Linux DT e Device Drivers

Técnicas de Programação para Sistemas Embarcados II

Prof. Francisco Helder

Universidade Federal do Ceará

October 18, 2023

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Descrevendo Hardware não Detectáveis

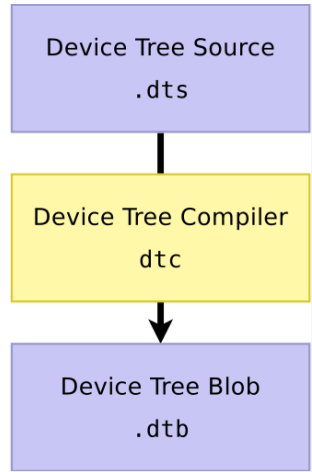
- Em sistemas embarcados, os dispositivos geralmente não são conectados por meio de um barramento que permite enumeração, PlugingPlay e fornecimento de identificadores exclusivos para dispositivos.
- Por exemplo, os dispositivos em barramentos I2C ou SPI, ou os dispositivos que fazem parte diretamente no system-on-chip.
- No entanto, ainda queremos que todos esses dispositivos façam parte do Device.
- Tais dispositivos, em vez de serem detectados dinamicamente, devem ser descritos estaticamente.

Usando o Device Tree

- Origina-se do **OpenFirmware**, definido pela Sun, usado em SPARC e PowerPC
  - É por isso que muitas funções Linux/U-Boot relacionadas ao DT têm um prefixo of_
- Agora usado pela maioria das arquiteturas de CPU embarcados que executam Linux: ARC, ARM64, RISC-V, ARM32, PowerPC, Xtensa, MIPS, etc.
- Escrever/ajustar um DT é necessário ao portar o Linux para uma nova placa ou ao conectar periféricos adicionais

Device Tree: dos fontes para blob

- Estrutura em árvore que descreve o hardware, com extensão **Device Tree Source** (.dts)
- Processado pelo **Device Tree Compiler** (dtc)
- Gera representação mais eficiente: **Device Tree Blob** (.dtb)
- .dtb → descreve com precisão o hardware de maneira independente do SO.
- .dtb ≈ algumas dezenas de kilobytes
- DTB também chamado de **Flattened Device Tree** (FDT), uma vez carregado na memória.
  - comando fdt no U-Boot
  - APIs fdt_



Ecemplo de DTC

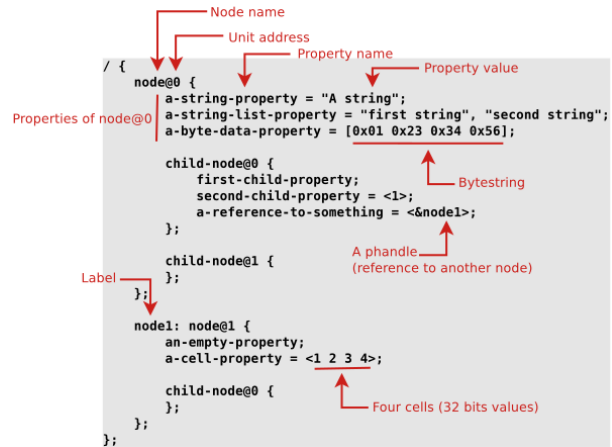
```
$ more foo.dts
/dts-v1/;
/ {
 welcome = <0xBADCAFE>;
 bootlin {
  webinar = "great";
  demo = <1>, <2>, <3>;
 };
};
```

```
$ ls -lha foo.dt*
-rw-rw-r-- 1 heldercs heldercs 16 10:01 foo.dtb
-rw-rw-r-- 1 heldercs heldercs 16 09:59 foo.dts
```

```
$ dtc -I dtb -O dts  foo.dtb
/dts-v1/;

/ {
 welcome = <0xbadcafe>;

 bootlin {
  webinar = "great";
  demo = <0x1 0x2 0x3>;
 };
};
```

