

Universidade Federal do Rio Grande do Sul  
Escola de Engenharia  
Departamento de Sistemas Elétricos de Automação e Energia  
ENG10032 Microcontroladores

## **Roteiro de Laboratório 9**

### **Comunicação Serial Assíncrona**

Prof. Walter Fetter Lages

28 de maio de 2019

## **1 Objetivo**

O objetivo deste laboratório é verificar o funcionamento das portas seriais da Galileo Gen 2 e em sistemas Unix em geral, mais especificamente do *driver* para terminais seriais. Também será verificado o funcionamento do console serial da Galileo Gen 2.

## **2 Fundamentação Teórica**

A Galileo Gen 2 possui duas portas seriais assíncronas (UARTs). A UART0 está conectada nos pinos IO0 e IO1 do conector de *shield* e a UART1 é reservada para uso do console serial do Linux, mas também está conectada aos pinos IO2 e IO3 do conector de *shield*. Para usar a UART1 para outras finalidades, que não o console do Linux, é necessário desabilitar o seu uso como console. A forma como isso pode ser feito depende do tipo do sistema de inicialização que a distribuição Linux em questão usa.

A distribuição Linux no *firmware* da Galileo Gen2 usa inicialização do tipo System-V. Neste caso, é possível desativar o console serial na UART1 editando o arquivo `/etc/inittab` para desabilitar o *respawn* do programa `getty` para o dispositivo `ttyS1`, comentando a linha:

```
# 1:2345:respawn:/sbin/getty 38400 tty1
```

Por outro lado, a distribuição Linux usada para criar o cartão microSD usa inicialização do tipo systemd. Neste caso, para desativar o programa `getty` é necessário executar o seguinte comando como superusuário:

```
systemctl stop serial-getty@ttyS1.service
```

Neste laboratório será utilizada a UART0 como serial genérica. A UART1 será usada apenas como console serial. Portanto, não será necessário desativar o console serial na UART1.

Ambas UARTs emulam um DTE, ou seja, o pino TX é saída e o pino RX é entrada<sup>1</sup>. A UART0 não implementa sinais de controle. Na Galileo Gen2, a UART1 implementa os sinais de controle #RTS e #CTS, ambos ativos em nível lógico baixo.

Note que a comunicação serial através da interface SPI é síncrona, enquanto as UARTs implementam uma comunicação assíncrona.

**É importante também ter em mente que as UARTs da Galileo trabalham em 3.3 V e portanto não podem ser conectadas diretamente às portas seriais de um PC, ou conversores USB/serial convencionais, que são RS232, e portanto trabalham em -12 V e +12 V.**

## 2.1 Acesso as Portas Seriais

O Linux, como a maioria dos sistemas operacionais, possui um *driver* para as portas seriais. Assim, as portas seriais podem ser acessadas através de dispositivos denominados `/dev/ttyS0`, `/dev/ttyS1` e assim por diante. Estes nomes são derivados do fato de que, originalmente, as portas seriais eram utilizadas para conectar terminais seriais, também chamados de TTYs (*teletypes*), a um computador<sup>2</sup>. Consequentemente, na inicialização do sistema, as portas seriais e os seus *drivers* são configurados da forma mais conveniente para trabalhar com terminais. Por exemplo, ecoando de volta os caracteres que são recebidos. No entanto, para transmissão de dados e para comunicação com outros dispositivos seriais nem sempre deseja-se este comportamento. Assim, o *driver* da porta serial pode funcionar em dois modos<sup>3</sup>:

**Modo Canônico:** Este é o modo *default*, criado para uso com terminais seriais.

A característica principal é que os dados são processados em linhas. Isto significa que a função `read()` vai retornar sempre uma linha. Uma linha é, por *default*<sup>4</sup>, terminada pelo caractere ASCII LF (*Line Feed*, ^J, \n

---

<sup>1</sup>Em dispositivos que emulam um DCE, o pino TX é entrada e o pino RX é saída.

