Проектиране на вградени автомобилни електронни системи
Лабораторно упражнение №21

Работа с Xilinx Vivado и Vitis. Крослинкер LD. Преместване на изпълнимия код в RAM. Статични и динамични библиотеки.

==========

1. Превключете джъмпера вдясно на платката на позиция JTAG. Свържете µUSB кабел към PROG/UART USB куплунга. Включете платката от ключа ON/OFF. Включете USB-към-UART конвертор към сигнали JB1_P (T20), отговарящ на Uartlite_TxD, и JB1_N (U20), отговарящ на Uartlite_RxD. Модулът Uartlite ще бъде свързан към един MicroBlaze.

2. Стартирайте две копия на Cutecom (или gtkterm) и отворете портовете, отговарящи на printf канала на дебъгера и на Uartlite. Стартирайте терминал с CTRL + ALT + t и изпълнете командите:

```
cd /home/user/workspaces/xilinx_workspace
source /home/user/programs/Xilinx/Vitis/2022.2/settings64.sh
```

3. Променете програмата за изчисляване на средноквадратична стойност от миналото лабораторно упражнение, така че да използва библиотечни функции за изчисляване на коренквадратен и умножение на квадрат. За деление използвайте операторът / на C. Направете един отделен сорс и отделен хедърен файл, в който да поместите имплементацията на вашата функция. Включете хедърния файл “math.h” и добавете към линкера (в основния Makefile) аргумент за включване на стандартната библиотека за математика -lm.

ВНИМАНИЕ: включете библиотеката в края на реда с командата за линкване (най-дясно).

Линкерът на GCC се нарича LD и аналогично на асемблера може да бъде извикван през GCC. По-важни аргументи на линкера са дадени в таблицата по-долу [3] [4]. Забележете, че статичните библиотеки трябва да имат префикс “lib” и окончание “.a”, които обаче не се включват в аргумента.

Аргумент на LD	Значение
-L[директория-с-библиотеки]	Указва директория през GCC, където линкера ще търси обектовите файлове на библиотеката (не е задължителен, когато са стандартни библиотеки)
-l[име-на-статична-библиотека]	Указва името на обектовия файл (през GCC) на библиотеката. Не се пише префикса “lib” и вида на файла “.a”.
-T[име-на-линкерен-скрипт]	Включва конфигурационен файл за линкера (през GCC), в който са описани сегментите на паметта за дадената система. Трябва да е с разширение “.ld”.
-Wl,[линкерен-аргумент]	Предава аргумент на линкера LD през командата GCC.
`-Wl,--start-group [няколко-линкерни-аргумента] -Wl,--end-group`	Предава два или повече аргумента на линкера LD през командата GCC.
-M (или --print-map)	Създай файл с информация за това къде даден обектов файл ще се намира в паметта, кои библиотеки са включени, както и инициализацията на глобалните променливи (символи). Предава се през GCC.
--gc-sections	Изключи кода от обектовите файлове на функции, които не се използват никъде в останалта част от програмата. Тази опция **намалява значително обема** на изпълнимия файл и е валидна, само ако се предаде -ffunction-sections аргумента на компилатора.
--print-memory-usage	Принтирай използваната ROM/RAM памет.
-nostdlib	Търси директории на библиотеки единствено указани с аргумента -L. Директории, указани в линкерния скрипт ще бъдат игнорирани.
--print-multi-lib	Списък с всички поддържани набори от инструкции за конкретния порт на GCC. Предава се през GCC.

Примери:
Библиотека, която се казва m трябва да бъде записана във файл с име libm.a и да бъде включена към линкера по следния начин:

arm-none-eabi-gcc -mcpu=cortex-m7 -nostartfiles --specs=nosys.specs -mfloat-abi=hard -mfpu=fpv5-d16 -Tscript.ld main.o -o main.axf -lm

Библиотека, която се казва gcc трябва да бъде записана във файл с име libgcc.a и да бъде включена към линкера по следния начин:

arm-none-eabi-gcc -mcpu=cortex-m7 -nostartfiles --specs=nosys.specs -mfloat-abi=hard -mfpu=fpv5-d16 -Tscript.ld main.o -o main.axf -lgcc

и т.н.

