ChatGPT

Приступ меморијски мапираним регистрима

Меморијски мапирани I/O у микроконтролерима. У типичној *embedded* архитектури, периферни уређаји се контролишу путем меморијски мапираних регистара – посебних хардверских регистара који су изложени у заједничком адресном простору процесора. Део расположивих адреса рачунара резервисан је за ове уређаје, па упис података на одређену меморијску адресу заправо шаље податак периферном уређају, док читање са те адресе доводи до очитавања податка из уређаја [1]. Ово значи да се исте инструкције које CPU користи за приступ обичној меморији (нпр. *load/store* операције) могу користити и за приступ периферијама [2]. Хардверски адресни декодер на системској магистрали препознаје да ли дата адреса припада меморији или уређају и усмерава сигнале и податке ка одговарајућој компоненти [3]. За разлику од тзв. *port-mapped* I/O приступа (карактеристичног за неке раније архитектуре са посебним I/O инструкцијама), меморијски мапиран I/O поједностављује дизајн процесора и омогућава јединствен и ефикасан начин комуникације са уређајима, у складу са RISC филозофијом [4].

Адресни простор и распоред периферија. Микроконтролери обично имплементирају *Von Neumann* модел меморије са јединственим адресним простором за програмски код, податке и периферије. На пример, ARM Cortex-M архитектура дефинише 4 GB адресни простор подељен на регионе за Flash (код), SRAM и периферије [5]. Типично је велики блок од 512 MB резервисан за регистре on-chip периферних уређаја – код ARM Cortex-M језгара овај *Peripheral* регион обухвата адресе отприлике од 0x4000_0000 до 0x5FFF_FFFF [6]. У том опсегу смештени су регистри разноврсних модула као што су GPIO, тајмери, UART, A/D конвертори и др; сваки уређај добија сопствени подпојас адреса за своје регистре [7]. Читањем или писањем на било коју адресу у оквиру тог опсега, CPU у ствари приступа одговарајућем регистру периферије. Поред уобичајених периферија, и поједини системски контролни регистри (нпр. регистри за управљање прекидима, тактом или дебагом) такође су мапирани у посебан регион адресног простора – на пример, *Private Peripheral Bus* регион око адресе 0xE000_0000 код ARM Cortex-M садржи NVIC, SysTick и друге кључне регистре језгра [8]. Овако дефинисана меморијска мапа поједностављује пројектовање *boot* софтвера и олакшава преносивост програма, јер сва Cortex-M језгра имају сличну организацију адресног простора за основне компоненте система [9] [10].

Приступ регистрима у програму (C језик). Са становишта софтвера, рад са меморијски мапираним регистрима своди се на уписивање и читање одређених адреса у меморији. Језик C омогућава веома директан приступ – корисник може декларацијом показивача на дату адресу или коришћењем одговарајућег *header*-а читати и мењати вредности хардверских регистра као да су промењљиве у меморији. Међутим, да би се очувала исправна семантика, неопходно је те промењљиве означити као `volatile`. Кључна реч `volatile` упозорава компајлер да се вредност дате промењљиве може мењати изван тренутног програма (нпр. од стране хардвера или другог *thread*-а) те да не сме оптимизовати приступе – сваки упис или читање у изворном коду мора резултирати стварним уписом или читањем на датој адреси [11] [12]. У супротном, могло би се десити да компајлер негенерише очекивану инструкцију (нпр. ако „закључи“ да се вредност није променила) или да је задржи у регистру процесора, што би нарушило комуникацију са уређајем [13]. Из тог разлога, регистарске константе у *header*-има микроконтролера увек су декларисане као `volatile`.

Моделирање хардверских регистра у C. Да би се олакшало коришћење меморијски мапираних регистра, у пракси се примењује техника мапирања регистара на C структуре. Идеја је да се дефинише *typedef struct* чија поља тачно одговарају регистрима једног периферног модула редом којим су они распоређени у меморијском простору [14]. Затим се креира показивач (или макро) на ту структуру на базној адреси периферије. На тај начин, сваки регистар се може именовано адресирати преко поља структуре уместо преко *„магичних“* хексадецималних константи. Ознака `volatile` се може применити на саму структуру или на показивач, чиме се гарантује да ће сваки приступ полима структуре заиста приступити физичком регистру [15] [16]. Практично сваки савремени произвођач микроконтролера уз своје уређаје испоручује и одговарајуће заглавље са већ унапред дефинисаним структурама и базним адресама периферија. ARM је стандардизовао овај приступ кроз CMSIS (*Cortex Microcontroller Software Interface Standard*), па се у CMSIS *header*-има налазе описне структуре и макрои за све регистре циљаног система [17] [18]. На пример, у наставку је приказана поједностављена дефиниција једног GPIO модула и коришћење његових регистара:

```
typedef struct {
    volatile uint32_t IN;    // регистар улазних вредности пинова
    volatile uint32_t OUT;   // регистар излазних вредности пинова
    volatile uint32_t DIR;   // регистар правца (0 = улаз, 1 = излаз)
    // ... остали регистри периферије
} GPIO_TypeDef;

#define GPIO ((GPIO_TypeDef *) 0x50000000UL)  // базна адреса GPIO модула

// Пример употребе:
GPIO->DIR |= 0x1;   // поставља пин 0 као излаз
GPIO->OUT = 0x1;    // поставља логичку '1' на пин 0
```