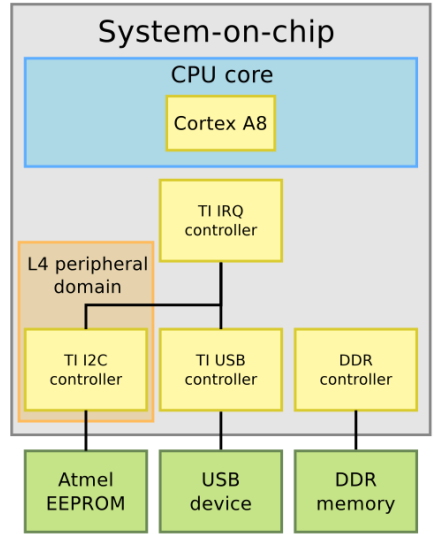
Sintaxe Básica de Device Tree

- Árvore de nós
- Nós com propriedades
- Nó ≈ um dispositivo ou bloco IP
- Propriedades ≈ características do dispositivo
- Noção de cells em valores de propriedades
- Noção de phandle para apontar para outros nós
- dtc faz apenas verificação de sintaxe, sem validação semântica



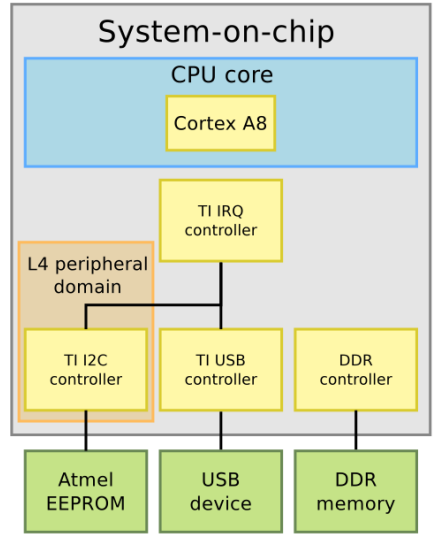
Estrutura Geral do DT: Exemplo Simplificado

```
/ {
 #address-cells = <1>;
 #size-cells = <1>;
 model = "TI AM335x BeagleBone Black";
 compatible = "ti,am335x-bone-black";
 cpus { ... };
 memory@80000000 { ... };
 chosen { ... };
 ocp {
  intc: interrupt-controller@48200000 { ... };
  usb0: usb@47401300 { ... };
  l4_per: interconnect@44c00000 {
   i2c0: i2c@40012000 { ... };
  };
 };
};
```



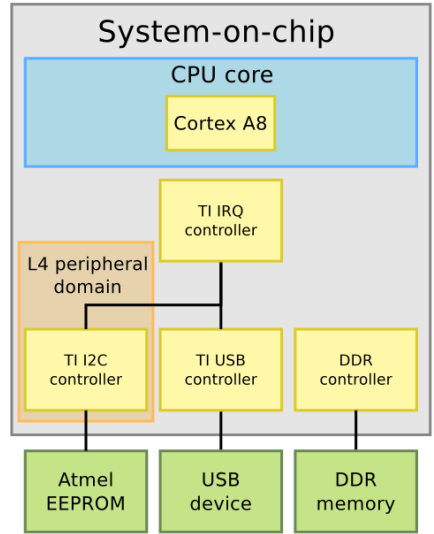
Estrutura Geral do DT: Exemplo Simplificado

```
/ {
 cpus {
  #address-cells = <1>;
  #size-cells = <0>;
  cpu0: cpu@0 {
   compatible = "arm,cortex-a8";
   enable-method = "ti,am3352";
   device_type = "cpu";
   reg = <0>;
  };
 };
 memory@0x80000000 {
  device_type = "memory";
  reg = <0x80000000 0x10000000>; /* 256 MB */
 };
 chosen {
  bootargs = "";
  stdout-path = &uart0;
 };
 ocp { ... };
};
```



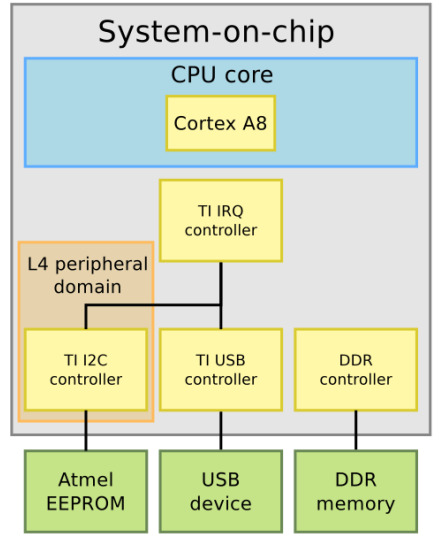
Estrutura Geral do DT: Exemplo Simplificado

```
/ {
 cpus { ... };
 memory@0x80000000 { ... };
 chosen { ... };
 ocp {
  intc: interrupt-controller@48200000 {
   compatible = "ti,am33xx-intc";
   interrupt-controller;
   #interrupt-cells = <1>;
   reg = <0x48200000 0x1000>;
  };
  usb0: usb@47401300 {
   compatible = "ti,musb-am33xx";
   reg = <0x1400 0x400>, <0x1000 0x200>;
   reg-names = "mc", "control";
   interrupts = <18>;
   dr_mode = "otg";
   dmas = <&cppi41dma 0 0 &cppi41dma 1 0 ...>;
   status = "okay";
  };
  l4_per: interconnect@44c00000 {
   i2c0: i2c@40012000 { ... };
  };
 };
};
```



