

Дешифрация на адресното поле



Автор: гл. ас. д-р инж. Любомир Богданов



Европейски съюз

ПРОЕКТ BG051PO001--4.3.04-0042

***„Организационна и технологична инфраструктура за учене през
целия живот и развитие на компетенции”***

Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси”,
съфинансирана от Европейския социален фонд на Европейския съюз

Инвестира във вашето бъдеще!



Европейски социален фонд

Съдържание

1. Обща дешифрация
2. Местна дешифрация
3. Смесена дешифрация
4. Пълна и непълна дешифрация
5. Симетрична и несиметрична дешифрация
6. Преместване на региони от паметта
7. Съпоставяне на C структури към адреси
8. Контрол на отделни битове (bit banding)

Обща дешифрация

В презентация №1 беше посочено, че в една микропроцесорна система се използват най-малко три вида магистрали за комуникация между μ PU и периферията.

Това са: **даннова, адресна и управляваща** магистрала.

По **адресната магистрала** се задава адреса на **периферията** или **паметта**, с която искаме да комуникираме.

Обща дешифрация

В презентация №2 беше посочено, че от разредността на адресната магистрала на μPU се формира т.нар. **адресно поле**. То може да бъде изобразено графично чрез **карта на паметта**.

Картата на паметта представлява правоъгълник, разделен на области, отбелязани с начален и краен адрес.

Примерна карта на паметта на една микропроцесорна система е дадена на следващия слайд.

Обща дешифрация

Вектори на прекъсване	0xFFFF FFFF
Програмна памет (ROM)	0xFFFF FF00
Неизползван регион	0xFFFFB FF01
АЦП модул	0x0010 002E
I2C модул	0x0010 001F 0x0010 0019
UART модул	0x0010 0010 0x0010 000F
SPI модул	0x0010 000A 0x0010 0009
Неизползван регион	0x0010 0000
EEPROM	0x0004 0400
Неизползван регион	0x0004 0000
Даннова памет (RAM)	0x0000 3FFF
	0x0000 0000

Обща дешифрация

В този пример адресното поле се простира от начален адрес 0x0000.0000 до краен адрес 0xFFFF.FFFF. Това означава, че могат да се адресират $4\ 294\ 967\ 296_{(10)}$ регистъра на периферни модули и памет. В практиката толкова много регистри не са нужни и затова съществуват неизползвани области от картата на паметта.

За да може да се осъществи правилна комуникация между μ PU и всеки един периферен модул, **трябва отделните модули да са разположени на различни адреси**. Когато μ PU зададе адрес трябва само един единствен модул да отговори на заявката за комуникация. В противен случай, ще се получи объркване на сигналите. На следващия слайд е показано директно свързване на адресните магистрали на μ PU и периферните модули/паметта.

Обща дешифрация

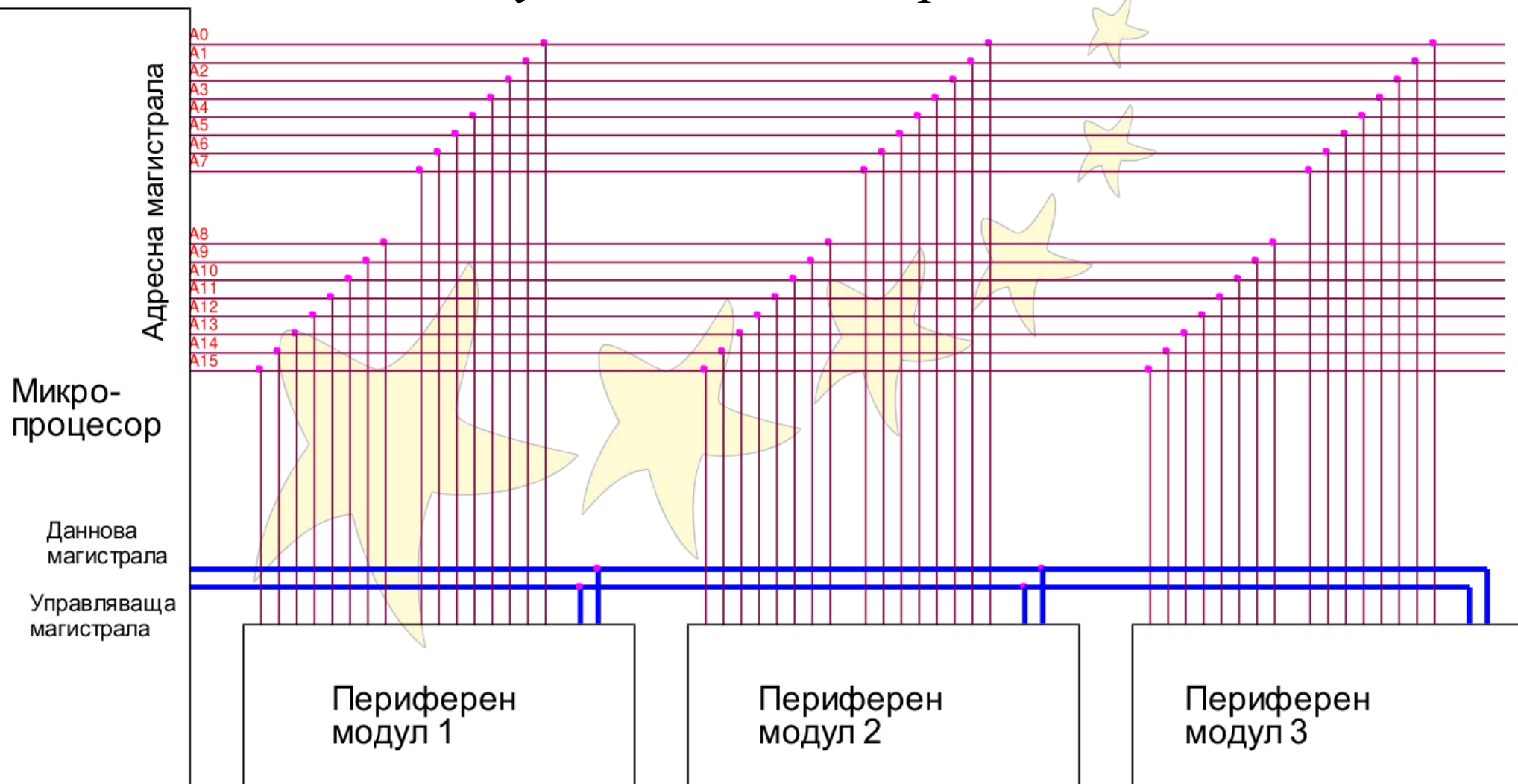
На схемата на следващия слайд е показано 16-битово адресиране. Всеки един от модулите притежава 16 адресни входа. Това означава, че всеки дешифратор ще има **65536** изхода.

Такъв вид свързване е **излишно**, защото **всеки модул има група от няколко регистъра**, заемащи няколко адреса.

Очевидно сложността на този вид адресиране се увеличава с увеличаване разредността на адресите. Например още по-сложна схема би се получила при 64-битово адресиране (т.е. адресната магистрала щеше да има 64 отделни адресни сигнала, а всеки дешифратор от периферията - 2^{64} изхода).

Обща дешифрация

Ненужно сложно свързване!

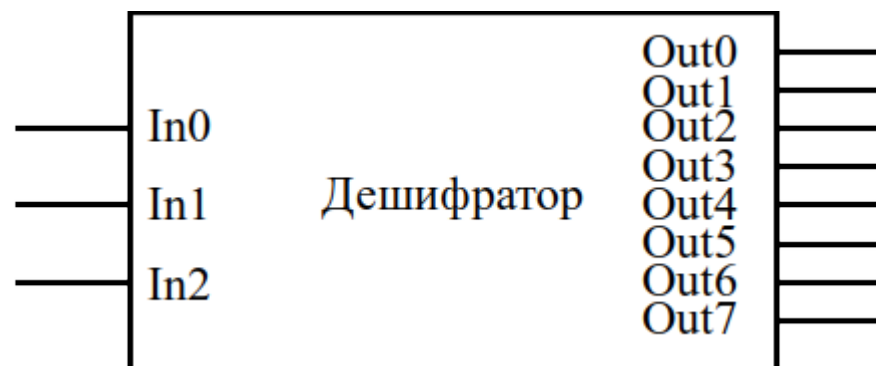


Обща дешифрация

От курса “Цифрова схемотехника” знаем, че **дешифратор** (или още – декодер) е комбинационна логическа схема, която активира само един от изходите си в съответствие с двоичното число, подадено на входа си [1].

На следващия слайд е демонстрирана работата на дешифратора.

Обща дешифрация

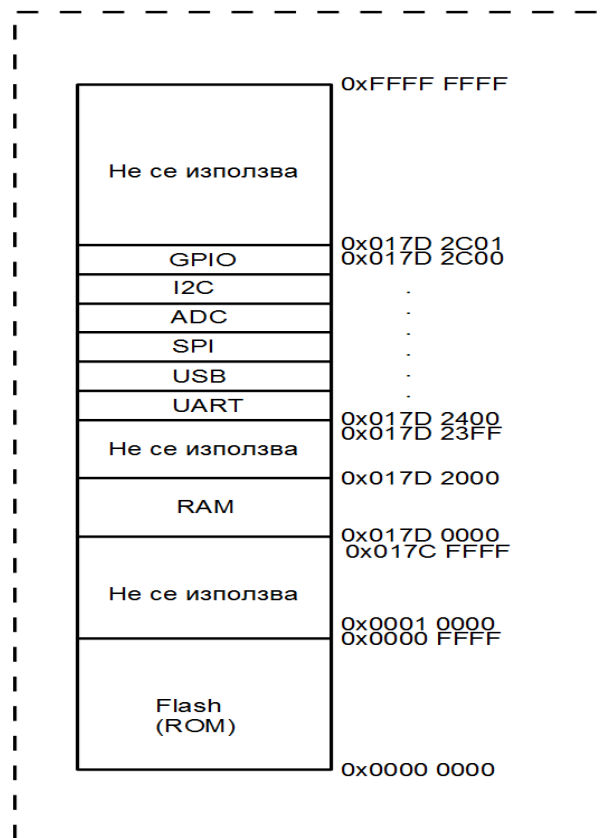


In ₍₁₀₎	In2	In1	In0		Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0		0	0	0	0	0	0	0	1
1	0	0	1		0	0	0	0	0	0	1	0
2	0	1	0		0	0	0	0	0	1	0	0
3	0	1	1		0	0	0	0	1	0	0	0
4	1	0	0		0	0	0	1	0	0	0	0
5	1	0	1		0	0	1	0	0	0	0	0
6	1	1	0		0	1	0	0	0	0	0	0
7	1	1	1		1	0	0	0	0	0	0	0

