1. СТРУКТУРА И ПРИНЦИП НА РАБОТА НА ПРОМИШЛЕНИТЕ КОНТРОЛЕРИ

Въпросът за автоматизация на производството е толкова стар, колкото е старо и самото производство. Решаването на този въпрос е поверено на специфичен клас устройства и системи, наречени най-общо Системи за управление – Control Systems.

Въведение

Обект на системите за управление са машини, апарати, съоръжения, технологични процеси и др. – вж. фиг. 1.1. Исторически първи се появяват чисто механичните системи за управление. В началото на 20-ти век се появяват електромеханичните системи за управление (релейно-контакторни), които господстват до средата на 20-ти век.

Фиг. 1.1. Системи за управление.

С появата и развитието на електрониката, се появяват и развиват електронни цифрови системи за управление, реализиращи алгоритмите за управление чрез „твърда" (непрограмируема логика). За пръв път идеята за програмируемите логически системи за управление възниква през 60-ти години на 20-ти век във фирмите производителки на автомобили. По същото време започва и работа по създаването устройства за логическо управление, използуващи електрон-

но-изчислителни машини.

Бурното развитие и възходът на логическите системи за управление започва през 70-те години на 20-ти век и продължава и до днес. То безспорно е свързано с появата и усъвършенстването на микропроцесорните системи. Въвеждането на микропроцесорната техника в системите за управление върви по два пътя. При първият от тях, те буквално заместват твърдото логическо управление с програмно, запазвайки общата структура и изградените вече принципи за логическо управление. При втория, микропроцесорните системи се допълват с необходимата апаратна част (предимно интерфейсна) и необходимо програмно осигуряване за изграждането на система за управление.

През 80-те години на 20-ти век, повлияно от взривното развитие на компютърната техника, започва изграждането на компютърни системи за управление. В момента е трудно да се предскаже дали компютърните системи ще изместят програмируемите логически контролери в системите за управление. Съвременните програмируеми контролери представляват сложни микропроцесорни системи, с изключителна функционалност и комуникативност. Процесът на развитие на системите за управление върви по пътя на интегрирането на програмируемите логически контролери с компютърните системи.

Терминът „Програмируеми логически контролери" – Programmable Logical Controller (PLC) се използува за обозначаване на програмно-управлявани електронни системи, проектирани за използуване в качеството на промишлено оборудване за логическо управление на различни машини, съоръжения и технологични процеси през цифрови или аналогови входове и изходи.

Този термин PLC се използува предимно от европейските производители на устройства за логическо управление. В американската литература този тип устройства са познати като Програмируеми контролери – Programmable controller (PC). В българската литература като синоним се среща понятието Промишлени контролери.

Съдържанието на понятието „Програмируеми логически контролери" е дадено през 1978 г. от NEMA (National Electrical Manufactures Association). Промишлените контролери са програмно управлявани устройства. Това означава, че връзката между входните въздействия (аналогови или цифрови) и изходните реакции на системата (също аналогови и цифрови) се осъществява чрез последователност от инструкции, реализиращи алгоритъма на управление. Програмната връзка се съхранява в програмируемата памет на промишления контролер и може да бъде променена в процеса на настройване на работата (при някои контролери и в процеса на самата работа). Основните концепции на промишления контролер са формулирани още през 1968 г. от фирмата General Motors. Те се изразяват в следното:

– лесно програмиране и препрограмиране (промяна в последователността на изпълняваните операции), по възможност при устройството за управление;

– лесно поддържане и замяна, с предпочитание към модулна конструкция;

– повишена надеждност и минимални габаритни размери;
– конкурентна цена.

Принципите са относителни и са били формулирани по времето когато все още е доминирала релейно-контакторната логика, но те остават в сила и важат и днес.

Обобщена структура на промишлен контролер

Типичната структура на един промишлен контролер се състои от описаните по-долу основни блокове.

– ***Захранващ блок.*** Основното предназначение на захранващия блок е да осигури вътрешно необходимите работни напрежения за контролера и неговите блокове. Най-често използваните вътрешни напрежения са +5V, ±12V и +24V. Съществуват два основни типа захранващи блокове – такива, които ползват входно напрежение от мрежата (например 220V AC) и такива, които ползват оперативно захранващо напрежение от управляемия обект (например 24V DC). Допълнително, захранващият блок може да осигури на контролера сигнал за отпадане на входното напрежение (Power Down).

– ***Процесорен блок (CPU).*** Процесорният блок съдържа процесорната част на контролера. При промишлените контролери се използуват микропроцесори или едночипови микрокомпютри от стандартен тип, т.е. такива, които се използуват в микропроцесорните системи с общо предназначение. Процесорът на промишления контролер определя основните характеристики на процесорния модул, като разредност, бързодействие и т.н. В първите промишлени контролери са използувани специализирани процесорни чипове, като например AMD2901, 2903 и др.