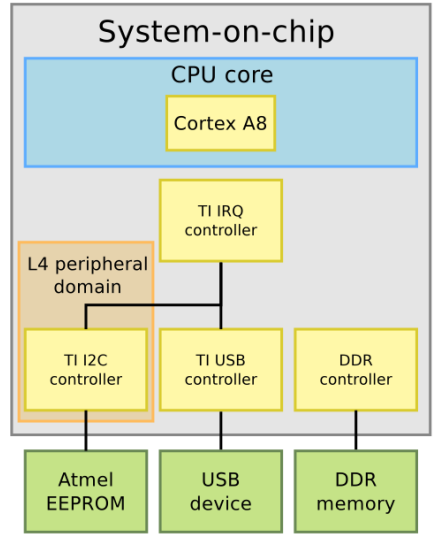
Estrutura Geral do DT: Exemplo Simplificado

```
/ {
 cpus { ... };
 memory@0x80000000 { ... };
 chosen { ... };
 ocp {
  compatible = "simple-pm-bus";
  clocks = <&l3_clkctrl AM3_L3_MAIN_CLKCTRL 0>;
  clock-names = "fck";
  #address-cells = <1>;
  #size-cells = <1>;
  intc: interrupt-controller@48200000 { ... };
  usb0: usb@47401300 { ... };
  l4_per: interconnect@44c00000 {
   compatible="ti,am33xx-wkup","simple-pm-bus";
   reg = <0x44c00000 0x800>, <0x44c00800 0x800>,
   <0x44c01000 0x400>, <0x44c01400 0x400>;
   reg-names = "ap", "la", "ia0", "ia1";
   #address-cells = <1>;
   #size-cells = <1>;
   i2c0: i2c@40012000 { ... };
  };
 };
};
```



Estrutura Geral do DT: Exemplo Simplificado

```
/ {
 cpus { ... };
 memory@0x80000000 { ... };
 chosen { ... };
 ocp {
  intc: interrupt-controller@48200000 { ... };
  usb0: usb@47401300 { ... };
  l4_per: interconnect@44c00000 {
   i2c0: i2c@40012000 {
    compatible = "ti,omap4-i2c";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x0 0x1000>;
    interrupts = <70>;
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins>;
    clock-frequency = <400000>;
    baseboard_eeprom: eeprom@50 {
     compatible = "atmel,24c256";
     reg = <0x50>;
    };
   };
  };
 };
};
```



Hierarquia Device Tree

- Os arquivos Device Tree não são monolíticos, eles podem ser divididos em vários arquivos, inclusive entre si.
- Os arquivos .dtsi são arquivos de includes, enquanto os arquivos .dts são Device Trees finais
  - Somente arquivos .dts são aceitos como entrada para **dtc**
- Normalmente, .dtsi conterá
  - definições de informações em nível de SoC
  - definições comuns a vários conselhos
- O arquivo .dts contém as informações no nível da placa
- A inclusão funciona sobrepondo a árvore do arquivo incluído sobre a árvore do arquivo, de acordo com a ordem das diretivas #include.
- Permite que um arquivo incluído substitua valores especificados no arquivo.
- Usa a diretiva **#include** do pré-processador C

Exemplo de Hierarquia de Device Tree

Definition of the AM33xx SoC family

```
&l4_wkup {
    target-module@b000 {
        i2c0: i2c@0 {
            compatible = "ti,omap4-i2c";
            reg = <0x0 0x1000>;
            interrupts = <70>;
            status = "disabled";
        };
    };
};
```

am33xx-l4.dtsi

+

Definition of the Bone Black board

```
#include "am33xx-l4.dtsi"

&i2c0 {
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins>;
    status = "okay";

    baseboard_eeprom: eeprom@50 {
        compatible = "atmel,24c256";
        reg = <0x50>;
    };
};
```

am335x-boneblack.dts

Note 1

The actual Device Trees for this platform are more complicated. This example is highly simplified.

