات روی فلش این خطا رفع می‌گردد. ذخیره پیکربندی جاری به روز بر روی حافظه فلش سریال با استفاده از دستور saveenv انجام می‌شود. دقت نمایید که به دلیل استفاده از آدرس 0x0000 تا 0x0FFF از حافظه فلش سریال برای ذخیره پیکربندی U-Boot ایمیج های ذخیره شده در فلش سریال می‌بایست از آدرس 0x1000 به بعد قرار گیرند.

نحوه ذخیره پیکربندی بعدی فعلی U-Boot بر روی فلش سریال به صورت زیر می باشد:

```
ECA-DEV1788> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Erase: 20 00 00 00
Writing to SPI flash...done
ECA-DEV1788>
```

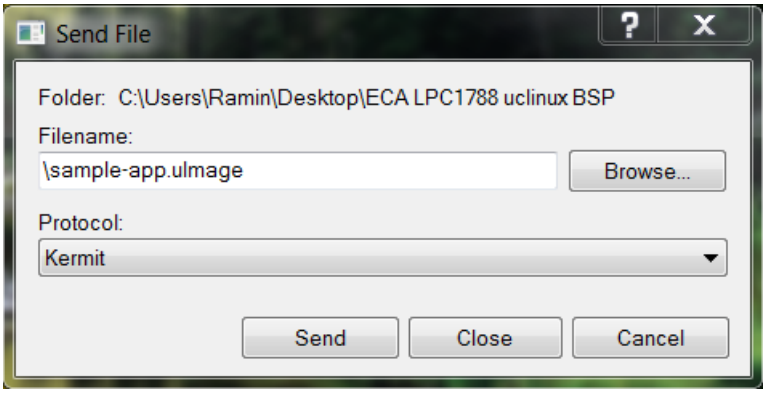
5.3انتقال ایمیج لینوکس به برد

برای انتقال ایمیج لینوکس به برد روشهای مختلفی نظیر انتقال از طریق شبکه اترنت، پورت سریال و ... وجود دارد که در اینجا از روش ساده استفاده از هایپرترمینال ویندوز و استاندارد Kermit استفاده شده است.

ابتدا با استفاده از دستور loadb بوت لودر را برای دریافت فایل ایمیج از طریق پورت سریال و ذخیره برو روی حافظه SDRAM برد (آدرس 0xA0000000) آماده میکنیم.

```
ECA-DEV1788> loadb
## Ready for binary (kermit) download to 0xA0000000 at 115200 bps...
```

حال از منوی Transfer نرم افزار هایپر ترمینال گزینه Send File را انتخاب کرده و فایل ایمیج مورد نظر را مشخص میکنیم. همچنین مطابق شکل زیر پروتکل ارسال را روی استاندارد Kermit قرار می‌دهیم.



در آخر کلید Send را فشار میدهیم تا عملیات انتقال فایل شروع شود. بسته به سایز فایل و سرعت ارتباطی این عملیات ممکن است تا چند دقیقه به طول بکشد. وضعیت ارسال فایل و زمان باقی مانده در صفحه باز شده قابل مشاهده می‌باشد. پس از اتمام آپلود فایل آدرس فایل ذخیره شده در SDRAM و اندازه فایل در ترمینال نمایش داده می‌شود. این اعداد در مراحل بعدی برای ذخیره ایمیج روی فلش سریال لازم خواهند بود.

```
ECA-DEV1788> loadb
## Ready for binary (kermit) download to 0xA0000000 at 115200 bps...
## Total Size      = 0x00085fc0 = 548800 Bytes
## Start Addr      = 0xA0000000
ECA-DEV1788>
```

5.4ذخیره ایمیج لینوکس روی فلش سریال و بازخوانی آن

این مرحله صرفا برای شرایطی می‌باشد که می‌خواهیم فایل ایمیج دریافتی بر روی حافظه فلش سریال ذخیره و در دفعات بعدی، بعد از ریست میکروکنترلر دوباره خوانده و اجرا شود. بنابرین در غیر این صورت از این مرحله صرفنظر می کنیم.

