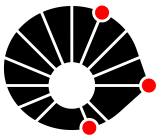

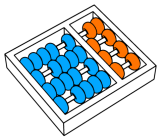

Abordagens para Superar Limitações de Hardware em TV Box RK3229 para uso em IoT

Igor Gabriel C. de C. Borges *Juliana Freitag Borin*

Relatório Técnico - IC-PFG-23-48
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Abordagens para Superar Limitações de Hardware em TV Box RK3229 para uso em IoT

Igor Gabriel Cavalcante de Carvalho Borges Juliana Freitag Borin *

Resumo

Este trabalho explora a capacidade de transformação de uma TV Box baseada no SoC Rockchip RK3229 em um dispositivo de Internet das Coisas (IoT), abordando duas estratégias para contornar o desafio da ausência de pinos expostos por padrão. A primeira envolve a reutilização de conexões para componentes existentes na placa, como LEDs e alguns conectores. Esta abordagem inclui a dessoldagem de componentes e a modificação ou desativação de drivers, além do uso de Device Tree Overlays para converter os pinos disponíveis em GPIOs genéricos. A segunda estratégia emprega a UART de depuração da placa para conectá-la a um microcontrolador e utilizar seus pinos. Para isso foi desenvolvido um driver personalizado que expõe um chip GPIO falso na TV Box, realizando a comunicação entre dispositivos de forma transparente ao usuário. Ambas as abordagens são detalhadas e comparadas, e seus resultados discutidos.

1 Introdução

A era da Internet das Coisas (IoT) trouxe uma revolução na interação humana com dispositivos tecnológicos, impulsionando inovações em várias áreas, desde eletrodomésticos inteligentes até sistemas avançados de segurança. Neste contexto, a reutilização e a transformação de dispositivos eletrônicos comuns em ferramentas conectadas à IoT emerge como uma frente promissora e sustentável de pesquisa e desenvolvimento.

Este trabalho é resultado de uma iniciativa entre a Universidade Estadual de Campinas (Unicamp) e a Receita Federal. Em junho de 2023, a Unicamp recebeu um número de aparelhos de TV Box, apreendidos por captação irregular de sinais de televisão, para estudo e pesquisa [5].

Como um dos vários trabalhos iniciados a partir desta iniciativa, este estudo foca na questão do hardware e a possibilidade de aplicações em IoT similares às de um Raspberry Pi, um dispositivo de baixo custo. Outras propostas incluem adaptações de software, como o uso dos dispositivos como *hosts* para máquinas virtuais com maior poder de processamento para uso educacional.

O foco do estudo se dá por conta do dispositivo ser inicialmente limitado pela ausência de pinos expostos. Duas abordagens principais são exploradas: a reutilização de conexões para componentes existentes na placa, como LEDs, e o uso da UART (Universal Asynchronous Receiver-Transmitter) para conexão com um microcontrolador.

O documento está estruturado da seguinte maneira: a seção 2 apresenta o dispositivo utilizado no estudo, a seção 3 descreve o sistema operacional utilizado, e a seção 4 descreve os conceitos e ferramentas de Linux embarcado utilizados. As seções 5 e 6 descrevem as abordagens utilizadas, e a seção 7 faz uma análise comparativa entre elas. Por fim, a seção 8 apresenta as conclusões e possibilidades de elaboração futura.

*Instituto de Computação, Universidade Estadual de Campinas, SP, Brasil.

2 Visão Geral do Dispositivo

O dispositivo utilizado neste estudo é uma TV Box baseada no chip RK3229, um SoC (System-on-Chip) de baixo custo desenvolvido pela Rockchip. É vendido com a marca TX 2, e placas similares são comercializadas sob diferentes nomes. A Figura 1 mostra o dispositivo e sua caixa original.

Figura 1: Dispositivo e sua caixa original.

A placa possui 2GB de RAM e 16GB de armazenamento interno. Dentre seus componentes estão portas USB, Ethernet, HDMI, áudio óptico (SPDIF), e entrada para cartão SD. A placa também possui um receptor de infravermelho e um LED principal indicando o status do dispositivo. A Figura 2 mostra a placa e a localização dos componentes.