=

Compiled DTB

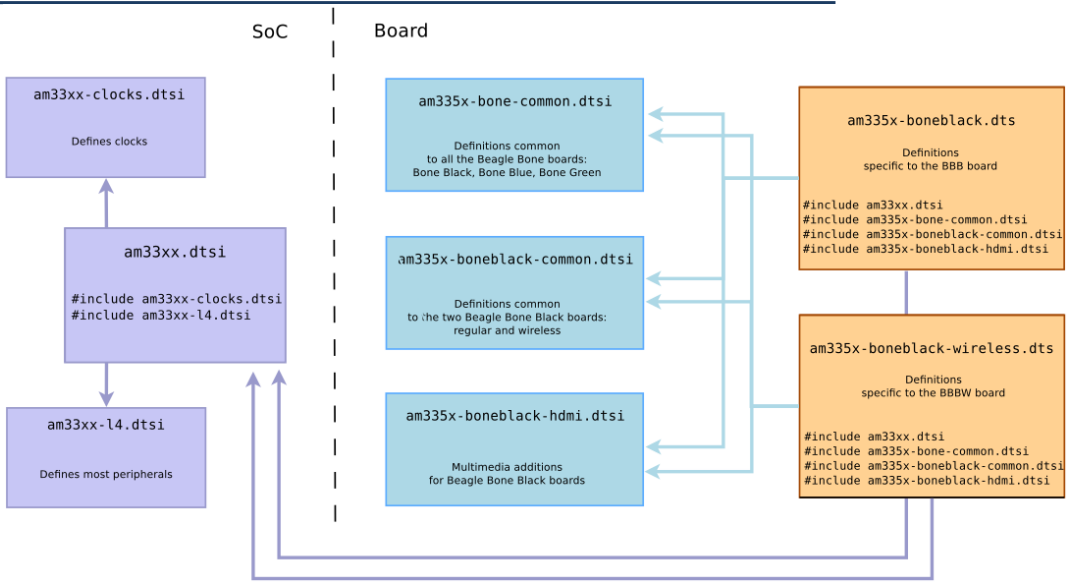
```
&l4_wkup {
    target-module@b000 {
        i2c0: i2c@0 {
            compatible = "ti,omap4-i2c";
            reg = <0x0 0x1000>;
            interrupts = <70>;
            pinctrl-names = "default";
            pinctrl-0 = <&i2c0_pins>;
            status = "okay";

            baseboard_eeprom: eeprom@50 {
                compatible = "atmel,24c256";
                reg = <0x50>;
            };
        };
    };
};
```

am335x-boneblack.dtb

Note 2

The real DTB is in binary format. Here we show the text equivalent of the DTB contents.

Hierarquia DT na Bone Black



Princípios de Projeto DT

- Descrever o hardware (como é o hardware), não a configuração (como escolho usar o hardware)
- Independente de sistema operacional
  - Para um determinado hardware, DT deve ser a mesma para U-Boot, FreeBSD ou Linux
  - Não deve haver necessidade de alterar a árvore de dispositivos ao atualizar o sistema operacional
- Descreva a integração dos componentes de hardware, não a parte interna dos componentes de hardware
  - Os detalhes de como um dispositivo/bloco IP específico está funcionando são tratados pelo código nos drivers de dispositivo
  - DT descreve como o dispositivo/bloco IP está conectado/integrado com o resto do sistema: linhas IRQ, canais DMA, clocks, linhas de reset, etc.
- Como todos os princípios de design, estes princípios são por vezes violados.