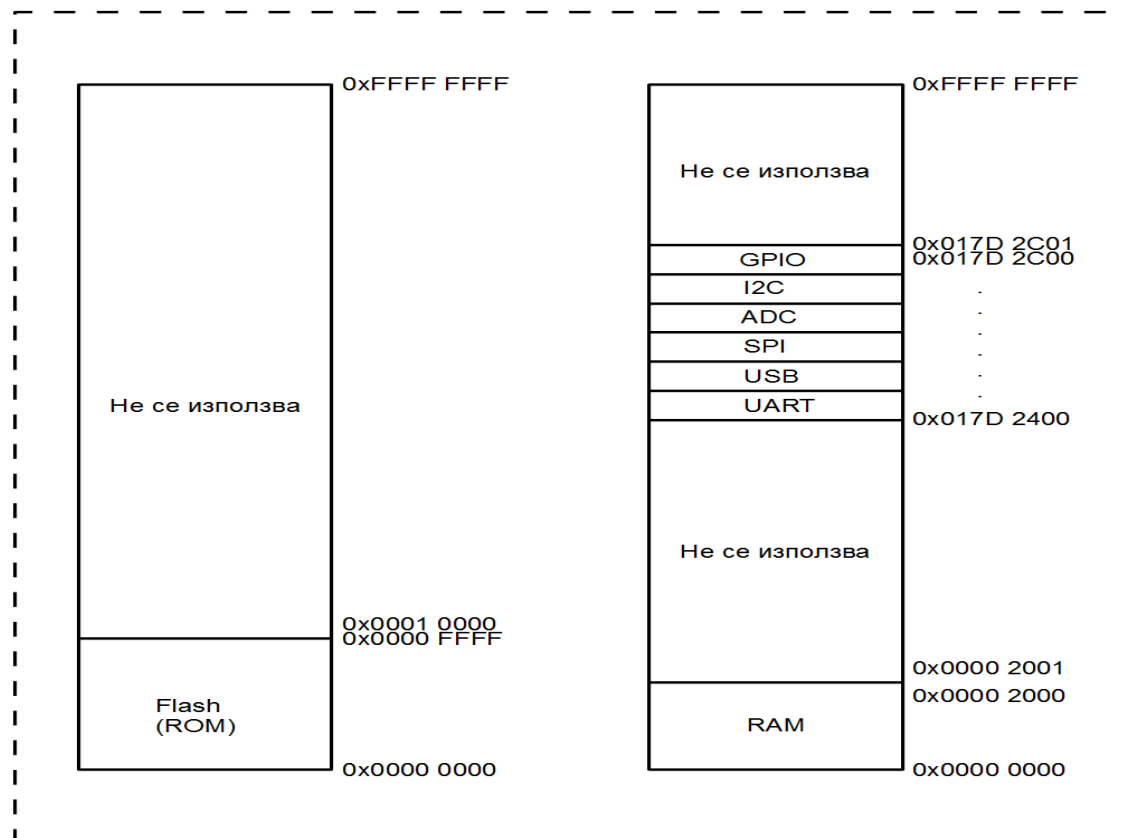
Обща дешифрация

В презентация №2 беше посочено, че Фон Ноймановата архитектура притежава едно адресно поле, а Харвард – две. Те са показани на фигурата по-долу. В практиката обаче се използват и Харвард архитектури с повече от две адресни полета. Този пример е демонстриран на следващия слайд.

Фон Нойман

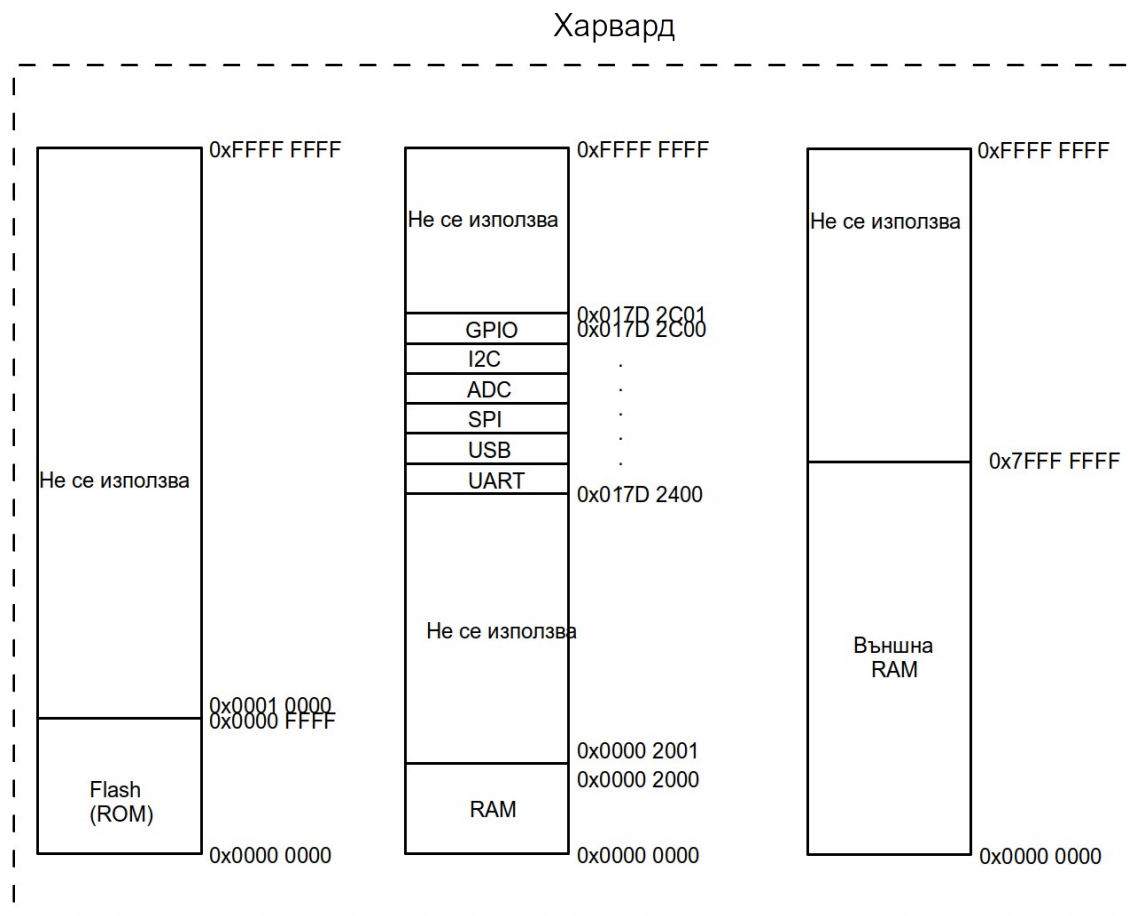
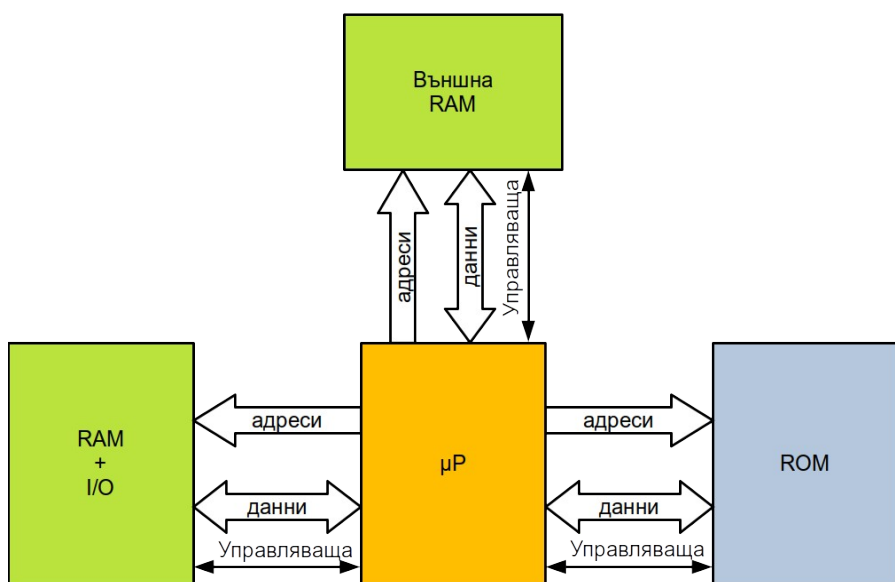


Харвард



Обща дешифрация

Харвард архитектура с 3 адресни полета. Едното се използва за адресиране на външна RAM памет.



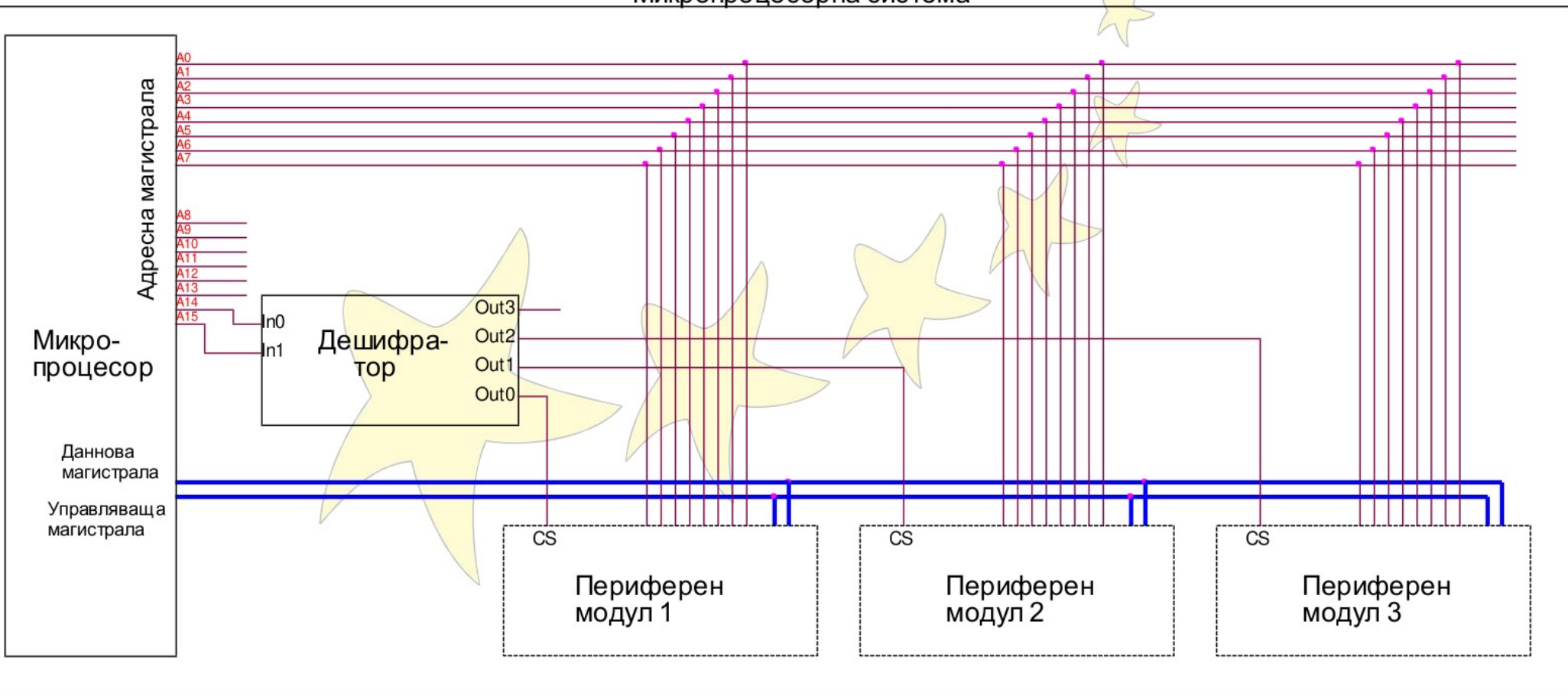
Обща дешифрация

Обща дешифрация - разделяне на адресното поле на области чрез дешифратор, свързан към няколко бита от старшата част на адресната магистрала в микропроцесорната система.

Този метод е показан на следващия слайд.

Обща дешифрация

Микропроцесорна система



Обща дешифрация

Пример - на схемата се използва 4-изходен дешифратор, с помощта на който паметта е разделена на 4 области от по 256 адреса. Една от областите не се използва, защото има само 3 периферни модула. Ако μPU иска да адресира регистър 127 от модул 2, той трябва да формира адреса: 0x407F, защото младшата част указва адреса на регистъра $7F_{(16)} = 127_{(10)}$, а бит 14 и 15 от старшата част контролират дешифратора.