<sup>2</sup>Até o *kernel* 2.2.x eram utilizados também dispositivos denominados `/dev/cua0`, `/dev/cua1`, e assim por diante, para representar as portas seriais conectadas a modems. Atualmente estes nomes são considerados obsoletos e os nomes `/dev/ttyS*` são utilizados independentemente da finalidade da porta serial.

<sup>3</sup>Versões anteriores dos *drivers* seriais do Unix definiam três modos, modo *cooked*, equivalente ao modo canônico, modo *raw*, equivalente ao modo não canônico e modo *cbreak*, um meio termo entre os dois anteriores (os caracteres não são agrupados em linhas, mas os caracteres de controle são processados.).

<sup>4</sup>É possível configurar cada um dos caracteres de controle individualmente.

em linguagem C). O modo canônico também pode tratar diversos outros caracteres de controle, traduzir CR (*Carriage Return*, ^M, \r em linguagem C) para LF e realizar outras ações semelhantes.

**Modo Não Canônico:** Neste modo os caracteres não são agrupados em linhas e caracteres de controle não são processados. É o modo a ser usado para transmitir dados binários de forma transparente.

Os parâmetros de configuração da porta serial são organizados na estrutura `termios` definida em `termios.h` e mostrada na listagem 1.

Listagem 1: Estrutura `termios`.

```
struct termios {
    tcflag_t c_iflag;      /* input mode flags */
    tcflag_t c_oflag;      /* output mode flags */
    tcflag_t c_cflag;      /* control mode flags */
    tcflag_t c_lflag;      /* local mode flags */
    cc_t c_cc[NCCS];       /* control characters */
};
```

As *flags* de entrada controlam o processamento de entrada (recepção pela linha serial) de caracteres, as *flags* de saída controlam o processamento de saída (transmissão pela linha serial) de caracteres, as *flags* de controle afetam os parâmetros da linha serial (status do modem, número de *stop* bits, etc.) e as *flags* de modo local controlam a interação entre o usuário e o *driver*, determinando se os caracteres são ecoados, se sinais são enviados ao programa, etc. O array `c_cc` define os caracteres de controle, cujos valores *default* estão definidos em `termios.h`.

Os valores configurados na estrutura `termios` podem ser obtidos através da função:

```
int tcgetattr(int fd, struct termios *termios_p)
```

e podem ser ajustados utilizando-se a função:

```
int tcsetattr(int fd, int optional_actions,
              struct termios *termios_p)
```

onde a *flag* `optional_actions` pode ter os valores:

**TCSANOW** as alterações ocorrem imediatamente.

**TCSADRAIN** as alterações ocorrem após a saída escrita em `fd` ter sido transmitida.

**TCSAFLUSH** as alterações ocorrem após a saída escrita em `fd` ter sido transmitida e toda a entrada recebida mas não lida é descartada antes da alteração ser realizada.

Embora os campos da estrutura `termios` possam ser manipulados diretamente para configurar a porta serial, existem funções para auxiliar esta manipulação. As principais são:

**`void cfmakeraw(struct termios *termios_p)`** utilizada para ajustar os campos da estrutura com os valores adequados para o modo não canônico. Ou seja, configura os campos de `termios_p` como mostrado na listagem 2

Listagem 2: Ajuste dos campos da estrutura `termios` para modo não canônico.

```
termios_p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP
    |INLCR|IGNCR|ICRNL|IXON) ;
termios_p->c_oflag &= ~OPOST;
termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN) ;
termios_p->c_cflag &= ~(CSIZE|PARENB) ;
termios_p->c_cflag |= CS8;
```

**`speed_t cfgetispeed(const struct termios *termios_p)`** obtém a taxa de recepção

**`speed_t cfgetospeed(const struct termios *termios_p)`** obtém a taxa de transmissão

**`int cfsetispeed(struct termios *termios_p, speed_t speed)`** configura a taxa de recepção

**`int cfsetospeed(struct termios *termios_p, speed_t speed)`** configura a taxa de transmissão

**`int cfsetspeed(struct termios *termios_p, speed_t speed)`** configura as taxas de transmissão e recepção para o mesmo valor.