Device Drivers

Técnicas de Programação para Sistemas Embarcados II

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

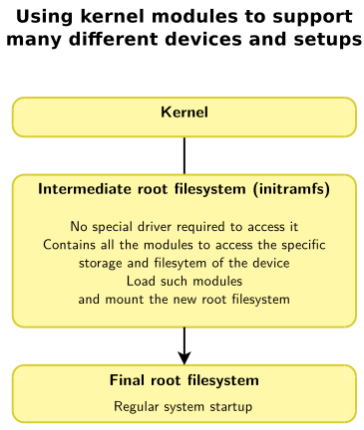
Prof. Francisco Helder

Universidade Federal do Ceará

October 18, 2023

Vantagens dos Módulos

- Os módulos facilitam o desenvolvimento de drivers sem reinicialização: carregar, testar, descarregar, reconstruir, carregar...
- Útil para manter o tamanho da imagem do kernel no mínimo.
- Também é útil para reduzir o tempo de inicialização: você não perde tempo inicializando dispositivos e recursos do kernel que só serão necessários mais tarde.
- Para aumentar a segurança, possibilidade de permitir apenas módulos assinados ou desabilitar totalmente o suporte a módulos.
- Atenão: uma vez carregado, tenha total controle e privilégios no sistema. É por isso que apenas o usuário root pode mexer nos módulos.



Dependências dos Módulos

- Alguns módulos dependem de outros módulos, que precisam ser carregados primeiro.

Exemplo

o módulo usb_storage depende do módulo usbcore.

- As dependências entre os módulos estão descritas no arquivo.

```
$ /lib/modules/<kernel-version>/modules.dep
```

- Este arquivo é gerado automaticamente quando você instala os módulos, através da ferramenta depmod.

Carregando um Módulo

Carrega apenas o módulo passado. É necessário passar o caminho completo do módulo.

```
$ insmod /path/to/module.ko
```

Carrega o módulo e todas as suas dependências. Deve-se passar apenas o nome do módulo, que deve estar instalado em /lib/modules/.

```
$ modprobe <module_name>
```

Descarregando um Módulo

Descarrega apenas o módulo passado. Deve-se passar apenas o nome do módulo. Possível apenas se o módulo não estiver mais em uso.

```
$ rmmod <module_name>
```

Descarrega o módulo e todas as suas dependências (que não estão sendo usadas). Deve-se passar apenas o nome do módulo.

```
$ modprobe -r <module_name>
```

Listando Informações dos Módulos

Lê informações de um módulo, como sua descrição, parâmetros, licença e dependências. Deve-se passar apenas o nome do módulo, que deve estar instalado em /lib/modules/.

```
$ modinfo <module_name>
```

Lista todos os módulos carregados.

```
$ lsmod
```

Listando Informações dos Módulos

- Passando um parâmetro via linha de comando:

```
$ modprobe <module> param=value
```

- Passando um parâmetro via arquivo de (/etc/modprobe.conf ou /etc/modprobe.d/):

```
$ options <module> param=value
```

- Para passar um parâmetro via linha de comandos do kernel:

```
$ <module>.param=value
```

Passando Parâmetros para Módulos

- Encontre os parâmetros disponíveis:

```
$ modinfo usb-storage
```

- Via insmod:

```
$ sudo insmod ./usb-storage.ko delay_use=0
```

- Via modprobe:

```
Set parameters in /etc/modprobe.conf or in any file in /etc/modprobe.d/: options usb-storage delay_use=0
```

- Via linha de comando no kernel, Quando o módulo e construído estaticamente no kernel:

```
usb-storage.delay_use=0
\begin{itemize}
 \item usb-storage is the module name
 \item delay_use is the module parameter name. It specifies a delay before accessing a
 USB storage device (useful for rotating devices).
 \item 0 is the module parameter value`
\end{itemize}
```

conferindo os Valores dos Parâmetros no Módulo

- Como encontrar/editar os valores atuais dos parâmetros de um módulo carregado?
- Verifique /sys/module/<name>/parameters.
- Existe um arquivo por parâmetro, contendo o valor do parâmetro.
- Também é possível alterar os valores dos parâmetros se estes arquivos tiverem permissões de escrita (depende do código do módulo).

Examplo:

```
$ echo 0 > /sys/module/usb_storage/parameters/delay_use
```

Entendendo Erros de carregamento do módulo

- Quando o carregamento de um módulo falha, o insmod geralmente não fornece detalhes suficientes!
- Os detalhes geralmente estão disponíveis no log do kernel.