Ако искаме дешифратора да активира първия си изход (свързан към модул 2), трябва на входа му да се подаде $\text{In}0 = 1$, $\text{In}1 = 0$, което отговаря на $A14 = 1$, $A15 = 0$. Битове $A8 - A13$ са без значение. Приемаме, че са нули. Тогава старшата част на адреса става $0100.0000_{(2)} = 40_{(16)}$. След обединение на старшата и младшата част се получава числото 0x407F.

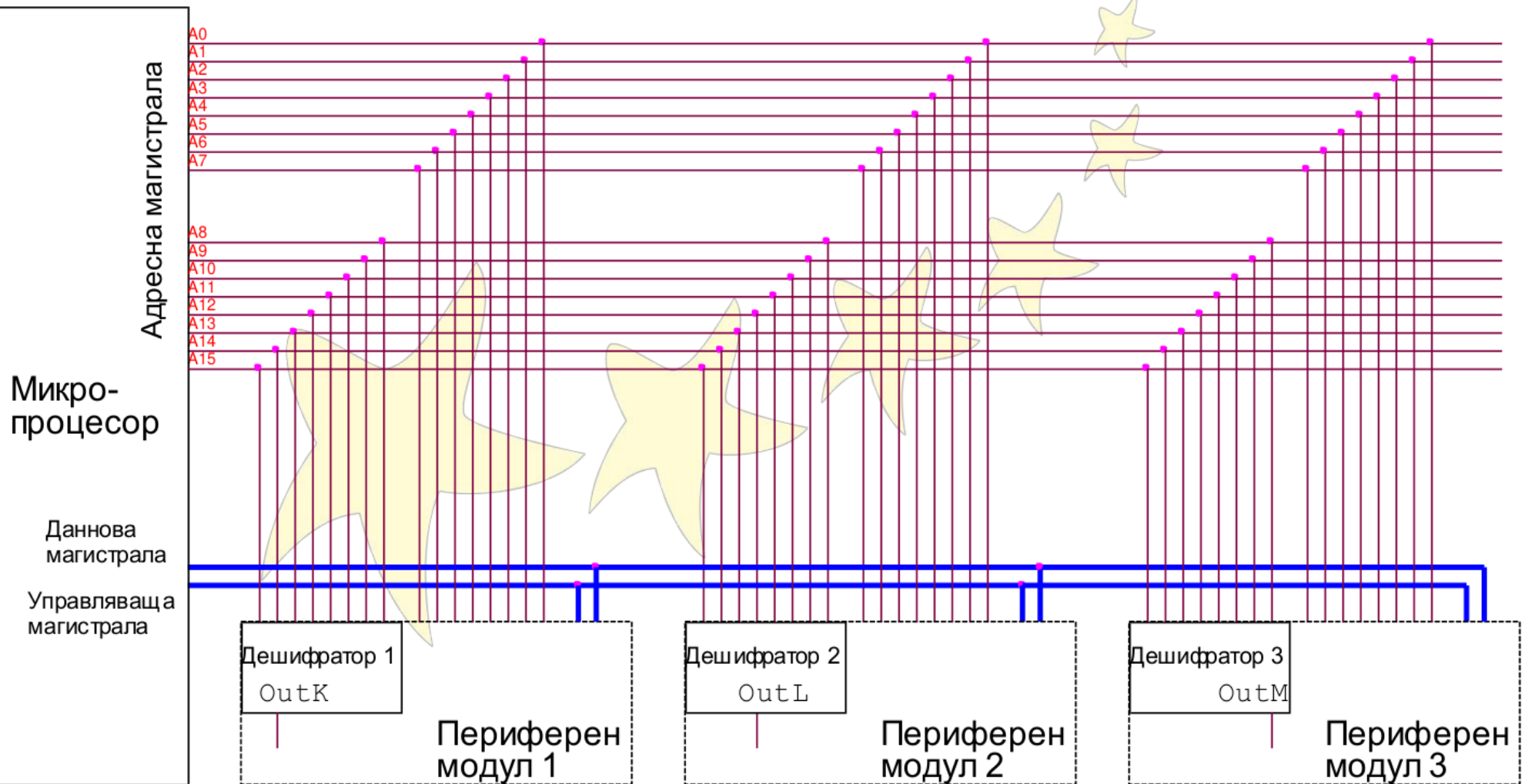
Местна дешифрация

Местна дешифрация - представлява разделяне на адресното поле на области чрез дешифратори, свързани към адресната магистрала на микропроцесорната система и вградени в самите периферни модули. Този метод е показан на следващия слайд.

Да се обърне внимание на изходите на дешифраторите – CS сигнаът се взима от различни изходи, за да не се получи конфликт (дублиране) между модулите.

Местна дешифрация

Микропроцесорна система



Смесена дешифрация

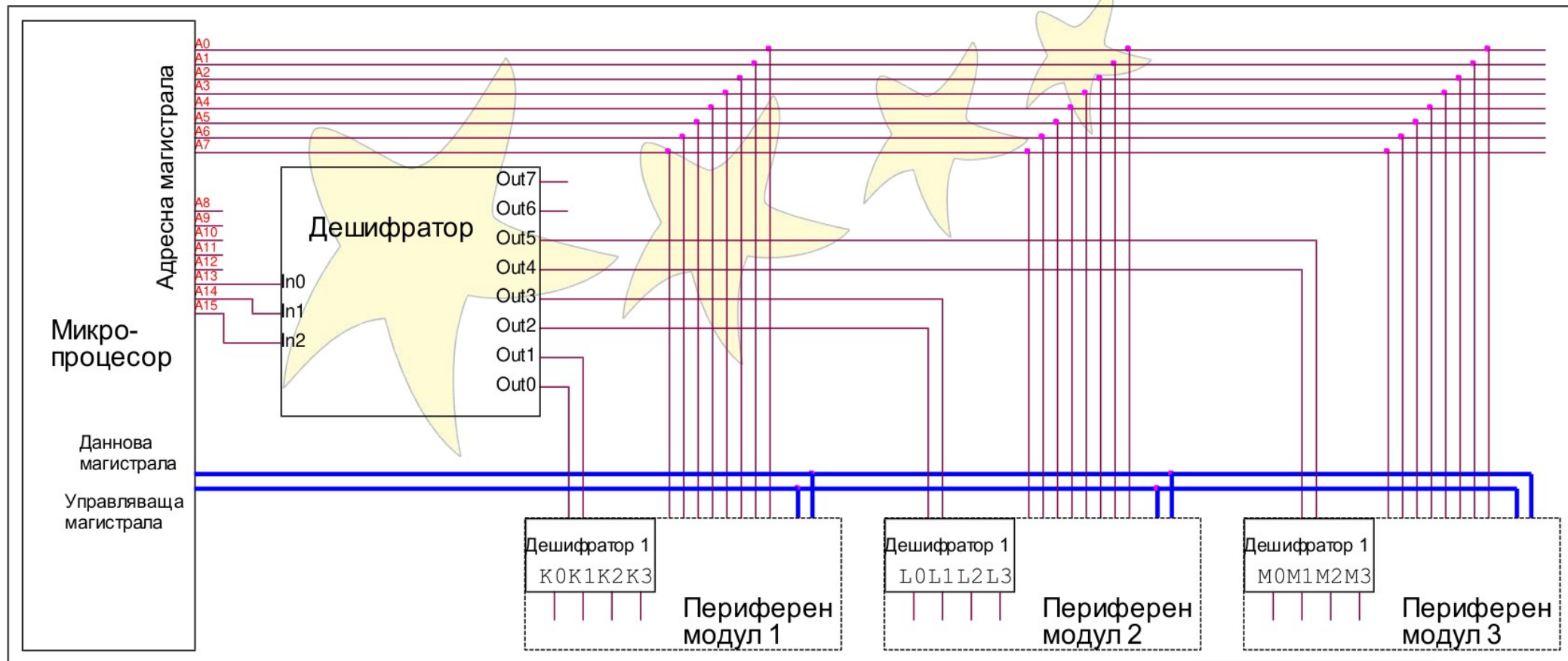
Смесена дешифрация - представлява комбинация от обща и местна дешифрация - разделя се адресното поле на области чрез дешифратори, глобално в микропроцесорната система и локално при периферните модули.

Този метод е показан на фигурата на следващия слайд.

Смесена дешифрация

долу.

Микропроцесорна система



Пълна и непълна дешифрация

Пълно дешифриране — това е дешифриране, при което част от адресните сигнали отиват към периферен модул/памет, а останалите към дешифратор.

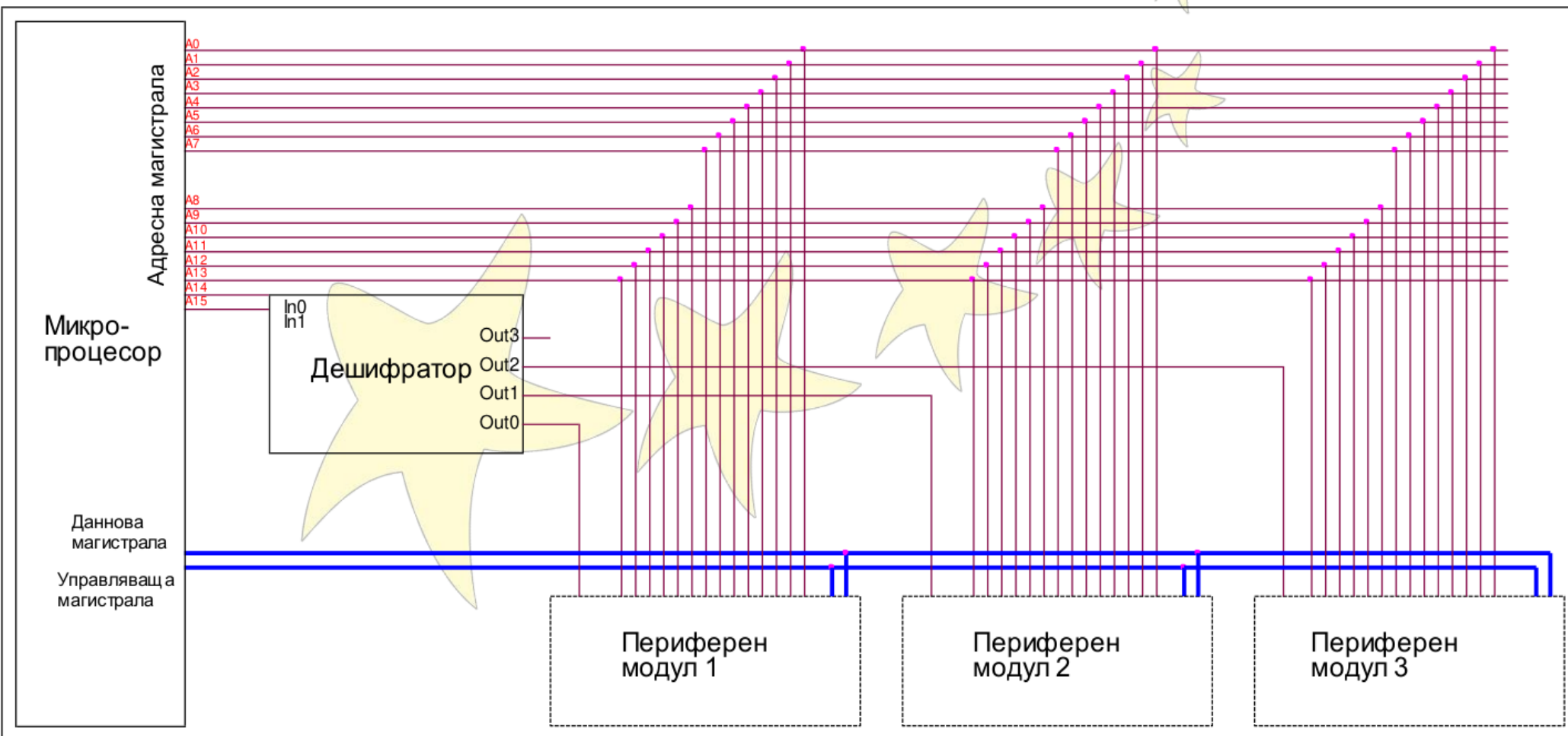
Методът е показан на следващия слайд.