As taxas de transmissão e recepção são especificadas através das constantes: B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, B38400, B57600, B115200 e B230400<sup>5</sup>.

---

<sup>5</sup>Esta taxa não pode ser obtida com o *hardware* padrão do PC.

## 2.2 Console Serial

Como mencionado anteriormente, na Galileo a UART1 é usada como console serial do Linux, cujo conector é mostrado na Figura 1. Isto significa que o Linux está configurado para tratar esta UART como se houvesse um terminal serial (ou um emulador de terminal) conectado a ela.



Figura 1: Conector de console da Galileo Gen2.

Esta UART aparece para o Linux executando na Galileo como `/dev/ttyS1` e por *default* há um gerenciador de *login* monitorando esta UART e aguardando o *login* de um usuário.

**Como as UARTs da Galileo trabalham em 3.3V, não se pode conectar diretamente um terminal ou emulador que funcione com o padrão RS-232.** Por outro lado, a maioria dos PCs atuais também não tem mais portas seriais RS-232. Assim, para que se possa usar um emulador de terminal (minicom, seyon, screen, hyperterminal, etc.) com a Galileo é necessário utilizar um conversor USB/serial que opere em 3.3V. **Note que a maioria dos cabos USB/serial disponíveis no mercado utiliza níveis de tensão RS-232 e não 3.3V.** Esta interface serial com sinais de controle RS-232, mas operando em 3.3V é conhecida comercialmente como FTDI<sup>6</sup>.

No laboratório será usado um conversor USB/FTDI como mostrado na Figura 2. Note que existem diversos fabricantes de conversores similares, mas cuja pinagem é diferente e não diretamente compatível com a Galileo Gen2. O conversor disponível no laboratório tem uma pinagem tal que pode ser conectado diretamente no conector da UART da Galileo Gen2, criando um null-modem. Porém, este conversor não implementa o sinal `#RTS`. No seu lugar está o sinal `#DTR`. Ou seja, é usada uma sinalização `#DTR/#CTS` ao invés de `#RTS/#CTS`, que seria o correto. Mas isso não faz muita diferença na maioria das aplicações que

---

<sup>6</sup>FTDI é o nome do fabricante do *chip* mais comum para converter USB em serial 3.3V porém com a mesma semântica da RS-232.

usam um computador como emulador de terminal, pois se o computador for bem mais rápido do que a Galileo, não há real necessidade de controle de fluxo por *hardware*.



Figura 2: Conversor USB/FTDI.

Quando conectado à porta USB de um PC, este conversor aparece como um dispositivo `/dev/ttyUSB?` e pode ser usado da mesma forma que uma porta serial nativa.

## 3 Experimentos

### 3.1 Configuração da Galileo para usar a UART0

1. Configure o Linux na Galileo para que seu usuário tenha acesso aos dispositivos seriais. Para isso, inclua o seu usuário no grupo `dialout`. Note que este grupo já existe e que o dispositivo `/dev/ttyS0` já está configurado para poder ser acessado por membros deste grupo.
2. Baixe do Moodle <<http://moodle.ece.ufrgs.br>> o *script* de inicialização para configurar os pinos de I/O da Galileo Gen 2 para uso da UART0. Instale o *script* e reinicialize a Galileo.

### 3.2 Comunicação em Modo Canônico

3. O programa mostrado na listagem 3 transmite pela porta serial em modo canônico o texto digitado no teclado. Note que este programa é compatível com qualquer sistema Unix, não apenas com a Galileo. As particularidades da Galileo foram configuradas no *script* de inicialização.