Горњи код илуструје принцип меморијски мапираног приступа периферији. Структура `GPIO_TypeDef` декларативно описује низ од три 32-битна регистра – замислимо да су то улазни, излазни и регистар правца GPIO порта. Макро `GPIO` дефинише показивач на ову структуру на меморијској адреси *0x50000000*, за коју претпостављамо да је базна адреса одговарајућег GPIO контролера у датом микроконтролеру. Када у програму извршимо наредбу `GPIO->OUT = 0x1;`, компајлер ће генерисати машинску инструкцију за упис вредности 1 на меморијску адресу која одговара регистру *OUT* тог модула (нпр. инструкцију `STR` на ARM архитектури) [19]. Овим уписом се хардверски излаз на пину 0 поставља на високи ниво (под условом да је тај пин претходно конфигурисан као излаз, као у примеру где се `GPIO->DIR` подешава). Читљивост је знатно побољшана – уместо неразумљивог израза `*(volatile uint32_t *)(0x50000004) = 0x1;` који директно адресира меморију, програмер користи симболичко име `GPIO->OUT`, што јасно означава шта се догађа. Савремени преводиоци ће овакву употребу структура оптимизовати једнако ефикасно као и коришћење директних показивача или макроа; резултујући машински код је идентичан, па нема казне у погледу перформанси [20] [21]. Дакле, главна разлика је у побољшаној прегледности и типској безбедности кода, без жртвовања ефикасности.

Предности и значај апстракције регистра. Стандарди попут CMSIS-а и званични *header*-и произвођача обезбеђују да програмери не морају ручно да дефинишу сваки регистар и адресу – већ су им на располагању унапред проверене дефиниције структуре и базних адреса [22] [23]. Ово смањује

могућност грешке и унапређује преносивост софтвера између различитих платформи. Код написан уз коришћење симболичких регистара (нпр. `RCC->AHB1ENR` или `GPIOD->ODR` у случају STM32 микроконтролера [24] [25]) много је разумљивији него код са *„магичним“* бројевима адреса, што доприноси бољој одрживости. У академском и индустријском контексту, овакав ниво апстракције се препоручује као део добрих пракси пројектовања: повећава се кохезија и јасно раздвајање надлежности софтверских модула, чинећи систем лакшим за верификацију и одржавање [26]. На крају, приступ меморијски мапираним регистрима представља основни механизам којим *bare-metal* фирмвер остварује интеракцију са физичким светом – кроз промишљено коришћење овог механизма, постиже се детерминистичко, брзо и предвидиво извршавање управљачког кода, што је од пресудне важности за реалновременске примене.

---

[1] [3] lect14
https://pages.hmc.edu/harris/class/e85/old/fall19/lect14.pdf

[2] [4] Memory-mapped I/O and port-mapped I/O - Wikipedia
https://en.wikipedia.org/wiki/Memory-mapped_I/O_and_port-mapped_I/O

[5] [6] [7] [8] What is the memory layout of the ARM Cortex-M3? - SoC
https://s-o-c.org/what-is-the-memory-layout-of-the-arm-cortex-m3/

[9] [10] System-on-Chip Design with Arm® Cortex®-M processors - 1
https://velog.io/@jh708009/System-on-Chip-Design-with-Arm-Cortex-M-processors-1

[11] [19] Од изворног C кода до извршне бинарне слике - компилација и распоред у меморији микроконтролера.pdf
file://file-XSXgxjYbDPF9DijCnBmF9b

[12] [13] Memory mapped registers in C/C++ - OSDev Wiki
https://wiki.osdev.org/Memory_mapped_registers_in_C/C++

[14] [15] [16] [17] [18] [20] [21] [22] [23] [24] [25] [26] Peripheral register access using C Struct's - part 1 - Sticky Bits - Powered by FeabhasSticky Bits – Powered by Feabhas
https://blog.feabhas.com/2019/01/peripheral-register-access-using-c-structs-part-1/