Uma descrição completa do SoC pode ser encontrada em seu *datasheet*, no site da Rockchip [6]. A placa em si não possui documentação oficial disponível.

3 Sistema Operacional

O sistema operacional instalado por padrão é uma versão modificada do Android 7. O firmware do dispositivo é configurado de forma a priorizar a inicialização a partir de um cartão SD, permitindo a instalação de outros sistemas operacionais, utilizando o sistema na memória interna como *fallback*.

Utilizando instruções disponíveis no fórum oficial do Armbian [2], foi possível instalar sua distribuição Linux, baseada em Debian. Os passos para a instalação foram os seguintes:

- Baixar e gravar a ferramenta Multitool [1] no cartão SD;
- Baixar e salvar uma imagem do Armbian para TV Boxes RK322X na pasta "images" do cartão SD. A imagem utilizada foi obtida nas builds semanais no repositório *community* [2].
- Inserir o cartão SD no dispositivo e ligá-lo;
- Seguir as instruções na tela para instalar o Armbian na memória interna.
- Remover o cartão SD e reiniciar o dispositivo.

Instruções mais detalhadas podem ser encontradas na thread original do fórum.

[1] `https://users.armbian.com/jock/rk322x/multitool/multitool.img.xz`
[2] `https://github.com/armbian/community/releases`

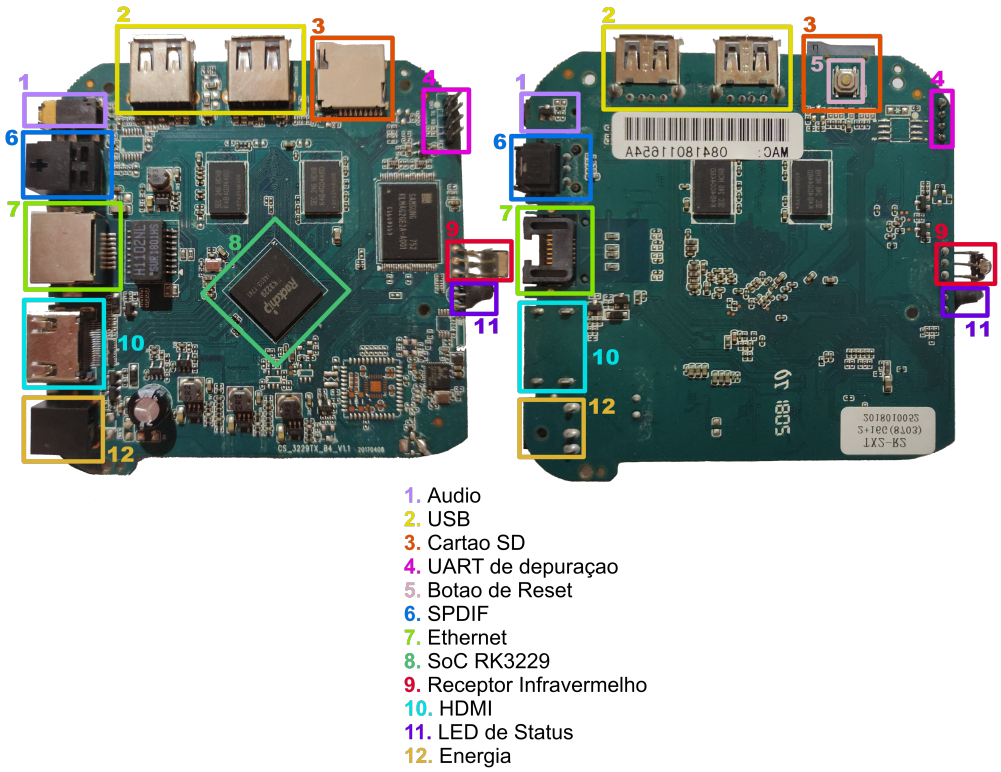

Figura 2: Frente e verso da placa.