Непълно дешифриране — това е дешифриране, при което част от адресните сигнали отиват към периферен модул/памет, друга част към дешифратор, а известен брой от тях са несвързани.


Методът е показан на по-следващия слайд.

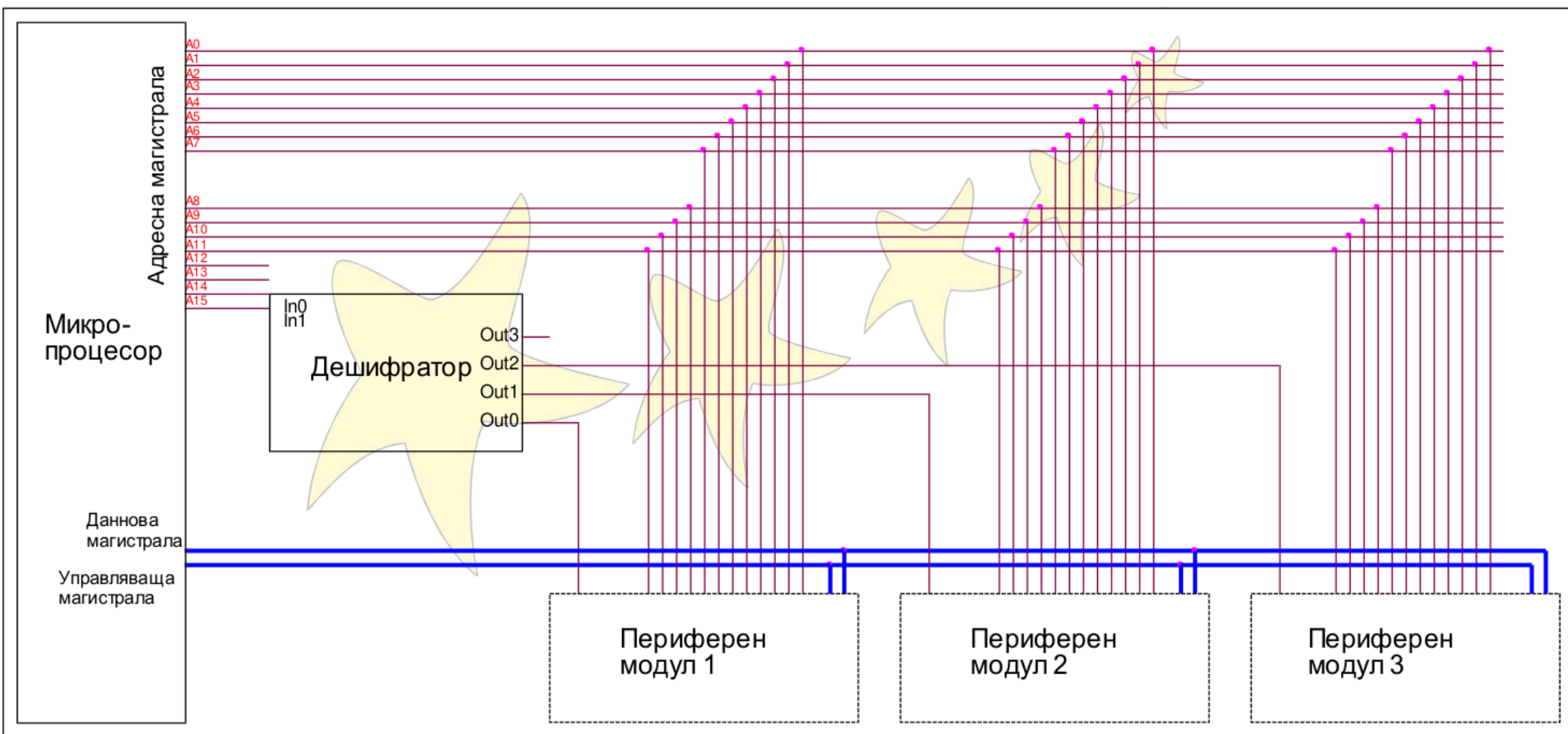
Пълна и непълна дешифрация

Микропроцесорна система (пълно дешифриране)



Пълна и непълна дешифрация

Микропроцесорна система (непълно дешифриране) 



Пълна и непълна дешифрация

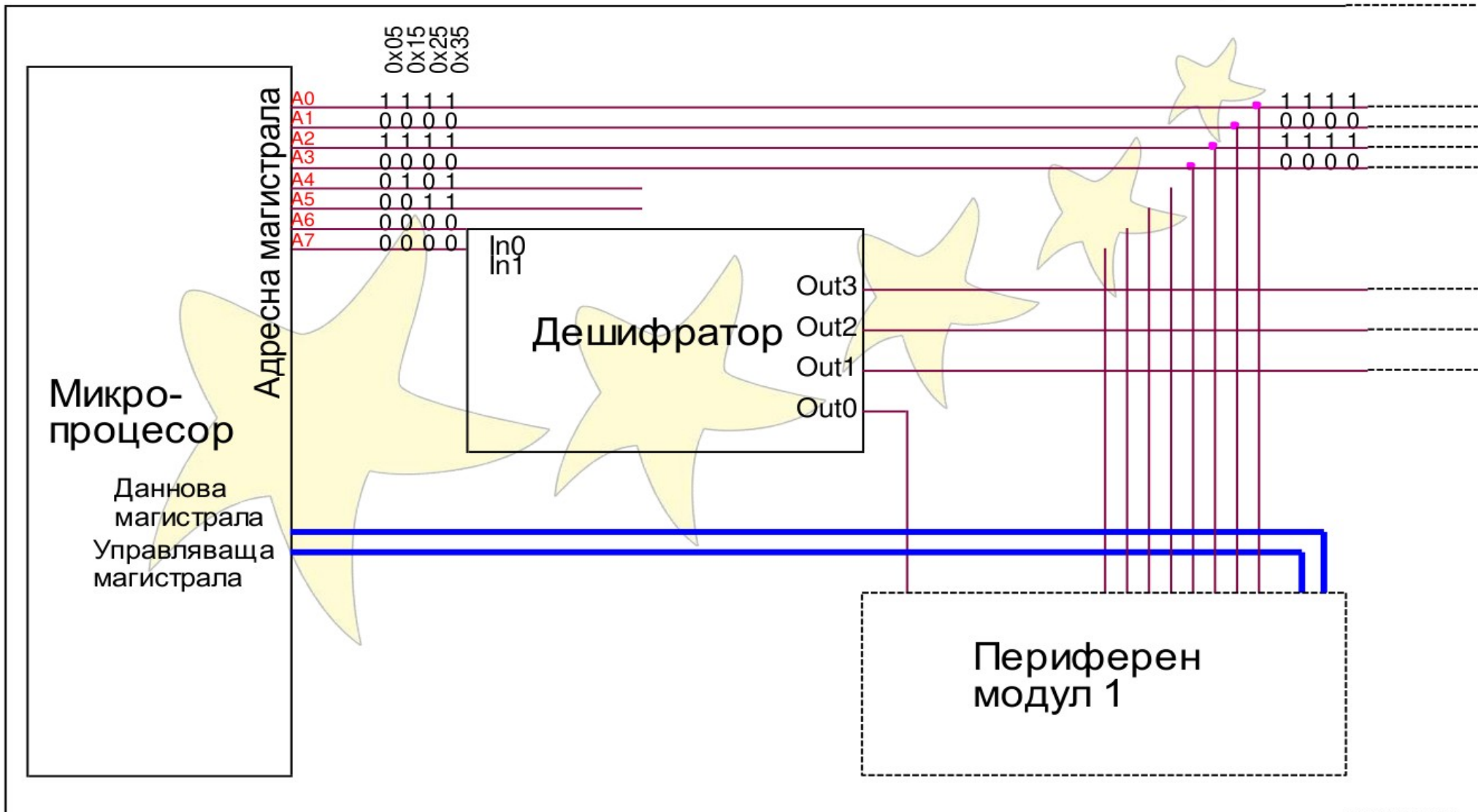
Важно е да се обърне внимание, че при непълното дешифриране, поради несвързаните сигнали, в адресното поле на някой от периферните модули ще се получи дублиране, т.е. един и същ регистър ще е достъпен на повече от един адрес.

Пример - 8-битово адресиране и неизползвани адресни изходи е илюстрирано на следващия слайд.

Вижда се, че регистър на Периферен модул 1 ще бъде видим както от адрес 0x05, така и от адреси 0x15, 0x25, 0x35. Адресните битове A4 и A5 не са значещи и каквато и стойност да приемат, винаги ще се адресира регистърът от адрес 0x05.

Пълна и непълна дешифрация

Микропроцесорна система (непълно дешифриране)



Симетрична и несиметрична дешифрация

Симетрично дешифриране – дешифриране, при което старшите няколко бита на адресното поле са свързани към входовете на **един дешифратор**. Към изходите му са свързани входовете за разрешаване (Enable) на отделните периферни модули и адресното поле е **разделено да равни области**.