برای ذخیره ایمیج دریافتی و ذخیره شده در SDRAM بر روی حافظه فلش سریال ابتدا توسط دستور sf probe فلش سریال را انتخاب و فعال می‌نماییم. سپس با توجه به اندازه فایل ایمیج که در مرحله قبلی به دست آورده ایم حافظه فلش را با دستور sf erase پاک می‌کنیم.

به دلیل اندازه مشخص بلوک های حافظه در فلش سریال، اندازه حافظه مورد نیاز برای پاکسازی در دستور erase می‌بایست کمی بزرگتر از سایز اصلی فایل و مضربی از 0x100 باشد. به عنوان مثال در حالی که اندازه فایل 85fc0 می باشد مقدار 86000 از حافظه پاک شده است. عدد 1000 نیز آدرس مبدا شروع پاکسازی می باشد که همان طور که قبلا اشاره شد به دلیل استفاده از حافظه فلش سریال برای ذخیره پیکربندی U-Boot از 4 کیلوبایت اول صرفنظر شده است.

```
ECA-DEV1788> sf probe 0
4096 KiB W25Q32 at 0:0 is now current device
ECA-DEV1788> sf erase 1000 86000
Erase: 20 00 10 00
.....
```

```
ECA-DEV1788>
```

حال با استفاده از دستور sf write ایمیج دریافتی را به حافظه فلش سریال منتقل می‌نماییم. با توجه به اندازه فایل این عملیات ممکن است مدتی طول بکشد.

```
ECA-DEV1788> sf write a0000000 1000 85fc0
ECA-DEV1788>
```

برای خواندن مجدد ایمیج ذخیره شده روی فلش سریال در آدرس 0xA0000000 از حافظه SDRAM از دستور sf read مطابق مثال زیر استفاده می‌کنیم.

```
ECA-DEV1788> sf read a0000000 1000 85fc0
ECA-DEV1788>
```

5.5 بوت لینوکس از روی حافظه

پس از آنکه فایل ایمیج سیستم عامل از هر طریقی به حافظه SDRAM برد منتقل شد (سریال ، شبکه یا خواندن از روی فلش سریال) می توانیم به سادگی توسط دستورbootm سیستم عامل را بالا بیاوریم. به صورت پیشفرض این دستور از آدرس 0xA0000000 حافظه رم به عنوان ابتدای فایل ایمیج استفاده می‌کند. ولی در صورتیکه آدرس فایل محلی غیر از این آدرس باشد می‌توان بعد دستور آدرس محل مورد نظر را نیز ارسال کرد.