4 Conceitos e Ferramentas de Linux Aplicados

A implementação das abordagens descritas neste trabalho envolveu o uso de certos conceitos e ferramentas próprios de sistemas Linux embarcados. Esta seção descreve os principais conceitos utilizados.

4.1 Device Tree e Device Tree Overlays

A Device Tree é uma estrutura de dados utilizada para descrever a configuração de hardware de um sistema embarcado, permitindo que o kernel seja compilado de forma genérica e que a configuração seja definida em tempo de boot.

Em sua forma pré-compilação, a Device Tree é um arquivo de texto (extensão .dts) que descreve a configuração de hardware do sistema, com uma sintaxe baseada em nós e propriedades. Cada nó representa um componente de hardware, e cada propriedade representa um atributo do componente. O arquivo de Device Tree é compilado em um arquivo binário (extensão .dtb), que é carregado pelo bootloader e utilizado pelo kernel para configurar o hardware.

A Device Tree do dispositivo pode ser encontrada no diretório `/boot/dtb/` do sistema, e decompilada utilizando o comando:

```
dtc -I dtb -O dts -o <nome do arquivo>.dts <nome da saida>.dtb
```

A Figura 3 mostra o começo do arquivo de Device Tree de um dispositivo similar, como um exemplo da sintaxe utilizada.

```
// SPDX-License-Identifier: (GPL-2.0+ OR MIT)

/dts-v1/;

#include <dt-bindings/input/input.h>
#include "rk3229.dtsi"

/ {
  model = "Rockchip RK3229 Evaluation board";
  compatible = "rockchip,rk3229-evb", "rockchip,rk3229";

  aliases {
    mmc0 = &emmc;
  };

  memory@60000000 {
    device_type = "memory";
    reg = <0x60000000 0x40000000>;
  };

  dc_12v: dc-12v-regulator {
    compatible = "regulator-fixed";
    regulator-name = "dc_12v";
    regulator-always-on;
    regulator-boot-on;
    regulator-min-microvolt = <12000000>;
    regulator-max-microvolt = <12000000>;
  };
  ...
```

Figura 3: Exemplo de arquivo de Device Tree.

Device Tree Overlays são extensões da Device Tree que permitem a adição ou modificação de nós e propriedades em tempo de execução, sem a necessidade de recompilar o arquivo de Device Tree. A Figura 4 mostra um exemplo de overlay que desabilita um nó na Device Tree da forma mais simples, utilizado neste trabalho para desabilitar o LED principal presente no nó `/gpio-leds`.

No Armbian, as overlays são armazenadas no diretório `/boot/dtb/overlay/`, e são carregadas se mencionadas no arquivo `/boot/armbianEnv.txt`. Para conveniência, o Armbian também possui um comando para compilar e carregar overlays:

```
armbian-add-overlay <nome do arquivo>.dts
```

Para ambas as abordagens, foi necessário o uso de overlays para modificar a configuração do hardware da placa. As formas de utilização são descritas nas seções correspondentes.

```
/dts-v1/;
/plugin/;

/ {
  compatible = "rockchip,rk3229";

  fragment@0 {
    target-path = "/gpio-leds";
    __overlay__ {
      status = "disabled"; // desabilita o no no caminho especificado
    };
  };
};
```

Figura 4: Exemplo de overlay que desabilita um nó na Device Tree.

4.2 UART

A placa possui uma UART de depuração, presente no sistema como `/dev/ttyS2`. Ela é utilizada para comunicação com o bootloader e depuração do kernel. Após o processo de boot, já no sistema operacional, o serviço `serial-getty` é iniciado, criando uma instância do terminal que pode ser acessada por um dispositivo conectado à UART, como um Raspberry Pi ou um computador comum com um adaptador USB-Serial. Para liberar a UART para outros usos, o serviço `serial-getty` deve ser desabilitado. Isso pode ser feito por meio do comando:

```
sudo systemctl mask serial-getty@ttyS2.service
```

Nos testes realizados, o uso de mascaramento em vez de desabilitação direta foi necessário para garantir que o serviço não fosse reiniciado durante o boot.