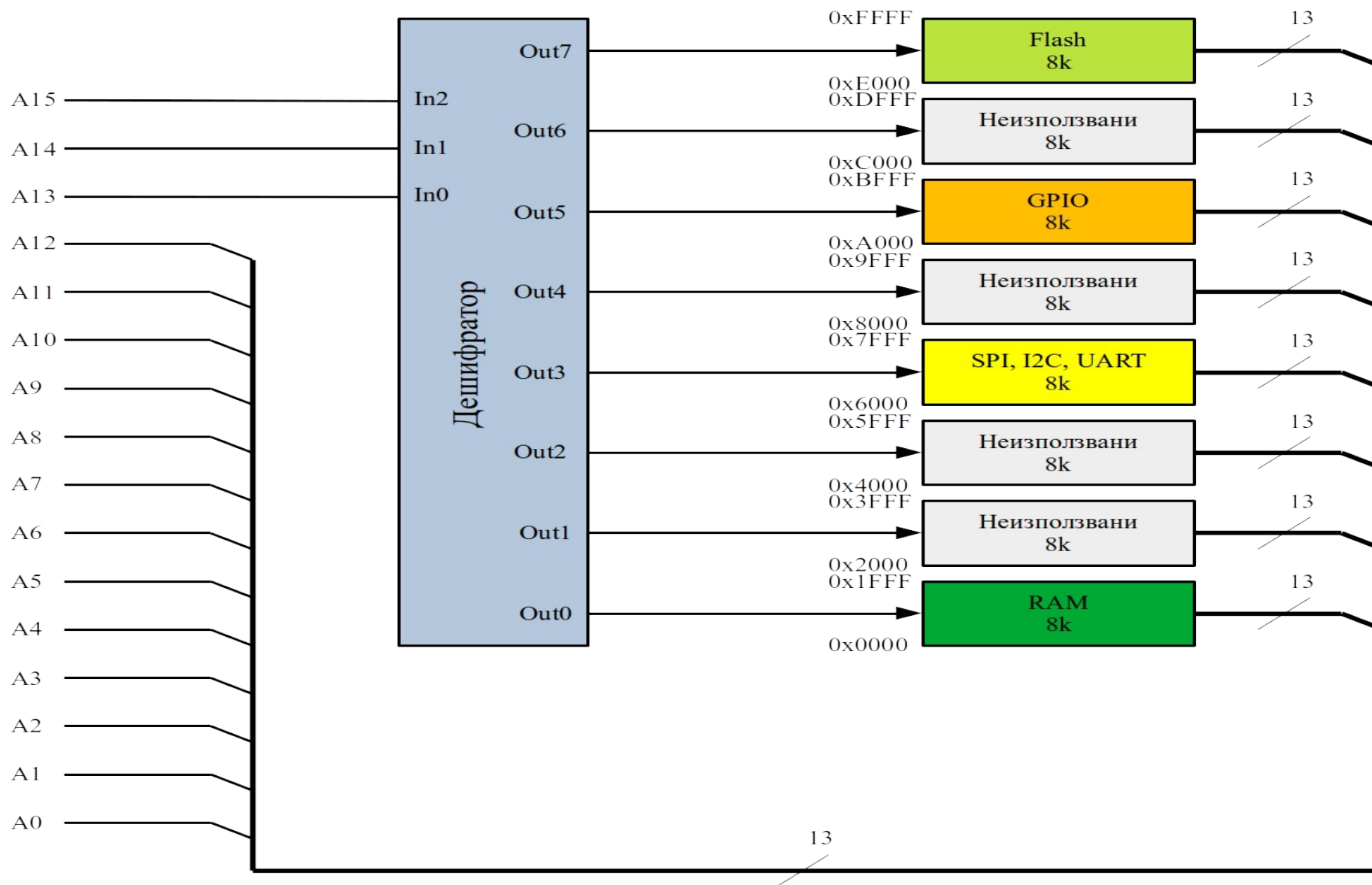
Симетрична и несиметрична деш.

Пример - на следващия слайд е показано симетрично дешифриране, при което старшите три сигнала от адресната шина са свързани към входовете на тривходов дешифратор. Останалите 13 линии се свързват към периферните модули и адресират регистри от тях (до $2^{13} = 8192$ регистъра). Изходите на дешифратора служат като сигнал за избор на модул (CS – Chip Select).

По този начин може да се каже, че адресното поле е разделено на 8 равни области от по 8 kB всяка. След последния адрес следва първия адрес на следващата област. Например след последния адрес на RAM паметта $0x1FFF$ следва начален адрес на незаета област $0x1FFF + 1 = 0x2000$.

Симетрична и несиметрична дешифрация

Симетрично дешифриране



Симетрична и несиметрична дешифрация

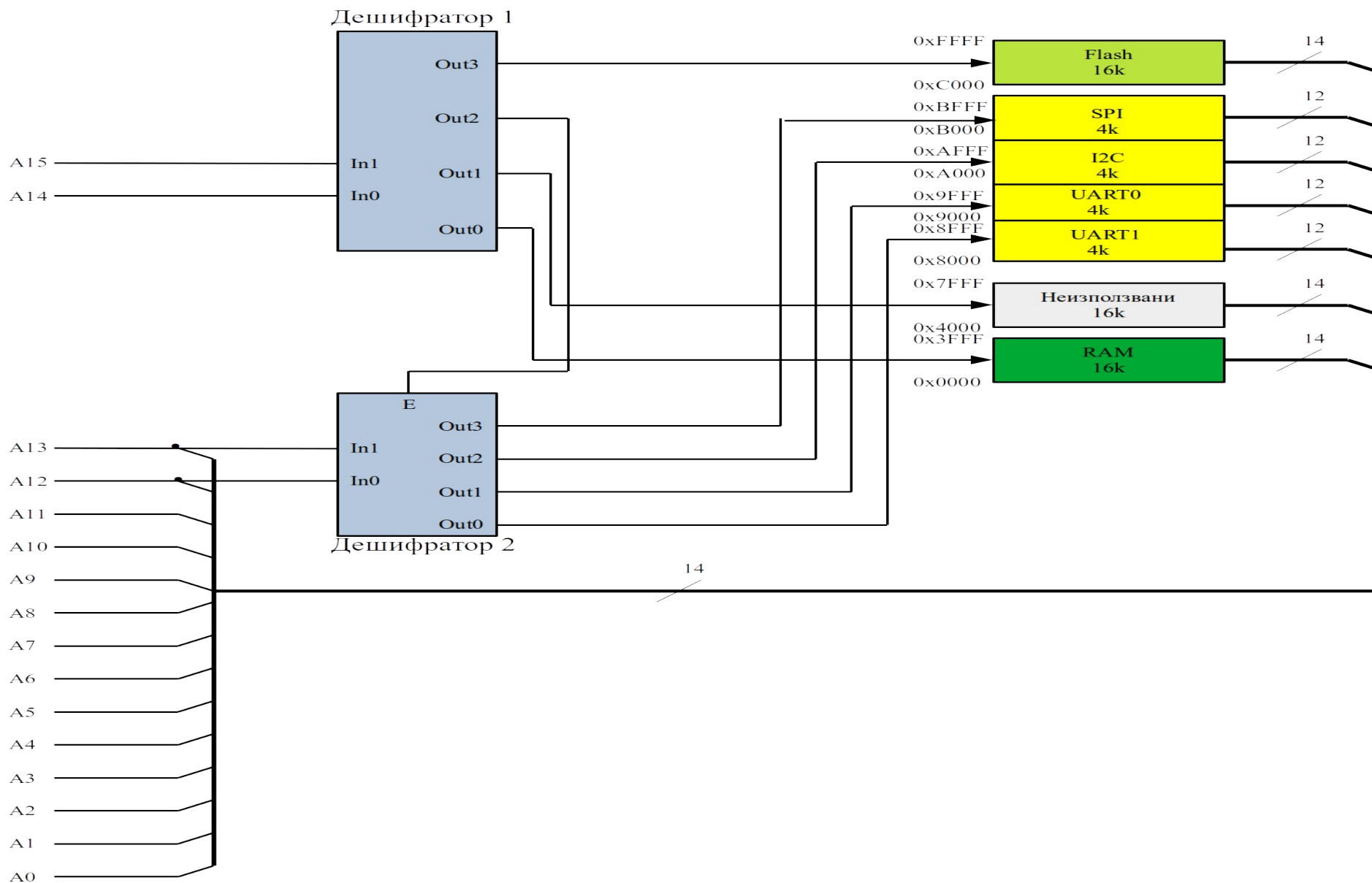
Несиметрично дешифриране - дешифриране, при което старшите няколко бита на адресната магистрала са свързани към входовете на **два или повече дешифратора**. Дешифраторите трябва да са с вход за разрешаване **E (Enable)**. Когато изходът на един дешифратор се свърже към входа за разрешаване на друг дешифратор се получава така, че дадени региони от адресното поле се разделят на подрегиони. Затова дешифрирането се казва несиметрично, защото адресното поле **съдържа различни по големина области**.

Симетрична и несиметрична дешифрация

Пример - на следващия слайд е показано 16-битово несиметрично дешифриране. Използват се два двувходови дешифратора, като вторият е разрешаван от първия с извода си Е. Адресното поле е разделено на 16- и 4-килобайтови области, защото свободните адресни линии са съответно 14 и 12 на брой.

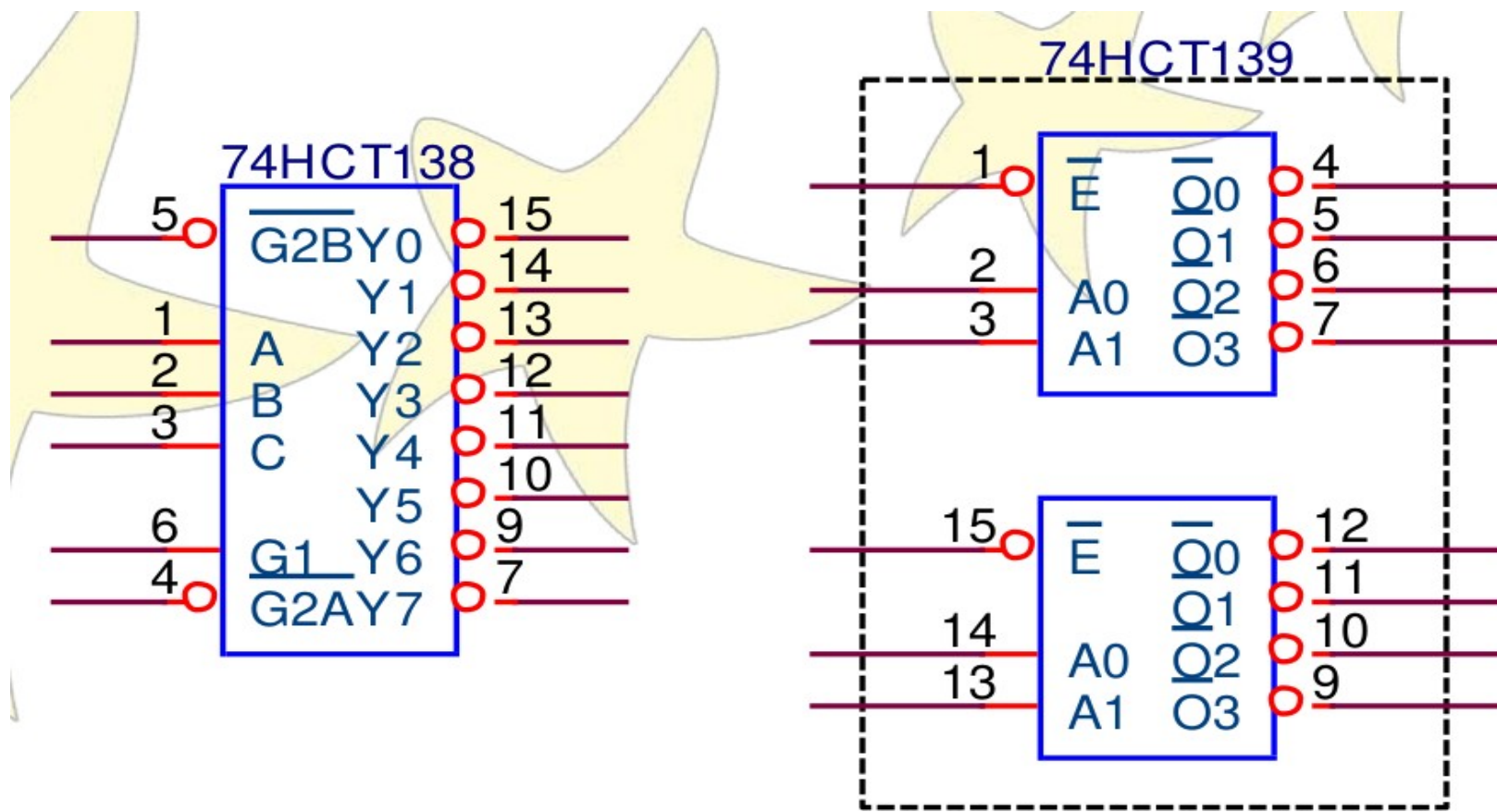
Симетрична и несиметрична дешифрация

Несиметрично дешифриране



Симетрична и несиметрична дешифрация

В практиката за дешифриране на адресното поле може да се използват интегралните схеми 74xx138 и 74xx139. Схемата '138 е един тривходов, а '139 са два двувходови дешифратора.