4. Заместете командата за линкване в top-level Makefile-а ви от GCC на LD. При такова директно извикване е необходимо изрично да се укажат директориите на всички библиотеки, включително и стандартните. За случая може да помогне линкерният аргумент -Map=./debug/main.map:

mb-gcc ... -Wl,-Map=./debug/main.map …

който ще принтира във файла main.map всички библиотеки, които GCC използва вътрешно, когато минава през фазите асемблиране и линкване (потърсете раздела “Archive member included to satisfy reference by file (symbol)”). Алтернативен метод е да се предаде -v на gcc, когато работи в режим линкер.
От main.map се вижда, че пътят до системните библиотеки на MicroBlaze е:

/home/user/programs/xilinx/Vitis/2022.2/gnu/microblaze/lin/bin/../microblazeeb-xilinx-elf/usr/lib/le/bs/p/m/fpd

От същия файл се вижда, че са включени следните обектови/архивни файлове:
crt0.o
crti.o
crtn.o
crtbegin.o
crtinit.o
crtend.o
libgloss.a
libm.a
libc.a
libgcc.a
libxil.a (този файл се намира в директорията на проекта ви, а не в ....p/m/fpd)

ВНИМАНИЕ: укажете на линкера да генерира .elf файл с little-endian формат. Това става с аргумент -EL. (-EB е за big-endian)

ВНИМАНИЕ: премахнете всички аргументи за асемблера (-m), а аргументите за предаване на линкерни аргументи (-Wl,) трябва да се премахнат, но

параметрите им трябва да останат. Например:
Wl,-T -Wl,lscript.ld -Wl,--no-relax -Wl,--gc-sections

се предава на LD по следния начин:

-Tlscript.ld --no-relax --gc-sections

ВНИМАНИЕ: редът на включване на библиотеките има значение. Включете първо -lgloss, след това -lm, след това -lc, след това -lgcc и накрая -lxil.

Заредете програмата във FPGA и вижте резултатът от изчислението на RMS в Cutecom – той трябва да е идентичен с предишните задачи. Светодиод LD3 трябва да започне да мига.

5. Добавете динамично заделяне на памет във вашата програма. То изисква отделен сегмент в RAM паметта, наречен хийп. Копирайте сорс файловете от директория 21_5 във вашия проект. Добавете target за sbrk.c файла и компилирайте програмата. Заредете я във FPGA и стартирайте дебъг сесия. Изпълнете програмата веднъж, за да научите стартовия адрес на HEAP сегмента. Изпълнете програмата втори път, като преди да заделите и напълните съответния масив изберете в Eclipse → Window → Show View → Other → Memory и натиснете Enter. От бутона “Add Memory Monitor” въведете началния адрес на HEAP региона. Изпълнявайте програмата стъпка по стъпка и наблюдавайте как заделената памет се пълни. Обърнете внимание на допълнителната служебна информация, която malloc( ) добавя преди и след заделената памет.

ВНИМАНИЕ: ако се използва Eclipse, таб Memory не се опреснява след всяка Step Over операция. Затова може да се сложат точки на прекъсвания на for-циклите, които пълнят динамично заделените масиви. След това премахнете и добавете наново началния адрес на heap-а.

ВНИМАНИЕ: за да използвате функцията malloc е необходимо да използвате runtime библиотеката newlib (-lc). Тази библиотека може да се окаже много голяма за някои микроконтролери, затова се използва нейна редуцирана версия. Тази версия съдържа системни функции с празни тела, или функции с минимална имплементация. В този случай malloc( ) е имплементирана, но една нейна функция липсва – sbrk( ), която трябва да бъде добавена от програмиста.

6. Изследвайте влиянието на премахването на неизползваните функции (dead code removal) от вашия код. За целта на компилатора в target-а за main.o предайте аргумента:

```
-ffunction-sections
```

а на линкера:

```
--gc-sections
```

Компилирайте проекта и си запишете данните от mb-size. Премахнете двата аргумента. Компилирайте отново проекта и си запишете данните от mb-size. Сравнете данните в двата случая. За настоящото лабораторно трябва да се получи редукция от около 4.7 %, но в други проекти, с много по-големи размери на статичната библиотека, е възможно да се получи редукция от порядъка на 67%.

Тези аргументи принуждават компилатора да постави всяка една функция в отделен регион от паметта, а след това линкера прави анализ кои функции се викат, и кои не, и неизползваните ги премахва.

7.Направете статична библиотека от вашите сорс файлове – led.c, printf.c и uart.c. Статичната библиотека представлява архив от обектови файлове. Вашите обектови файлове, съответсващи на въпросните сорс файлове са готови и се намират в директория ./debug. За да ги архивирате използвайте командата [6]:

```
mb-ar -r [libиме-на-библиотека.a] [файл–1.o] [файл–2.o] … [файл–n.o]
```

Добавете отделен target в top-level Makefile-а ви само за създаване на библиотеката (например нека “make static_lib” да прави .a файла). След това добавете току-що създадената библиотека в нов target, подобен на main.elf, но вместо да се предават обектовите ви файлове led.o, uart.o, print.o, предайте вашия .a файл.

