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

Roteiro de Laboratório 2
Ambiente de Desenvolvimento

Prof. Walter Fetter Lages

28 de agosto de 2019

1 Objetivo

O objetivo deste laboratório é familiarizar os alunos com a o ambiente de desenvolvimento da Galileo Gen2 em Linux. A princípio, seria possível instalar no cartão microSD uma distribuição de Linux "normal" e utilizar as ferramentas de desenvolvimento executando na própria Galileo. No entanto, como o processador da Galileo não é muito potente, a compilação certamente seria demorada. Outro ponto a considerar é que a memória *flash* da Galileo não é muito grande e para a versão final da aplicação seria interessante poder executá-la diretamente a partir da *flash*, sem a necessidade do cartão microSD. Para isso, é necessário que o Linux executando na Galileo tenha um pequeno *footprint* o que requer o uso de versões especiais das bibliotecas que não suportam adequadamente ambientes de desenvolvimento mais sofisticados.

A utilização de um ambiente de desenvolvimento independente da plataforma alvo, também torna o desenvolvimento menos dependente da Galileo em si e similar aos ambientes de desenvolvimento de outras plataformas para sistemas embarcados.

Ao final deste laboratório, espera-se que os alunos sejam capazes de compilar um programa em C para executar na Galileo utilizando `Makefile` e compilador cruzado e perceber as diferenças entre programas desenvolvidos para a plataforma *host* e para a plataforma alvo.

O ambiente de desenvolvimento da Galileo é baseado no projeto Yocto, que utiliza a ferramenta BitBake para criar imagens de sistemas Linux para sistemas embarcados e o OpenEmbedded como sistema de *build*. Estes componentes formam um sistema de *build* de referência chamado Poky.

Utilizando o Poky se pode criar uma distribuição Linux específica para a Galileo com pequeno *footprint* e as bibliotecas e utilitários necessários para a apli-

cação em questão. Além da distribuição Linux, cria-se também um conjunto de ferramentas de desenvolvimento cruzado que utilizam as bibliotecas incluídas na distribuição.

Mais adiante, haverá um laboratório onde será criada uma distribuição e as respectivas ferramentas de desenvolvimento. No entanto, por hora, serão utilizada as ferramentas de desenvolvimento já compiladas e incluídas no *Intel System Studio IoT Edition*, que executa no Linux em 64 bits. Note que 64 bits aqui se refere à plataforma onde as ferramentas serão executadas (sistema *host*), o código gerado para a Galileo (sistema alvo) tem que ser sempre de 32 bits.

Nas máquinas do laboratório o ambiente de desenvolvimento já está instalado. Para instalar em outras máquinas, veja os detalhes no apêndice A.

2 Experimentos

1. Configure as variáveis de ambiente do seu usuário para utilizar o *Intel System Studio IoT Edition* com os comandos:

```
export DEVKIT=/opt/iot-devkit/devkit-x86
export PATH=$PATH:$DEVKIT/sysroots/x86_64-pokysdk-linux/usr/bin/i586-poky-linux
export CROSS_COMPILE=i586-poky-linux-
export ARCH=x86
```

   Estes comandos podem ser inseridos no seu arquivo `~/.profile`, para serem executados automaticamente sempre que for feito *login* no sistema ou inseridos em um arquivo de *script* com o nome, por exemplo, `iss_setup.sh` que deve ser executado para configurar o ambiente após o *login* ou a abertura de um novo terminal. Para executar o *script* no *shell* deve ser usado o comando:

```
source iss_setup.sh
```

2. Crie um diretório no *host* para colocar os códigos fontes dos programas. Habitue-se também a criar um diretório para cada programa. Neste diretório deve haver o código fonte do programa e um `Makefile` para compilá-lo.

3. A Listagem 1 mostra um programa “Hello, World!” em C.

   Este programa pode ser compilado com o `Makefile` mostrado na Listagem 2.

   Digite os dois arquivos ou baixe-os do Moodle.

4. Compile o programa executando o comando:

Listagem 1: Programa "Hello, World!"em C.

```
1  #include <stdio.h>
2
3  int main(int argc,char *argv[])
4  {
5          printf("Hello, world!\n");
6
7          return 0;
8  }
```