Преместване на региони от паметта

От показаните схеми е видно, че конфигурацията на дешифраторите и адресната шина е фиксирана, а оттам следва, че и адресното поле не може да се променя. В практиката обаче понякога се налага адресът на даден модул да бъде променен [2]. Например, ако се свърже външен модул, който е с фиксиран адрес, съвпадащ с адреса на някой от вътрешните модули. Затова е добре дешифрицията да се имплементира по начин, позволяващ промени в адресното поле.

Това най-често става със:

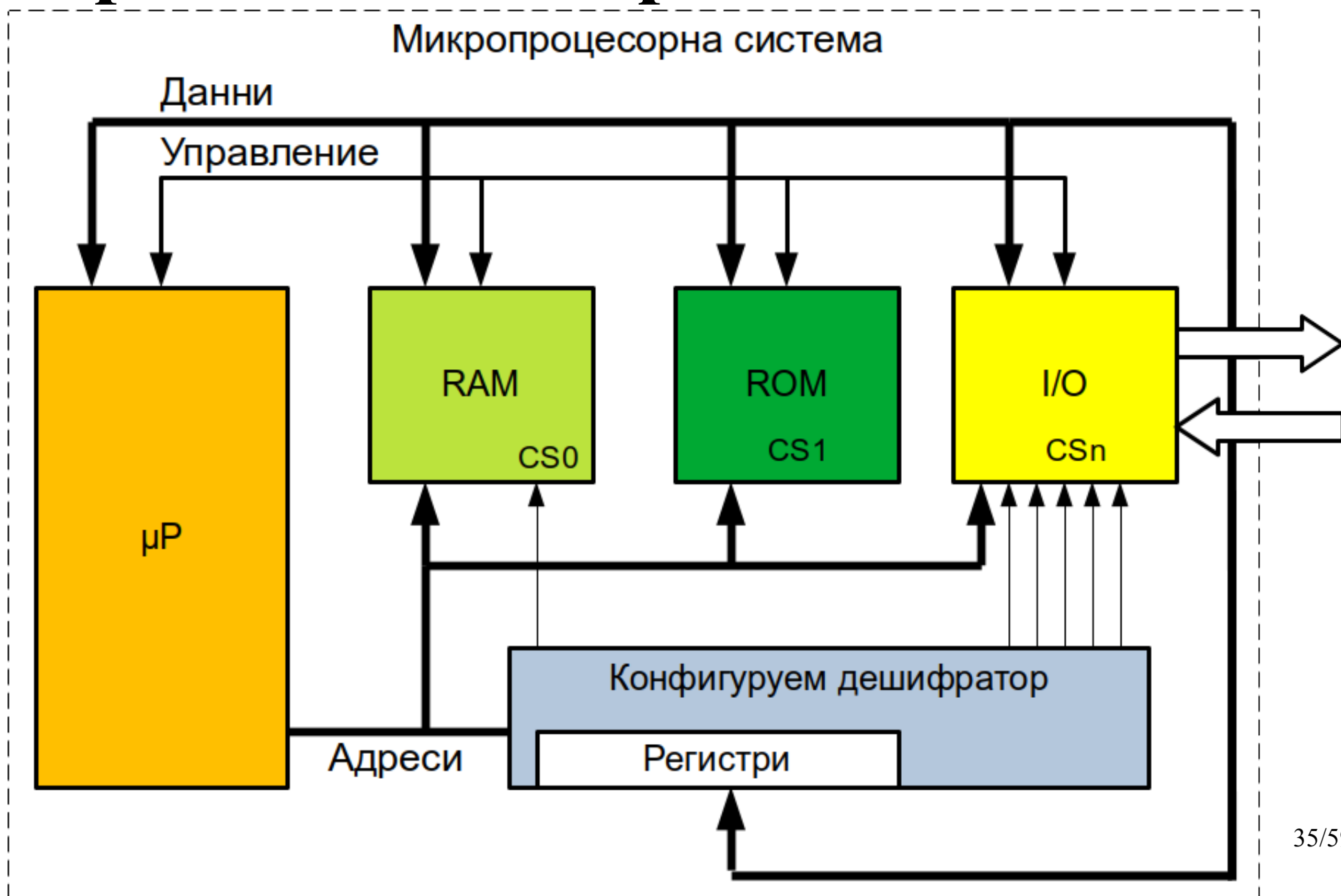
- *дешифратор с цифрови компаратори
- *памет (ROM, PROM, EPROM, EEPROM, Flash и др.)
- *програмируема логика (PAL, GAL, CPLD)

Преместване на региони от паметта

***Дешифратор с цифрови компаратори** - специален дешифратор, входовете на който са свързани към изходите на цифрови компаратори. Входовете на компараторите са свързани към адресната магистрала на μ PU. Адресите на μ PU се сравняват с потребителски регистри, които определят обхватите от адреси за всяка памет/периферия.

Потребителските регистри на компаратора се записват от μ PU при стартиране на фърмуера. Това налага постоянната памет да е на фиксиран адрес.

Преместване на региони от паметта

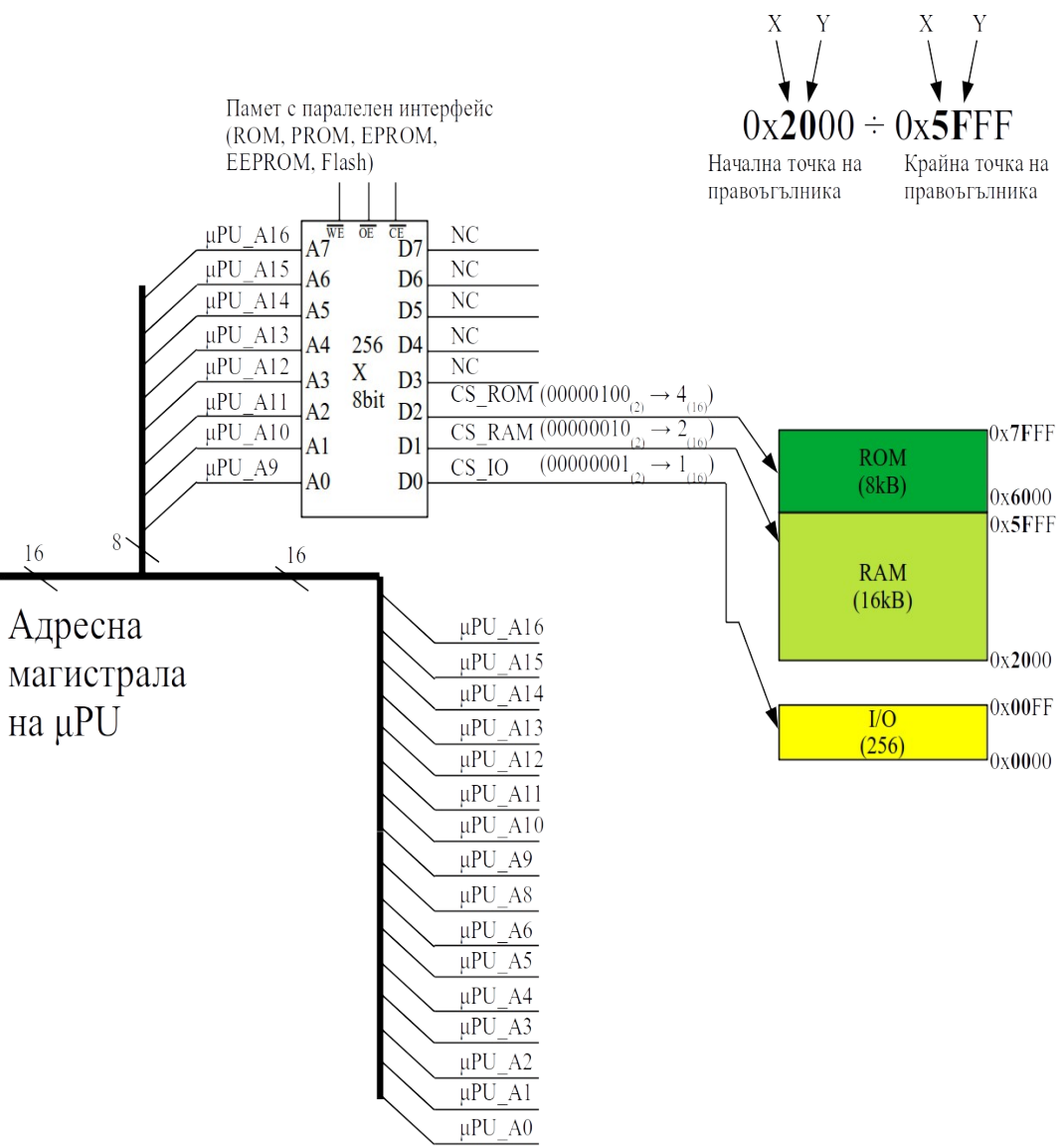


Преместване на региони от паметта

***Дешифриране с памет** — използва се метод с таблица на съответствието (Look-Up Table). Адресните сигнали на паметта се свързват към старшата част на адресната магистрала на μ PU. Данновите сигнали на паметта играят ролята на сигнали за избор на памет/периферия. Числата, които се записват в клетките на паметта са специално подбрани, така че да се активира само един сигнал (т.нар. бягаща нула, или бягаща единица).

Пример — на следващия слайд е даден пример с памет с 256 х 8-битови клетки и несиметрично дешифриране[6].

Преместване на региони от паметта



X Y X Y

$0x2000 \div 0x5FFF$

Начална точка на правоъгълника Крайна точка на правоъгълника

μPU [A8 ÷ A11]
ROM[A0 ÷ A3]

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x0	1															
0x1																
0x2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0x3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0x4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0x5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0x6	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
0x7	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
0x8																
0x9																
0xA																
0xB																
0xC																
0xD																
0xE																
0xF																

μPU [A12 ÷ A15]
ROM [A4 ÷ A7]

Преместване на региони от паметта

***Дешифриране с програмируема логика** — използват се PLD и CPLD схеми за реализация на пълни/непълни дешифратори. Повечето такива чипове имат вграден EEPROM/Flash и могат да се преконфигурират по програмен път.

Преместване на региони от паметта

Пример – на по-следващия слайд е показана част от принципната схема на машина за катодно разпрашване на фирмата Quorum Technologies, модел K550X.

С помощта на машината се нанасят нано слоеве за целите на микроелектрониката. Представлява боросиликатна камера с два електрода, в която се създава вакуум и след това се пуска аргон. На електродите се подават 1000 V, което е достатъчно за газов разряд и материал от единия електрод “полепва” върху всички предмети, поставени на другия електрод.

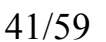
Преместване на региони от паметта

Машината използва PLD интегрална схема, от вида GAL [1], на фирмата Lattice Semiconductor (GAL20V8). Това е базова матрична логика, която се конфигурира от вградената в нея EEPROM памет. Може да бъде препрограмирана.