4.3 Blacklisting de Drivers

Para algumas modificações, foi necessário desabilitar drivers específicos que se apropriam dos pinos desejados. Para evitar que um driver seja carregado, é possível adicioná-lo à lista de drivers a serem ignorados pelo sistema, chamada de *blacklist*. No Armbian, como em outras distribuições baseadas em Debian, qualquer arquivo com a extensão `.conf` no diretório `/etc/modprobe.d/` é considerado parte da *blacklist*. O conteúdo do arquivo deve ser uma lista de drivers a serem ignorados, um por linha, no formato:

```
blacklist <nome do driver>
```

Os drivers *blacklisted* neste trabalho são descritos nas seções correspondentes.

4.4 libgpiod

A biblioteca libgpiod permite manipulação de pinos GPIO no espaço do usuário. Ela é utilizada para a leitura e escrita de pinos de forma simples e direta em programas C (ou outras linguagens como Python, por meio de *bindings*). A biblioteca também possui ferramentas de linha de comando para obtenção de informações, como `gpioinfo` e `gpioget`, e manipulação de pinos, como `gpioset` [4]. Os comandos utilizados neste trabalho são:

- `gpioinfo` (`gpioinfo`): lista informações sobre os pinos disponíveis;
- `gpioget` (`gpioget <chip> <pino>`): lê o valor de um pino;
- `gpioset` (`gpioset <chip> <pino>=<valor>`): escreve um valor em um pino.

4.5 Serdev e Platform Device

Serdev, ou Serial Device Bus [3], é um framework do kernel que permite a criação de drivers para dispositivos seriais, sucedendo o framework `tty`. Na implementação deste trabalho, com uma overlay de modificação, um driver customizado é selecionado como o driver da UART de depuração (definida como `<uart2>`). A parte do driver responsável pela comunicação com o microcontrolador cria um dispositivo serdev para a UART, que pode então ser controlado no espaço do kernel, pelo driver.

A GPIO falsa é registrada no sistema pelo driver como um Platform Device. Um Platform Device é considerado como um dispositivo autônomo *non-discoverable*, ou seja, que não consegue ser descoberto automaticamente pelo kernel por ser conectado através de um barramento sem suporte a hotplug, como UART ou I2C, ao contrário de outros como USB ou PCI, mais comuns em computadores pessoais [7]. O driver é responsável por criar o dispositivo e registrá-lo no sistema, e neste caso, por implementar as funções de leitura e escrita de pinos.

5 Primeira Abordagem: Reutilização de Componentes

O desenvolvimento dessa abordagem se deu com o objetivo de maximizar o uso dos recursos já existentes na placa, liberando e reutilizando pinos para novas funcionalidades. O método foca em alterações que requerem apenas modificações de hardware simples, que não necessitam de componentes adicionais, e pequenos ajustes de software.

5.1 Identificação dos Componentes Disponíveis

A investigação inicial dos pinos disponíveis foi realizada utilizando a ferramenta `gpioinfo`, que revelou a presença de vários pinos controlados por drivers específicos ou configurados pela Device Tree. Esta análise permitiu a identificação de componentes que poderiam ser reutilizados para fins IoT. A Figura 5 mostra a saída do comando `gpioinfo`, listando os componentes identificados e suas respectivas conexões.

5.2 Modificações de Hardware

Com base na análise inicial, foram realizadas as seguintes modificações de hardware:

5.2.1 LED Principal

O LED é do tipo duplo e ânodo comum. Foi cortado com uma tesoura, deixando os pinos suficientemente longos para permitir conexões sem necessidade de solda. Foi criada uma overlay para desabilitar o nó que liga o pino ao driver `gpio-leds`. Fisicamente, o LED possuia três pinos (vermelho, azul e do meio), mas o vermelho não estava conectado a nada.

```
gpiochip0 - 32 lines:
  ...
gpiochip1 - 32 lines:
  ...
  line  3: unnamed       "spk-en" output active-high [used]
  ...
  line 11: unnamed "ir-receiver"  input  active-low [used]
  ...
  line 17: unnamed           "cd"  input  active-low [used]
  ...
gpiochip2 - 32 lines:
  ...
  line 26: unnamed        "reset" output  active-low [used]
  ...
gpiochip3 - 32 lines:
  ...
  line 20: unnamed "vcc-host-regulator" output  active-high [used]
  line 21: unnamed     "working"        output  active-high [used]
  line 22: unnamed "vcc-otg-regulator"  output  active-high [used]
  ...
```

Figura 5: Saída truncada do comando `gpioinfo` antes das modificações.

5.2.2 Receptor de Infravermelho

Assim como o LED, o receptor de infravermelho foi cortado com uma tesoura. Os drivers `gpio_ir_recv` e `ir_nec_decoder` foram blacklisted.

5.2.3 Conector de Áudio Óptico (SPDIF)

A tampa foi removida, expondo o LED e seus três pinos (VCC, VIN e GND). Os drivers `snd_soc_spdif_tx` e `snd_soc_rockchip_spdif` foram blacklisted. Foram criadas overlays de desativação para os nós `/spdif@100d0000`, `/spdif-out` e `/spdif-sound`.

Para o nó `/pinctrl/spdif/spdif-tx`, foi criada uma overlay de modificação para alterar o modo do pino de um modo específico não determinado (valor 0x02) para GPIO (valor 0x00). A linha modificada foi a seguinte:

```
rockchip,pins = <0x03 0x1f 0x00 0x5b>;
```

5.2.4 Pinos de Depuração

A placa possui um conjunto de pinos de depuração UART funcionais mas não soldados. Ao conjunto foi soldado um conector de 4 pinos, permitindo o uso de uma shell por UART. Posteriormente, o serviço `serial-getty` foi desabilitado no UART (`ttyS2`) para liberá-lo para outros usos, como a interface direta com um microcontrolador explorada na segunda abordagem.

5.3 Resultado

O resultado dessas operações é a liberação dos pinos para uso como GPIOs genéricos. Antes listados como `used`, os pinos agora aparecem como `unused` e sem nome, mas se mantém funcionais e nos mesmos números. A Figura 6 mostra a saída do comando `gpioinfo` após as modificações.

```
gpiochip0 - 32 lines:
  ...
gpiochip1 - 32 lines:
  ...
  line 11: unnamed unused  input active-high
  ...
gpiochip2 - 32 lines:
  ...
gpiochip3 - 32 lines:
  ...
  line 21: unnamed unused output active-high
  ...
  line 31: unnamed unused output active-high
```

Figura 6: Saída truncada do comando `gpioinfo` após as modificações.

6 Segunda Abordagem: Conexão com Microcontrolador

A segunda abordagem envolve a conexão da placa a um microcontrolador, utilizando a UART de depuração. O microcontrolador é responsável por receber os sinais dos pinos e transmiti-los para a placa via UART, assim como receber sinais da placa e transmiti-los para os pinos. Para isso, foi desenvolvido um driver personalizado e uma overlay para a Device Tree, criando um chip GPIO falso que transmite alterações de pinos através da UART para o microcontrolador.

O objetivo dessa abordagem é superar as limitações de hardware da placa, mantendo a vantagem do resto do hardware e software e expandindo suas funcionalidades para uso em aplicações mais complexas de IoT.

A Figura 8 mostra a organização dos componentes na segunda abordagem.

6.1 Microcontrolador

A UART de depuração da placa, soldada na primeira abordagem, é conectada a um microcontrolador, que executa um programa que se comunica com o driver customizado por um protocolo simples, transmitindo informações sobre alterações nos pinos e recebendo comandos para alterar o estado dos pinos.