در مثال زیر ایمیج نمونه sample-app.uImage غیر فشرده از آدرس 0xA0000000 حافظه رم بارگذاری و اجرا شده است :

```
ECA-DEV1788> bootm a0000000
## Booting kernel from Legacy Image at a0000000 ...
   Image Name:   Linux-2.6.33-arm1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    548736 Bytes = 535.9 kB
   Load Address: a0008000
   Entry Point:  a0008001
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Linux version 2.6.33-arm1 (raminjafarpour@ECA) (gcc version 4.4.1
(Sourcery G++
Lite 2010q1-189) ) #2 Mon Aug 10 14:08:12 IRDT 2015
CPU: ARMv7-M Processor [412fc230] revision 0 (ARM
CPU: NO data cache, NO instruction cache
Machine: NXP LPC178x/7x
Built 1 zonelists in Zone order, mobility grouping off.  Total pages:
8128
```

```
Kernel command line: lpc178x_platform=ea-lpc1788 console=ttyS0,115200
panic=10 i
p=192.168.0.100:192.168.0.1:::ea-lpc1788:eth0:off
ethaddr=C0:B1:3C:88:88:88
PID hash table entries: 128 (order: -3, 512 bytes)
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 32MB = 32MB total
Memory: 31916k/31916k available, 852k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0x00000000 - 0x00001000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0x00000000 - 0xffffffff   (4095 MB)
    lowmem  : 0xa0000000 - 0xa2000000   (  32 MB)
    modules : 0xa0000000 - 0x01000000   (1552 MB)
      .init : 0xa0008000 - 0xa0018000   (  64 kB)
      .text : 0xa0018000 - 0xa0086000   ( 440 kB)
      .data : 0xa0086000 - 0xa008df80   (  32 kB)
Hierarchical RCU implementat
NR_IRQS:41
Calibrating delay loop... 31.23 BogoMIPS (lpj=156160)
Mount-cache hash table entries: 512
Switching to clocksource lpc178x-timer1
Serial: 8250/16550 driver, 5 ports, IRQ sharing disabled
serial8250.0: ttyS0 at MMIO 0x4000c000 (irq = 5) is a 16550A
console [ttyS0] enabled
serial8250.2: ttyS1 at MMIO 0x40098000 (irq = 7) is a 16550A
Freeing init memory: 64K
Mounting /proc..
Reading /proc/meminfo:
MemTotal:          31980 kB
MemFree:           31556 kB
Buffers:               0 kB
Cached:               32 k
SwapCached:            0 kB
Active:                0 kB
Inactive:              0 kB
Active(anon):          0 kB
Inactive(anon):        0 kB
Active(file):          0 kB
Inactive(file):        0 kB
Unevictable:           0 kB
Mlocked:               0 kB
MmapCopy:             72 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:             0 kB
Mapped:                0 kB
Shmem:                 0 kB
Slab:                240 kB
SReclaimable:         24 kB
SUnreclaim:          216 kB
KernelStack:          72 kB
```

```
PageTables:             0 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:        15988 kB
Committed_AS:           0 kB
VmallocTotal:           0 kB
VmallocUsed:            0 kB
VmallocChunk:           0 kB
Done
Hello, A2F-Linux!
```

5.6 بوت خودکار

در صورتیکه نیاز باشد هر بار بعد ریست شدن میکروکنترلر ایمیج لینوکس از روی حافظه فلش سریال خوانده و اجرا شود از روش زیر میتوانیم استفاده کنیم.

برای استفاده از این روش نیاز است تنها برای بار اول فایل ایمیج مطابق بخش های 5.3 و 5.4 به حافظه فلش سریال منتقل شده و اندازه فایل نیز مشخص باشد.

به عنوان مثال فرض می‌کنیم ایمیج سیستم عامل با سایز 85fc0 از آدرس 1000 ذخیره شده است. با استفاده از قابلیت اسکریپت نویسی و دستور بوت خودکار میتوان U-Boot را طوری پیکربندی کرد که هر بار بعد ریست میکروکنترلر فایل ایمیج را به صورت خودکار از فلش سریال به حافظه SDRAM منتقل کرده و آنرا اجرا نماید.

برای اینکار توسط دستور set ابتدا یک اسکریپت به اسم sfboot با دستورات زیر ایجاد میکنیم:

```
set sfboot 'sf probe 0; sf read a0000000 1000 85fc0; bootm'
```

این اسکریپت ابتدا فلش سریال را فعال کرده و سپس مقدار 85fc0 بایت (اندازه فایل) با آدرس مبدا 1000 را از روی فلش سریال خوانده و به آدرس A0000000 حافظه SDRAM که آدرس پیش فرض بوت می باشد منتقل میکند. در نهایت نیز توسط دستور bootm ایمیج آماده شده اجرا می‌شود.