ВНИМАНИЕ: понеже printf зависи от вътрешните функции (instrinsic functions):

__umodsi3
__umoddi3
__nedf2
__ltdf2
__subdf3
__fixdfsi
__floatsidf
__subdf3
__muldf3 и др.

които се намират в libgcc.a, то включването на вашата статична библиотека трябва задължително да бъде преди -lgcc.

Заредете и тествайте програмата ви. Ако всичко е минало успешно, тя трябва да работи по същия начин като в предишните задачи. Дебъгването на функциите от библиотеката ще съдържа сорс кода, но ако искате да създадете комерсиална библиотека, то трябва да премахнете аргумента -g при компилацията на съответните сорс файлове преди да ги архивирате. В този случай дебъгерът ще покаже само Асемблерни инструкции без C код.

8. Разгледайте вътрешните (intrinsic) функции, поддържани от компилаторът GCC. В [1] е даден списък с функции конкретно за MicroBlaze, а в [2] са изброени всички вътрешни функции, които са общи за всички архитектури, поддържани от GCC. Спрямо inline Асемблерът, вътрешните функции са по-елегантен начин за достъпване на специфичен изчислителен ресурс на един микропроцесор – например операции с числа с плаваща запетая и двойна прецизност.

9. Програмата mb-size може да работи върху архиви. Тогава тя показва пълен списък с обектовите файлове от архива, както и заеманото място в паметта на контролера от всеки един файл. Добавете отделен target, който да извиква mb-size с входен файл – вашата библиотека.

10. Динамичните библиотеки се различават от статичните по това, че кодът в тях не се добавя към изпълнимия файл на потребителската програма и по това, че адресите в обектовите файлове са условни и се разбират чак когато операционната система зареди библиотеките в паметта. Програмата от ОС, която прави това се нарича динамичен линкер (dynamic linker / dynamic loader). Динамичният линкер изчислява адресите **при стартирането** на програмата, което забавя малко изпълнението на самата програма в началото, но пък позволява по-лесното ѝ ъпдейтване.

Пример за създаване на динамична библиотека (.so) във вградена система, която работи с Linux е (логнете се на самата система) [8]:

```
gcc -c -fpic my_library.c -o my_library.o
gcc -shared -o libmy_library.so my_library.o
gcc main.c -L[път/до/директорията/на/libmy_library.so] -o main.bin -lmy_library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[път/до/директорията/на/
libmy_library.so]
./main
```

Съществуват и динамично-линквани (динамично зареждаеми) библиотеки (.dll), при които кодът на библиотеката се извиква от програмата, **по време на нейното изпълнение**, т.е. когато се стигне до извикване на системна функция за зареждане на DLL. При .so файлът на библиотеката трябва да е наличен при стартирането, докато при .dll той трябва да е наличен, когато се достъпва определена функция от приложението.

11. Разгледайте линкерния файл на вашата програма. Опитайте се да преместите изпълнението на кода от адрес 0x6000 нагоре (т.е. да преместите .text сегмента и всички сегменти след него от адрес 0x00000050 на 0x00006000). Адресите на инструкциите могат да бъдат видяни в mb-gdb с командата stepi. Всяка инструкция в обектовия файл притежава два вида адреси [9]: LMA (load memory address) и VMA (virtual memory address). LMA е адресът, на който инструкцията ще бъде записана при програмирането на FPGA. VMA е адресът, от който инструкцията ще бъде изпълнена от микропроцесора.

* * *

[1] “C/C++ Language Reference”, Technical Reference TR0173 (v4.0), Altium Limited, 2009.
[2] Richard Stallman, “Using the GNU Compiler Collection”, GNU Press, chapter 6 “Extensions to the C Language Family”, section 59 “Other Built-in Functions provided by GCC”, online, 2023.
[3] Gatliff W., “An Introduction to the GNU Compiler and Linker”, Embedded Systems Programming Magazine, 2002.
[4] https://www.rowleydownload.co.uk/arm/documentation/gnu/ld/Options.html
[5] https://sourceware.org/newlib (Docs секцията)
[6] https://sourceware.org/binutils/docs/binutils/ar.html
[7] “Dynamic Linking with the ARM Compiler toolchain”, Application note 242, ARM Ltd, 2010.
[8] https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html
[9] S. Chamberlain, I. Taylor, “The GNU Linker”, Free Software Foundation, Inc, 2009.

доц. д-р инж. Любомир Богданов, 2023 г.