Para usos em aplicações reais, é vantajoso o uso de um microcontrolador de baixo custo, como o Arduino Uno, que possui uma UART, pinos digitais e desempenho aceitável, mas não possui módulos de comunicação mais complexos, como WiFi ou Bluetooth, dos quais a placa dispõe, além de um sistema operacional completo. Porém, para fins de prova de conceito, evitando o problema de compatibilidade de voltagem entre o Arduino Uno e a placa, foi utilizado um ESP8266. Modificações no código do driver customizado podem ser necessárias para o uso de outros microcontroladores.

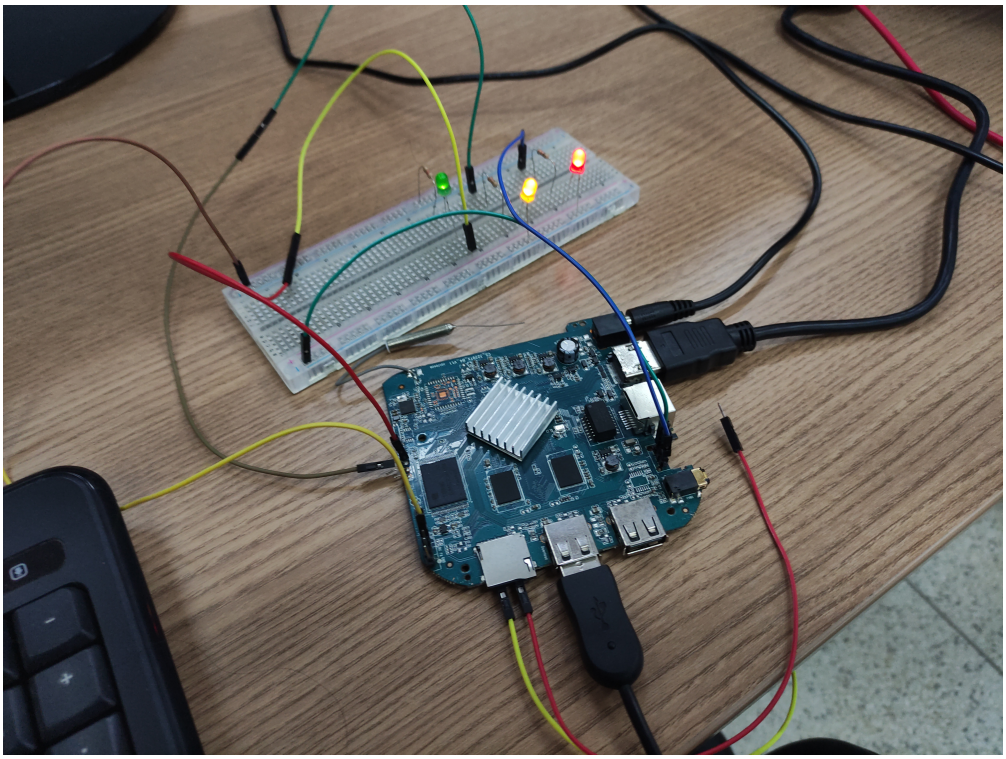

Figura 7: Controlando LEDs com os pinos obtidos pela primeira abordagem.

6.1.1 Protocolo

O protocolo utilizado é baseado em comandos e respostas de quatro bytes, cuja estrutura é mostrada na Figura 9, onde:

- **command** é o bit do commando (0 para leitura, 1 para escrita);
- **pin type** é o bit de tipo do pino (0 para digital, 1 para analógico). Sempre 0 para protocolo atual;
- **pin number** são 6 bits dedicados ao número do pino (0 a 63). Não foi pensado com nenhum microcontrolador específico em mente.
- **value** é o bit de valor do pino (0 ou 1). Sempre 0 para leitura;
- **status code** é o byte de status do comando (status implementados: 0x00 - OK, 0x01 - pino inválido, 0x02 - comando inválido);
- **unused (future work)** é um byte reservado para possíveis futuras implementações.

Para comandos enviados da placa para o microcontrolador, o esquema utilizado é o acima, com status code e unused sempre 0, e value sempre 0 para leitura. O microcontrolador somente responde, enviando um echo do comando recebido, com o status code correspondente e o valor do pino, se for uma leitura.