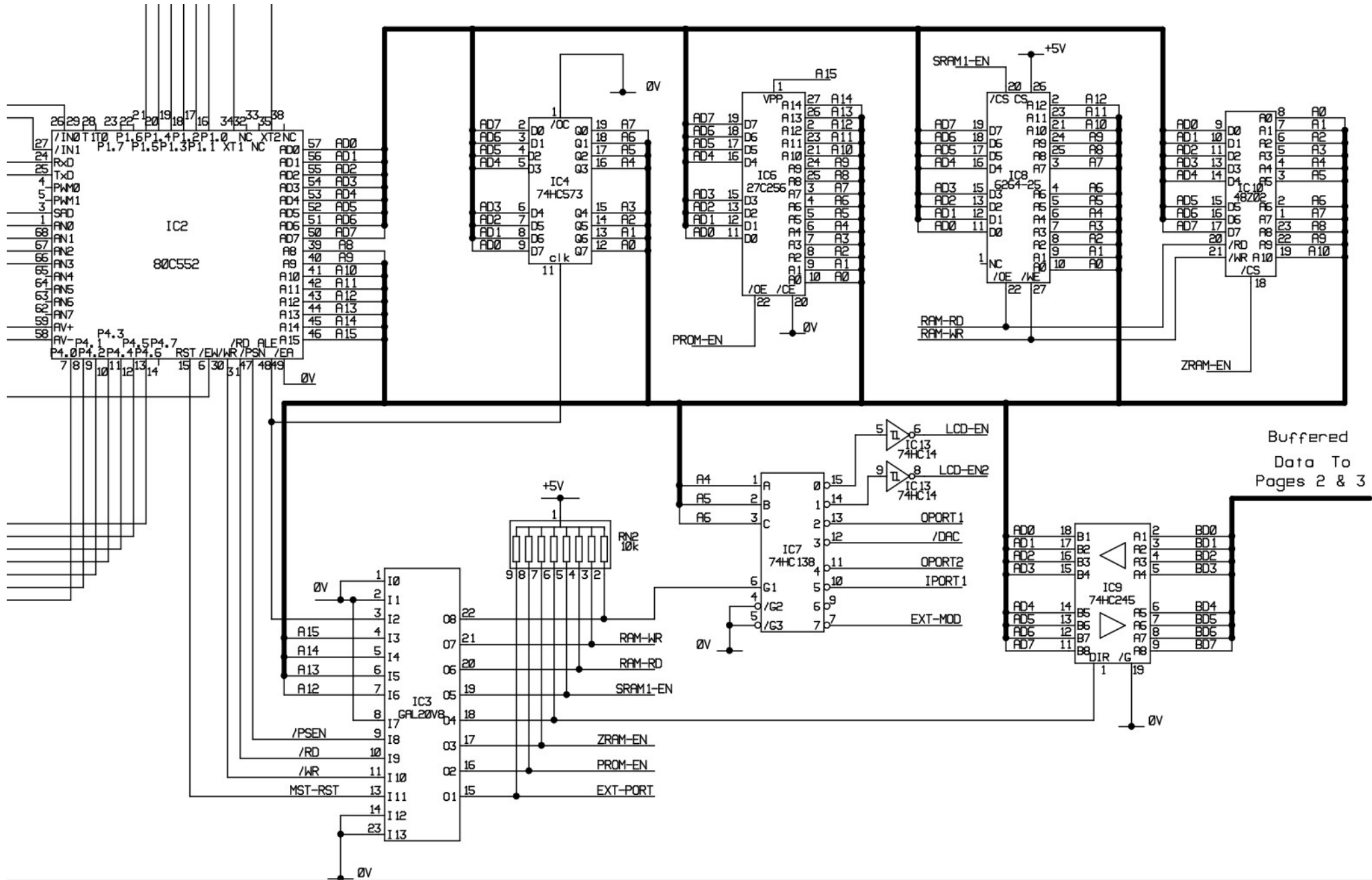
Чрез тази ИС може да се реализира таблица на съответствието (Look-Up Table, LUT) и при дадена комбинация от адреси на входа да се генерират сигнали за разрешаване на други ИС в системата. Методът е аналогичен на дешифратор, който може да бъде програмиран.

Машината използва несиметрично дешифриране с IC3 и IC7.

Lattice Semiconductor, GAL20V8, част от вътрешната структура на схемата.



Преместване на региони от паметта

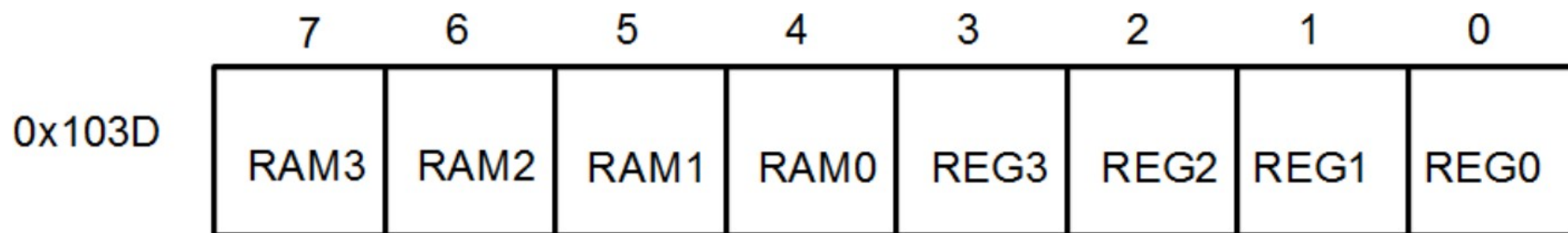


Преместване на региони от паметта

Пример - микроконтролерът на Motorola M68HC11 има конфигурационни регистри в областта 0x1000 ÷ 0x103F при стартиране на системата [3], [4]. Те могат да се преместват динамично в картата на паметта, с помощта на регистъра INIT (0x103D).

Битове REG0 ÷ REG3 представляват старшите 4 бита от 16-битовия адрес на първия регистър от конфигурационния блок.

Ако REG0 ÷ REG3 са 1-1-1-1, то регистрите ще се намират на адреси 0xF000 ÷ 0xF03F, и т.н.



Преместване на региони от паметта

Пример - микроконтролерът на Motorola M68HC11 има SRAM в областта $0x0000 \div 0x00FF$ при стартиране на системата [3], [4]. Те могат да се преместват динамично в картата на паметта, с помощта на регистъра INIT ($0x103D$).

Битове RAM0 \div RAM3 представляват старшите 4 бита от 16-битовия адрес на първия регистър от SRAM паметта.

Ако RAM0 \div RAM3 са 1-0-0-0, то регистрите ще се намират на адреси $0x8000 \div 0x80FF$, и т.н.

Съпоставяне на C структури към адреси

В програмите на C периферните модули се описват със структури.

На всеки един регистър съответства по една променлива от структурата (map).

Компилаторът разполага променливите на C структурите на последователни адреси.

Съпоставяне на C структури към адреси

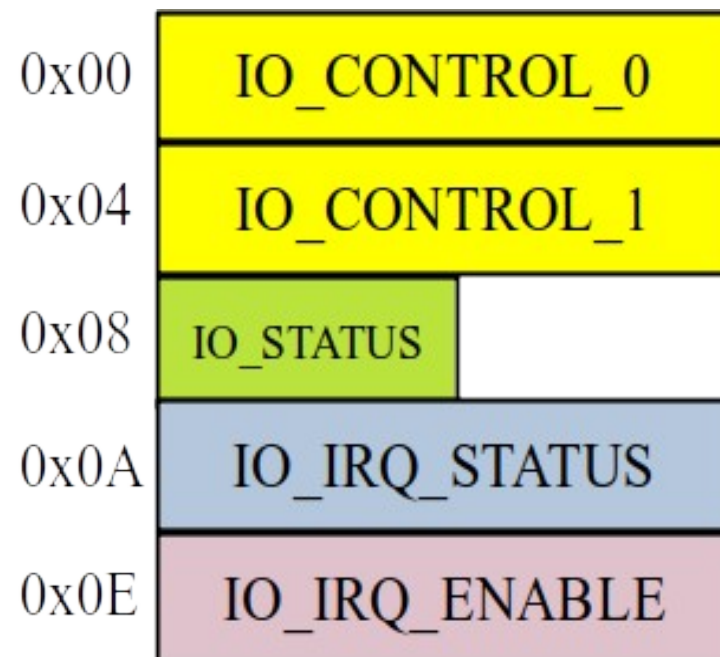
За да се съпостави софтуера към хардуера, трябва да са изпълнени условията:

- *променливите са еднакви по размер с регистрите, към които ще бъдат съпоставени
- *броят на променливите от C структурата е равен на броя на регистрите от хардуерния модул
- *адресът на първата променлива от структурата и първия регистър от хардуерния модул съвпадат
- *разполагането на данните в паметта (endianess) на μ PU и генерираните променливи от компилатора трябва да съвпадат

Съпоставяне на C структури към адреси

```
typedef struct {  
    uint32_t IO_CONTROL[2];  
    uint16_t IO_STATUS;  
    uint32_t IO_IRQ_STATUS;  
    uint32_t IO_IRQ_ENABLE;  
}my_io_module_t;
```

Компилятор



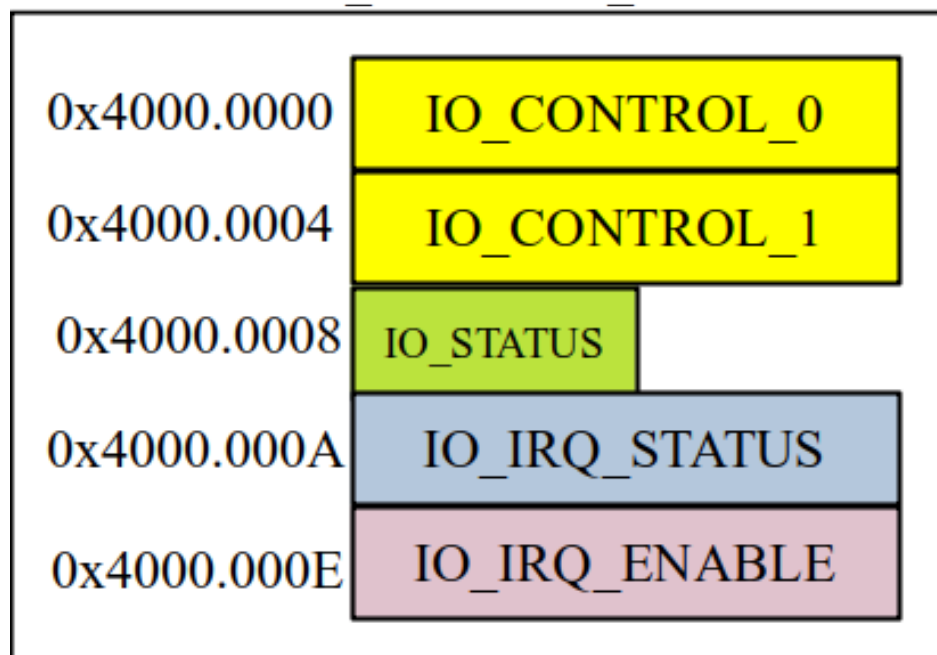
Съпоставяне на C структури към адреси

Съпоставяне (map) на структура към адрес

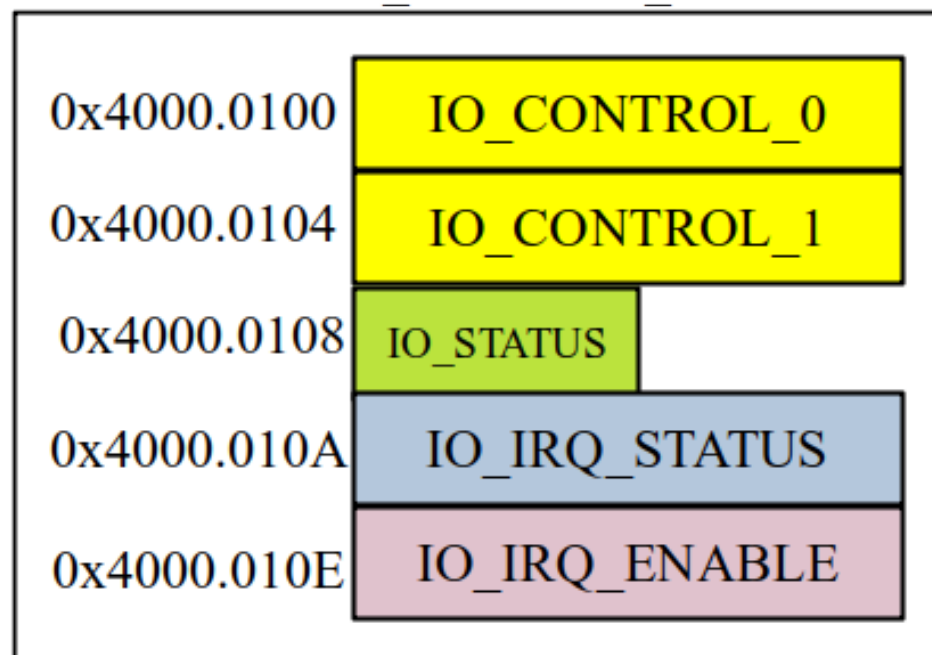
```
#define IO_A ((my_io_module_t *)0x40000000)  
#define IO_B ((my_io_module_t *)0x40000100)
```