Exemplo:

```
$ sudo insmod ./intr_monitor.ko
insmod: error inserting './intr_monitor.ko': -1 Device or resource busy
$ dmesg
[17549774.552000] Failed to register handler for irq channel 2
```

Exemplo de Código para Módulo

```c
// SPDX-License-Identifier: GPL-2.0
/* hello.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static int __init hello_init(void){
  pr_alert("Good morrow to this fair assembly.\n");
  return 0;
}

static void __exit hello_exit(void){
  pr_alert("Alas, poor world, what treasure hast thou lost!\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module");
MODULE_AUTHOR("William Shakespeare");
```

Exemplo de Código para Módulo

- Código definido com __init:
  - Removido após a inicialização (kernel ou módulo estático).
  - Veja como a memória init é recuperada quando o kernel termina a inicialização:

```
[2.689854] VFS: Mounted root (nfs filesystem) on device 0:15.
[2.698796] devtmpfs: mounted
[2.704277] Freeing unused kernel memory: 1024K
[2.710136] Run /sbin/init as init process
```

- Código definido com __exit:
  - Descartado quando o módulo é compilado estaticamente no kernel ou quando o suporte ao descarregamento do módulo não está habilitado.

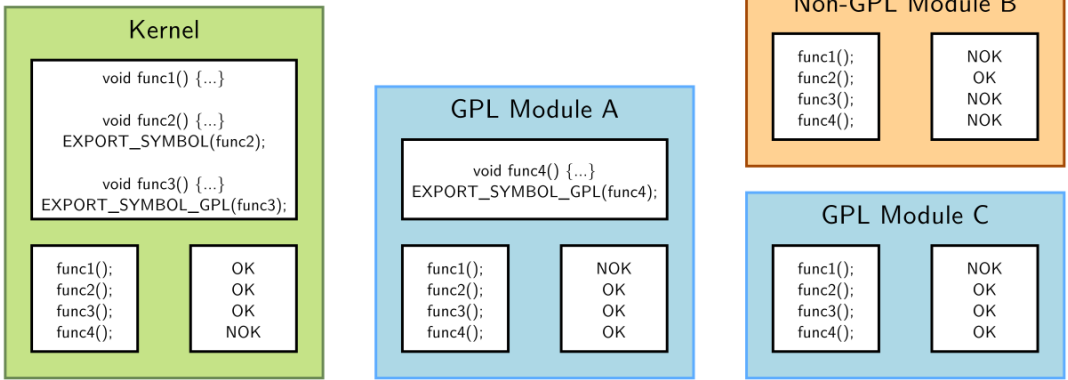
Explicação do Módulo Exemplo

- Biblioteca específicos para o kernel Linux: linux/xxx.h
  - Sem acesso à biblioteca C normal, estamos programando o kernel
- Função de inicialização
  - Chamado quando o módulo é carregado, retorna um código de erro (0 em caso de sucesso, valor negativo em caso de falha)
  - Declarado pela macro module_init(): o nome da função não importa, mesmo que <modulename>_init() seja uma convenção.
- Função de limpeza
  - Chamado quando o módulo é descarregado
  - Declarado pela macro module_exit().
- Informações de metadados declaradas usando MODULE_LICENSE(), MODULE_DESCRIPTION() e MODULE_AUTHOR()

Símbolos Exportado para o Módulo

- A partir de um módulo do kernel, apenas um número limitado de funções do kernel pode ser chamado
- Funções e variáveis devem ser explicitamente exportadas pelo kernel para serem visíveis para um módulo do kernel
- Duas macros são usadas no kernel para exportar funções e variáveis:
  - EXPORT_SYMBOL(symbolname), que exporta uma função ou variável para todos os módulos
  - EXPORT_SYMBOL_GPL(symbolname), que exporta uma função ou variável apenas para módulos GPL
  - Linux 5.3: contém o mesmo número de símbolos com EXPORT_SYMBOL() e símbolos com EXPORT_SYMBOL_GPL()
- Um driver normal não deve precisar de nenhuma função não exportada.

Símbolos Exportado para o Módulo



Compilando um Módulo

- Existem duas opções para compilar um módulo do kernel:
  - Fora da árvore do kernel: fácil de gerenciar, mas não permite compilar um módulo estaticamente (built-in).
  - Dentro da árvore do kernel: permite compilar um módulo estaticamente ou dinamicamente.

Prática de lab - Escrevendo Módulos e DT

Hora de começar o laboratório prático 06!

1. Crie, compile e carregue seu próprio modulo
2. Navegue através do DT da BBB e realize mudanças
3. Use GPIO LEDs