Nenhuma forma de checksum ou verificação de integridade foi implementada, por conta do escopo do trabalho e da simplicidade do protocolo.

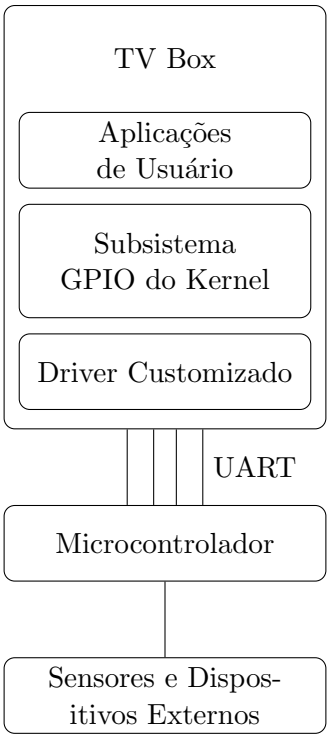

Figura 8: Organização dos componentes na segunda abordagem.

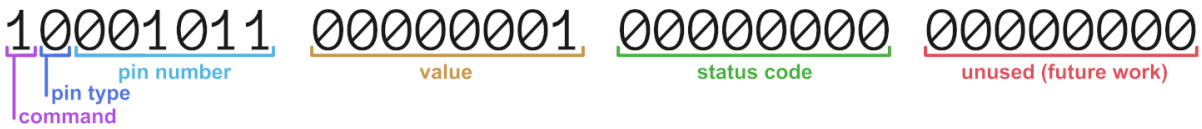

Figura 9: Estrutura de um comando.

6.2 Driver

O driver desenvolvido é composto de duas partes: um driver de dispositivo serdev e um platform driver de chip GPIO falso. A parte serdev do driver é responsável por controlar a comunicação com o microcontrolador, e a parte GPIO é responsável por criar um platform device para o chip GPIO falso, implementando as funções de leitura e escrita de pinos de forma a chamarem as funções de comunicação com o microcontrolador e atualizarem o estado dos pinos de acordo com as respostas recebidas.

Para repassar o controle da UART para o driver, foi criada e aplicada a seguinte overlay:

```
/dts-v1/;
/plugin/;

fragment@0 {
  target = <&uart2>;
  status = "okay";
  __overlay__ {
```

```
        tvbox-gpio-serdev {
          compatible = "tvbox,tvbox-gpio-serdev";
          status = "okay";
        };
      };
    };
```

O driver serdev é registrado no sistema como `tvbox-gpio-serdev`, fazendo com que a string `compatible` adicionada ao nó `uart2` seja reconhecida pelo kernel como o driver a ser utilizado para a UART. Ele implementa as funções `serial_comm_write_data` e `serial_comm_receive_buf`. A primeira é chamada pelo platform driver para enviar comandos para o microcontrolador, e a segunda é chamada automaticamente pelo kernel quando dados são recebidos na UART, salvando os dados recebidos em um buffer e setando uma flag para indicar que dados foram recebidos.

O platform driver é registrado no sistema como `tvbox-gpio-chip`. Ele implementa as funções de seu chip GPIO falso `virt_gpio_get_value`, `virt_gpio_set_value`, `virt_gpio_direction_input` e `virt_gpio_direction_output`. As funções de leitura e escrita de pinos chamam a função de escrita do driver serdev para enviar comandos para o microcontrolador, e entram em um spinlock até que a flag de recebimento de dados seja setada, indicando que a resposta foi recebida, e então retornam o valor recebido. As funções de direção atualmente apenas atualizam o estado do pino no chip GPIO falso, mas podem ser modificadas para enviar comandos para o microcontrolador, caso seja necessário.

7 Análise Comparativa

O trabalho foi desenvolvido com foco na prova de conceito. Por isso, ambas as abordagens foram testadas em termos de funcionalidade básica, sem a preocupação com a eficiência ou a robustez do código, ou com um exemplo de uso específico.