IO_MODULE_A



IO_MODULE_B



Съпоставяне на C структури към адреси

След съпоставянето достъпът до регистрите става по следния начин:

//Променяне (запис) на всички битове от контролен
//регистър 0

```
IO_A→IO_CONTROL[0] = 0x4350003F;
```

//Четене на всички битове от статус регистъра

```
uint32_t io_stat;
```

```
io_stat = IO_A→IO_STATUS;
```

Съпоставяне на C структури към адреси

//Вдигане само на един бит (#4) в логическа 1 от
//регистъра за разрешаване на прекъсванията

IO_IRQ_ENABLE = IO_IRQ_ENABLE | 0x10;

//Сваляне само на един бит (#7) в логическа 0 от
//регистъра за разрешаване на прекъсванията

IO_IRQ_ENABLE = IO_IRQ_ENABLE & ~0x80;

//Преобръщане само на един бит (#2) от контролен
//регистър 1

IO_CONTROL[1] = IO_CONTROL[1] ^ 0x04;

Съпоставяне на C структури към адреси

Пример – в библиотеката HAL на STM32L011 се използва съпоставяне на адреси.

В technical reference manual на STM32L011 може да бъде намерен списък с всички регистри на GPIO модулите. Адресите им се дават като отместване спрямо базов адрес, защото регистрите са еднотипни и се използват в няколко GPIO модула, всеки с уникален базов адрес.

Отместване (спрямо базов адрес)	Име на регистър
0x00	GPIOA_MODER
0x04	GPIOx_OTYPER
0x08	GPIOA_OSPEEDR
0x0C	GPIOA_PUPDR
0x10	GPIOx_IDR
0x14	GPIOx_ODR
0x18	GPIOx_BSRR
0x1C	GPIOx_LCKR
0x20	GPIOx_AFRH
0x24	GPIOx_AFRH
0x28	GPIOx_BRR

Съпоставяне на C структури към адреси

В technical reference manual на STM32L011 може да бъде намерена картата на паметта. По-долу е дадена извадка от нея.

Table 3. STM32L0x1 peripheral register boundary addresses⁽¹⁾

Bus	Boundary address	Size (bytes)	Peripheral	Peripheral register map
IOPORT	0X5000 1C00 - 0X5000 1FFF	1K	GPIOH	Section 8.4.12: GPIO register map
	0X5000 1400 - 0X5000 1BFF	2 K	Reserved	-
	0X5000 1000 - 0X5000 13FF	1K	GPIOE	Section 8.4.12: GPIO register map
	0X5000 0C00 - 0X5000 0FFF	1K	GPIO D	Section 8.4.12: GPIO register map
	0X5000 0800 - 0X5000 0BFF	1K	GPIO C	Section 8.4.12: GPIO register map
	0X5000 0400 - 0X5000 07FF	1K	GPIOB	Section 8.4.12: GPIO register map
	0X5000 0000 - 0X5000 03FF	1K	GPIOA	Section 8.4.12: GPIO register map
	0X4002 6400 - 0X4002 FFFF	49 K	Reserved	-
	0X4002 6000 - 0X4002 63FF	1 K	AES (Cat. 1, 2 and 5 with AES only)	Section 15.7.13: AES register map
	0X4002 5400 - 0X4002 5FFF	3 K	Reserved	-
	0X4002 4400 - 0X4002 53FF	3 K	Reserved	-

Съпоставяне на С структури към адреси

stm32l011xx.h

```
=====
#define PERIPH_BASE      ((uint32_t)0x40000000U)
#define IOPPERIPH_BASE   (PERIPH_BASE + 0x10000000U)
```

```
-----
#define GPIOA_BASE       (IOPPERIPH_BASE + 0x00000000U)
#define GPIOB_BASE       (IOPPERIPH_BASE + 0x00000400U)
#define GPIOC_BASE       (IOPPERIPH_BASE + 0x00000800U)
```

```
-----
#define GPIOA             ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB             ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC             ((GPIO_TypeDef *) GPIOC_BASE)
```

```
-----
typedef struct{
  __IO uint32_t MODER;      /*!< GPIO port mode register,           Address offset: 0x00 */
  __IO uint32_t OTYPER;     /*!< GPIO port output type register,      Address offset: 0x04 */
  __IO uint32_t OSPEEDR;    /*!< GPIO port output speed register,     Address offset: 0x08 */
  __IO uint32_t PUPDR;      /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
  __IO uint32_t IDR;        /*!< GPIO port input data register,       Address offset: 0x10 */
  __IO uint32_t ODR;        /*!< GPIO port output data register,      Address offset: 0x14 */
  __IO uint32_t BSRR;       /*!< GPIO port bit set/reset registerBSRR, Address offset: 0x18 */
  __IO uint32_t LCKR;       /*!< GPIO port configuration lock register, Address offset: 0x1C */
  __IO uint32_t AFR[2];     /*!< GPIO alternate function register,    Address offset: 0x20-0x24 */
  __IO uint32_t BRR;       /*!< GPIO bit reset register,            Address offset: 0x28 */
}GPIO_TypeDef;
53/59
```

Съпоставяне на C структури към stm32l0xx_hal_gpio.h

адреси

```
void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init){
```

```
....
```

```
temp = GPIOx->AFR[position >> 3U];
```

```
temp &= ~((uint32_t)0xFU << ((uint32_t)(position & (uint32_t)0x07U) * 4U));
```

```
temp |= ((uint32_t)(GPIO_Init->Alternate) << (((uint32_t)position & (uint32_t)0x07U) * 4U));
```

```
    GPIOx->AFR[position >> 3U] = temp;
}
```

```
....
```

```
}
```

Контрол на отделни битове (bit-banding)

В програмирането на микроконтролерите много често се използват операции за промяна състоянието на **отделен бит** в **даден регистър**. За да се промени **само желания бит**, а останалите да са незасегнати, в регистър-към-регистър архитектура (load-store) трябва да се извършат **три операции** в тази последователност:

- *четене на регистъра (read)
- *промяна на желания бит (modify)
- *запис в регистъра (write)

или т.нар. read-modify-write операция. За да се избегнат всичките тези стъпки, проектантите на микроконтролери са измислили начин за контрол на отделните битове (bit-banding), който става **само с една операция запис в регистър** (set/clear) [4]. На следващия слайд е илюстриран този метод,

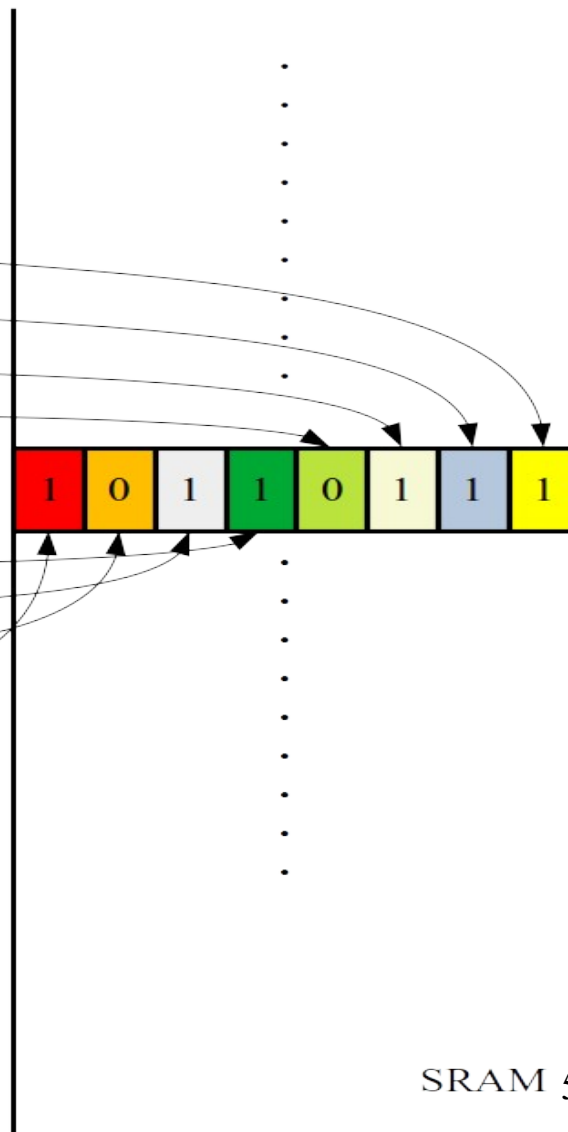
Контрол на отделни битове (bit-banding)

Битово-контролиран регион (bit-band region)

Псевдонимни регистри (bit-band alias)

0xC000	0	0	0	0	0	0	1
0xC001	0	0	0	0	0	0	1
0xC002	0	0	0	0	0	0	1
0xC003	0	0	0	0	0	0	0
0xC004	0	0	0	0	0	0	1
0xC005	0	0	0	0	0	0	1
0xC006	0	0	0	0	0	0	0
0xC007	0	0	0	0	0	0	1

0x00FB



Контрол на отделни битове (bit-banding)

Битово-контролиран регион (bit-band region) – част от адресното поле, която съдържа регистри с възможност за контрол на отделните им битове. Тези регистри може да са части от RAM паметта или регистри със специални функции от периферните модули.

Псевдонимен регистър (bit-band alias) – част от адресното поле, в което се намират регистри със само един значещ бит. Този бит отговаря на един единствен бит от друг регистър, разположен в битово-контролирания регион (bit-band region).

Обикновено адреса на псевдонимния регистър (bit-band alias) е **свързан чрез формула** с адреса и позицията на бита от регистъра в битово-контролирания регион (bit-band region).

Контрол на отделни битове (bit-banding)

При четене състоянието на отделен бит трябва да се извършат следните **три операции**:

- *четене на регистъра (read)
- *маскиране на останалите битове чрез лог. операция AND
- *логическо преместване надясно на желания бит (logic shift right)

С помощта на псевдонимните регистри може да се извърши и обратния процес на **четене на отделен бит** от битово-контролирания регион, което ще сведе операцията само до **едно четене на регистър**.

Литература

- [1]Г. Михов, “Цифрова схемотехника”, ТУ-София, 1999
- [2]ARM, Application note 88, DAI0088A, 2001
- [3]Freescale Semiconductor, M68HC11 Reference Manual, 2007
- [4]S. Rizvi, “Microcontroller Programming: An Introduction”, pp.117-118, CRC Press, 2021.
- [5]ARM, Application note 179, DAI0179B, 2007
- [6]М. Митев, “Микропроцесорна схемотехника”, записки на лекции, 2021