### Listagem 3: Transmissão em modo canônico.

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int fd;
    struct termios tty;
    char c;

    if (argc != 2)
    {
        printf("Canonical mode transmission\n");
        printf("\tUsage: \t%s <device>\n", argv[0]);
        return -1;
    }
    if ((fd=open(argv[1], O_RDWR)) == -1)
    {
        perror(argv[0]);
        return -errno;
    }

    if (tcgetattr(fd, &tty) )
    {
        perror(argv[0]);
        return -errno;
    }

    if (cfsetspeed(&tty, B9600))
    {
        perror(argv[0]);
        return -errno;
    }

    if (tcsetattr(fd, TCSANOW, &tty) )
    {
        perror(argv[0]);
        return -errno;
    }
    do
    {
        c=getchar();

        if (write(fd, &c, 1) == -1)
        {
            perror(argv[0]);
            return -errno;
        }
    } while (c != 0x2b);

    if (tcdrain(fd))
    {
        perror(argv[0]);
        return -errno;
    }

    if (close(fd))
    {

```

```

        perror(argv[0]);
        return -errno;
    }

    return 0;
}

```

4. Faça um `Makefile` e compile o programa para executar na Galileo Gen 2.
5. Conecte o pino TX da UART0 no pino RX da UART0 através de um *jumper*.
6. Configure a UART0 para a taxa de 9600 bps sem echo local com o comando:

```
stty -F /dev/ttyS0 9600 -echo
```

7. Teste o programa da Listagem 3 com uma taxa de 9600 bps, transmitindo pela UART0. Para receber os dados, abra outro terminal na Galileo Gen 2 e execute o comando:

```
cat /dev/ttyS0
```

8. Faça um programa para receber pela porta serial em modo canônico e exibir o resultado na tela. O nome do dispositivo deve ser passado como parâmetro na linha de comando.
9. Teste os programas desenvolvidos nos itens 3 e 8.

### 3.3 Comunicação em Modo não Canônico

10. Repita os itens 3, 8 e 9 utilizando o modo não canônico.

### 3.4 Console Serial

11. Conecte a Galileo ao *host* usando o conversor USB/FTDI. Tenha cuidado para alinhar o pino GND do conversor com o pino GND da Galileo (pino 1 do conector, na extremidade mais longe da borda da placa), como mostra a Figura 3
12. Ao conectar o conversor ao PC será criado um novo dispositivo serial cujo nome terá a forma `/dev/ttyUSB?`. Utilize no *host* o comando:

```
ls -l /dev/ttyUSB?
```

para identificar qual o dispositivo representa a serial recém criada.





Figura 3: Conversor USB-FTDI conectado na Galileo Gen2.

13. Abra um emulador de terminal com o comando<sup>7</sup>:

```
minicom -D <dispositivo> -b 115200 -o
```

onde <dispositivo> é o dispositivo identificado no item 12.

Para usar o seyon, o comando seria:

```
seyon -modem <dispositivo>
```

Para usar o screen, o comando seria:

```
screen -fn <dispositivo> 115200
```

14. Em outro terminal desative o controle de fluxo por *hardware*, pois o GRUB não usa controle de fluxo, com o comando:

```
stty -F <dispositivo> -crtcts
```

O controle de fluxo também pode ser desativado através do menu do `minicom` (CTRL-a z o). Infelizmente, o `minicom` reabilita o controle de fluxo por *hardware* ao ser inicializado, portanto é necessário desativa-lo após a inicialização do `minicom`.

---

<sup>7</sup>O emulador de terminal mais conhecido no Linux é o `minicom`, que opera apenas em modo texto. O `seyon` é um emulador de terminal que opera em modo gráfico.

15. Reinicialize a Galileo e observe o emulador de terminal. Note que através do console serial é possível acessar o *bootloader* GRUB e ver as mensagens emitidas pelo Linux durante o *boot*, além de poder se logar na Galileo.
16. Utilize o comando CTRL-a z x para sair do `minicom`.