Listagem 2: `Makefile` para compilação para execução no *host*.

```
1  TARGET=hello
2  SRCS=$(TARGET).c
3
4  FLAGS=-O2 -Wall -g -MMD
5  INCLUDE=-I.
6  LIBDIR=
7  LIBS=
8
9  CC=$(CROSS_COMPILE)gcc
10 CFLAGS=$(FLAGS) $(INCLUDE)
11 LDFLAGS=$(LIBDIR) $(LIBS)
12
13 all: $(TARGET)
14
15 $(TARGET): $(SRCS:.c=.o)
16         $(CC) -o $@ $^ $(LDFLAGS)
17
18 %.o: %.c
19         $(CC) $(CFLAGS) -c -o $@ $<
20
21 -include $(SRCS:.c=.d)
22
23 clean:
24         rm -f *~ *.bak *.o *.d
25
26 distclean: clean
27         rm -f $(TARGET)
```

```
make
```

5. Transfira para a Galileo o executável recém compiladoo:

   ```
   scp hello <login>@<galileoname>:
   ```

   onde `<login>` é o seu *login* na Galileo e `<galileoname>` é o nome da Galileo.

6. Faça *login* na Galileo e verifique que o arquivo efetivamente encontra-se lá.

7. Execute o programa para constatar o seu funcionamento.

8. Para depurar os programas na Galileo, é conveniente usar um depurador remoto, pelos mesmos motivos que se usa um compilador cruzado. É possível depurar remotamente os programas na Galileo usando o `gdb`. Na prática é mais conveniente usar um *front-end* para ele como o `kdbg` ou o `ddd`. Para tanto, deve ser executado na Galileo o `gdbserver`:

   ```
   gdbserver <host>:<port> hello
   ```

   onde `<host>` é o nome do *host* onde será executado o `gdb` ou seu *front-end* e `<port>` é a porta TCP a ser usada para comunicação. Tipicamente, deve ser escolhida uma porta acima de 1024, para evitar problemas de permissões de acesso.

9. Para usar o `kdbg` execute no *host* o comando:

   ```
   kdbg -r <target>:<port> hello
   ```

   onde `<target>` é o nome da Galileo e `<port>` é a porta TCP a ser usada para comunicação.

   A rigor, o `kdbg` deveria ser configurado para usar o `i586-poky-linux-gdb` do *Intel System Studio IoT Edition*, mas como o *host* possui a mesma arquitetura, é possível usar a configuração *default*, com o `gdb` nativo.

10. Faça um "Hello, World!" em C++ e compile-o com um `Makefile` adequado para executar na Galileo.

A Instalação do Ambiente de Desenvolvimento

Nas máquinas do laboratório o ambiente de desenvolvimento já está instalado, portanto esta seção pode ser desconsiderada. Ela está aqui apenas para o caso de ser desejado instalar as ferramentas em outra máquina. Para tanto, faça o seguinte, no *host*, como superusuário:

1. Baixe e descompacte o arquivo como o *Intel System Studio IoT Edition* no diretório `/opt`:

```
cd /opt
tar -xjf iss-iot-linux_03-24-16.tar.bz2
rm iss-iot-linux_03-24-16.tar.bz2
```

2. Ajuste as permissões dos arquivos descompactados:

```
chown -R root.root iss-iot-linux
chmod -R go-w iss-iot-linux
chmod -R -s iss-iot-linux
```

3. Ainda no diretório `/opt`, crie um *link* para a versão atual do *Intel System Studio Iot Edition*:

```
ln -s iss-iot-linux iot-devkit
```

4. Crie um *link* para o diretório das ferramentas de desenvolvimento:

```
cd iot-devkit
ln -s devkit-x86 1.7.2
```

5. Execute a relocação das ferramentas de desenvolvimento:

```
INSTALL_DIR=`pwd` sdk-relocator/relocate_sdk.sh
```

   Atenção: os nomes dos *links* tem que ser exatamente estes, pois as ferramentas cruzadas foram compiladas para serem instaladas nestes diretórios. A princípio, o *script* de relocação altera os *paths* que foram codificados nos programas para que funcionem quando o *Intel System Studio IoT Edition* é instalado em qualquer outro lugar, mas os diretórios onde são procurados os arquivos de cabeçalho e bibliotecas padrão continuam os originais, o que exigiria que os novos diretórios fossem sempre especificados na linha de comando através de opções de compilação.

B Voltar a Usar o Compilador Nativo

Lembre-se que se a variável de ambiente `CROSS_COMPILE` estiver configurada, será utilizado o compilador cruzado, ao invés do compilador nativo. Para "limpar" a variável de ambiente, de forma a voltar a usar o compilador nativo, pode-se usar o comando:

```
export CROSS_COMPILE=
```

C Outros *Targets* do `Makefile`

Além do *target* `all`, que é a entrada *default* no `Makefile`, é comum a existência dos *targets* `clean` para limpar os arquivos temporários criados no processo de compilação, mas sem remover os arquivos finais da compilação, e `distclean` para remover todos os arquivos que podem ser recriados automaticamente, inclusive os arquivos finais da compilação.