حال توسط دستور زیر متغیر بوت خودکار bootcmd را روی اسکریپت تولید شده تنظیم می نماییم تا هر بار آنرا اجرا نماید.

```
set bootcmd run sfboot
```

در نهایت نیز تغییرات اعمال شده در پیکربندی را توسط دستور save ذخیره می‌نماییم.

```
save
```

برای است عملکرد صحیح کافی است یکبار میکروکنترلر را ریست و یا دستور reset را روی ترمینال ارسال نماییم.

```
reset
```

مشاهده می‌شود که پس از نمایش صفحه اولیه، فریمور U-Boot مهلت سه ثانیه ای برای جلوگیری از بوت خودکار سیستم عامل را می‌دهد که در صورت عدم تمایل به بوت خودکار می‌بایست در این زمان یک کلید را ارسال کرد. پس از گذشت این زمان در صورتیکه هیچ کلیدی فشرده نشود ایمیج سیستم عامل به صورت خودکار خوانده و بالا می آید.

6. شروع برنامه نویسی لینوکس

در این بخش توضیح مختصری درباره نحوه کامپایل پروژه نمونه مبتنی بر لینوکس برای LPC1788 داده می شود. بدیهی است که بحث های تخصصی تر نظیر نحوه درایور نویسی و راه اندازی آن ها خارج از چارچوب این راهنما بوده و توصیه می‌شود به سایت ها و انجمن تخصصی مراجعه شود.

برای کار با بسته پشتیبانی برد و کامپایل پروژه ها نیاز به سیستم عامل لینوکس داریم که در این مثال از سیستم عامل Ubuntu نسخه 12.04 LTS استفاده شده است. استفاده از سایر توزیع های لینوکس نیز به احتمال زیاد بدون مشکل خواهد بود. دستورات ارائه شده در این بخش نیز مربوط به محیط فرمان (ترمینال) لینوکس نصب شده بر روی سیستم یا ماشین مجازی می‌باشد.

ابتدا فایل فشرده linux-ECA-1788.tar.bz2 را به سیستم عامل لینوکس منتقل کرده و در مسیر Home یا Desktop از حالت فشرده خارج میکنیم. برای خارج کردن فایل فشرده از دستور زیر میتوان استفاده کرد:

```
tar xvf linux-ECA-1788.tar.bz2
```

سپس در محیط خط فرمان با دستور cd به پوشه خارج شده از فایل فشرده تغییر مسیر می دهیم.

```
cd linux-ECA-1788
```

برای فعال کردن محیط توسعه gnu arm gcc می‌بایست مسیرهای مربوط به محیط توسعه را در PATH سیستم عامل اضافه نماییم. این کار به سادگی توسط اسکریپت فایل ACTIVATE.sh انجام می‌شود. برای اجرای این اسکریپت دستور زیر را در خط فرمان وارد می‌کنیم:

```
. ACTIVATE.sh
```

حال محیط توسعه مخصوص ARM آماده کامپایل پروژه های مبتنی بر این پلتفرم می‌باشد. پروژه نمونه که برای شروع کار با لینوکس روی LPC1788 آماده شده است در مسیر projects/hello قرار دارد. برای کامپایل این پروژه به مسیر مورد نظر تغییر مسیر می‌دهیم و توسط دستور make پروژه مورد نظر را کامپایل می‌نمائیم.

```
cd projects/hello
```

```
make
```

فرایند کامپایل پروژه ممکن است چند دقیقه به طول بکشد. در انتهای کار اندازه فایل ایمیج تولید شده نشان داده می‌شود. در صورت عدم وجود مشکل در برنامه فایل ایمیج نهایی hello.uImage در همان پوشه پروژه hello ایجاد می‌شود. برای تست برنامه نیز باید طبق آموزش های قبلی این فایل ایمیج به برد آموزشی منتقل و اجرا گردد.

برنامه به زبان C این پروژه با اسم hello.c نیز در داخل پوشه hello قرار دارد. این برنامه یک مثال ساده می‌باشد که از طریق پورت سریال یک رشته را به صورت متناوب با تاخیر 3 ثانیه ای ارسال می‌کند.