A primeira abordagem é mais simples e direta, e não requer componentes adicionais. A UART de depuração, se necessária, deve ser soldada, mas o resto dos componentes pode ser parcialmente cortado, mantendo pinos suficientemente longos para conexões. A abordagem também não requer adições ou modificações drásticas de software, apenas blacklisting de drivers e pequenas overlays para desabilitar nós na Device Tree. Porém, a abordagem é limitada pela pequena quantidade de pinos disponíveis, e pela necessidade de modificações de hardware que podem destruir componentes úteis para certas aplicações.

A segunda abordagem, por outro lado, permite o uso de um número maior de pinos e é mais flexível, com o custo de maior complexidade de hardware e software. A abordagem requer um microcontrolador adicional e o uso do driver customizado. A conexão com o microcontrolador pode ser dificultada pela voltagem de operação do microcontrolador, que deve ser compatível com a UART da placa (O Arduino Uno, por exemplo, opera em 5V, enquanto a UART da placa opera em 3.3V).

Em termos de efiência, a segunda abordagem é mais custosa, pois para cada alteração de pino é necessário o envio de um comando e a espera de uma resposta, naturalmente gerando uma latência consideravelmente maior do que a primeira abordagem, que permite a manipulação direta dos pinos. Por isso, a segunda abordagem é mais adequada para aplicações que não requerem alta frequência de alterações de pinos, como o controle de LEDs ou a leitura de sensores de baixa frequência.

8 Conclusão

Este trabalho explorou a capacidade de habilitar a transformação de uma TV Box baseada no RK3229 em um dispositivo de IoT, abordando duas estratégias para contornar o desafio da ausência de pinos expostos por padrão. Com diferentes níveis de complexidade, as abordagens permitem a conexão de sensores e dispositivos externos à placa, expandindo suas funcionalidades. Como prova de conceito, ambas as abordagens se mostraram satisfatórias, podendo ser boas opções para aplicações específicas.

Para trabalhos futuros, é possível expandir nas abordagens apresentadas, refinando o código e explorando aplicações específicas. Além disso, a segunda abordagem pode ser adaptada para usos adicionais, como o dos pinos analógicos e PWM do microcontrolador utilizado, que necessitam do uso de outros frameworks do kernel, como o IIO.

O código e outros materiais produzidos para este trabalho estão disponíveis no repositório do projeto no GitHub [1].

Referências

[1] Igor Gabriel Cavalcante de Carvalho Borges. *rk3229-tvbox-gpio*. 13 de dez. de 2023. URL: `https://github.com/discovery-unicamp/rk3229-tvbox-gpio` (acesso em 13/12/2023).

[2] *CSC Armbian for RK322X TV Boxes*. 8 de jan. de 2020. URL: `https://forum.armbian.com/topic/12656-csc-armbian-for-rk322x-tv-boxes/` (acesso em 25/11/2023).

[3] Johan Hovold. *The Serial Device Bus*. 23 de out. de 2017. URL: `http://events17.linuxfoundation.org/sites/events/files/slides/serdev-elce-2017-2.pdf` (acesso em 25/11/2023).

[4] *Manage the GPIO lines from command line with gpiod*. Acme Systems. URL: `https://www.acmesystems.it/gpiod` (acesso em 25/11/2023).

[5] Tote Nunes. "Unicamp vai transformar em computadores aparelhos de TV Box apreendidos pela Receita". Em: (20 de jun. de 2023). URL: `https://www.unicamp.br/unicamp/noticias/2023/06/20/unicamp-vai-transformar-em-computadores-aparelhos-de-tv-box-apreendidos-pela` (acesso em 25/11/2023).

[6] *RK3229 Datasheet V1.2*. RK3229. Fuzhou Rockchip Electronics Co., Ltd. 2017. URL: `https://rockchip.fr/RK3229%20datasheet%20V1.2.pdf` (acesso em 25/11/2023).

[7] "What is a Platform Device in Linux?" Em: (2018). URL: `https://www.learningaboutelectronics.com/Articles/Platform-device-linux.php` (acesso em 25/11/2023).