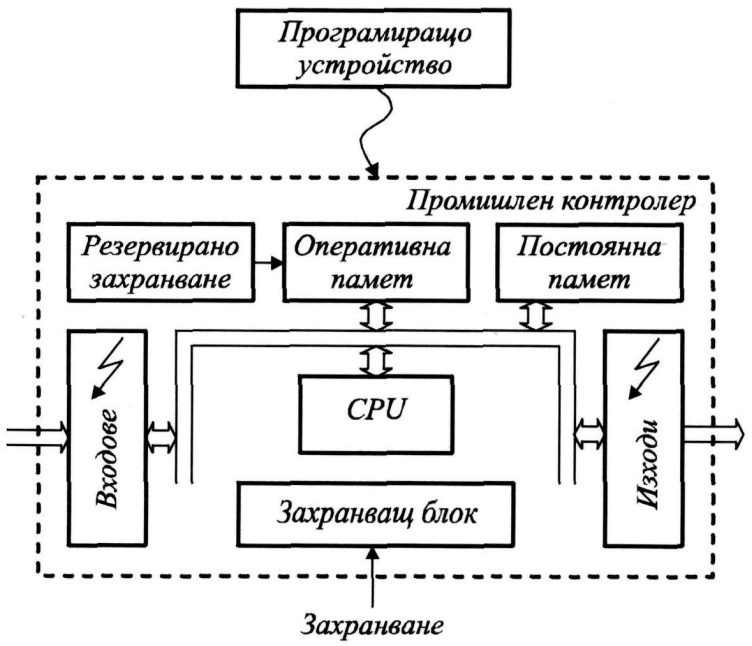
Към процесорният модул се включва и паметта на промишления контролер. Тя е разделена на постоянна и на оперативна памет. Постоянната памет се изгражда на базата на ROM, EPROM, EEPROM или FLASH. Цялата или част от постоянната памет може да бъде сменяема. Като оперативни памети най-често се използуват памети от статичен тип. Нормално част от оперативната памет (а може и цялата) е с резервирано захранване – батерийно или акумулаторно.

– ***Входни и изходни блокове.*** Входните и изходните блокове осигуряват интерфейса между промишления контролер и външния апарат или процес, който трябва да се управлява. Обикновено входните и изходните блокове са с галванично развързване към входните и изходните устройства на управляемия обект.

Общата блокова схема на промишлен контролер е показана на фиг. 1.2.

Както се вижда от блоковата схема, структурата на един промишлен контролер не се отличава от тази на микропроцесорна система с общо предназначение. Това е естествено, тъй като се ползува една и съща елементна база, концепция и организация. Ако трябва да се търсят някакви схемотехнически разлики и принципи на които се изграждат промишлените контролери, те биха

могли да се специфицират в следното:



Фиг. 1.2. Блокова схема на промишлен контролер.

– препоръчително галванично развързване на входовете и изходите на промишления контролер по отношение на управляемия обект, с цел повишаване на шумоустойчивостта и безопасността при работата му;

– резервирано захранване на част или на цялата оперативна памет, с оглед коректно и безопасно поведение на промишления контролер при аварийни ситуации на отпадане на захранващото напрежение.

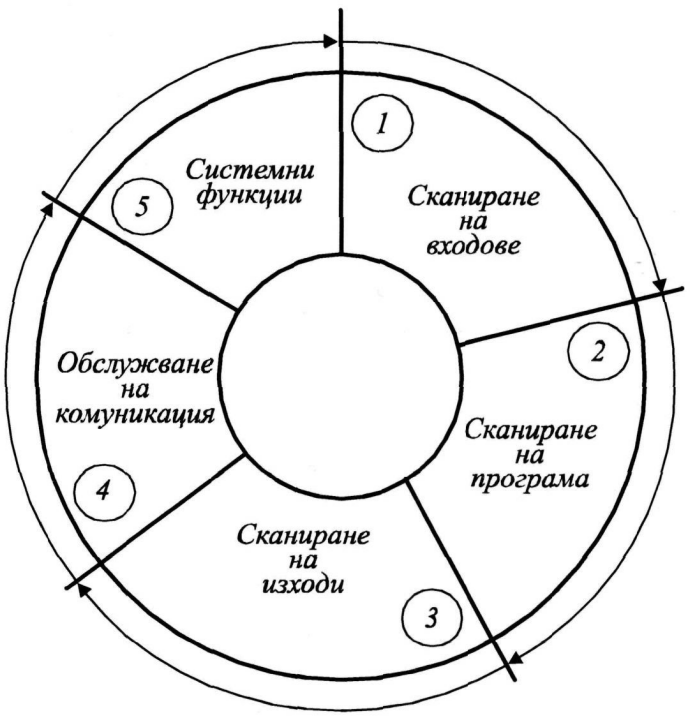
Принципи на работата на промишлените контролери

Основните различия между промишлените контролери и микропроцесорните системи с общо предназначение се състои в организацията на работата. Програмируемите логически контролери работят под управлението на операционна система за реално време (ОСРВ), която осигурява циклично изпълнение на логическата програма.

Операционната система осигурява последователното изпълнение на няколко основни дейности. Тази последователност, както и самото й изпълнение, се нарича оперативен цикъл (operating cycle) на промишления контролер. Основните дейности на промишлен контролер са 5 и са представени на фиг. 1.3. Те се изпълняват последователно и се наричат още „фази на оперативния цикъл".

Фаза 1 се нарича сканиране на входове (Input Scan). Физическите места, където постъпват входните сигнали в промишления контролер се наричат входни точки (Input Points). За всяка входна точка е определен по един бит в оперативната памет – т.нар. входен бит (Input Bit). Дейността по сканирането на входовете извършва прочитане на входните точки и записването на информацията от тях във входните битове. С други думи, в оперативната памет се създава „об-

раз" на входните въздействия към дадения момент (Process Image Inputs).



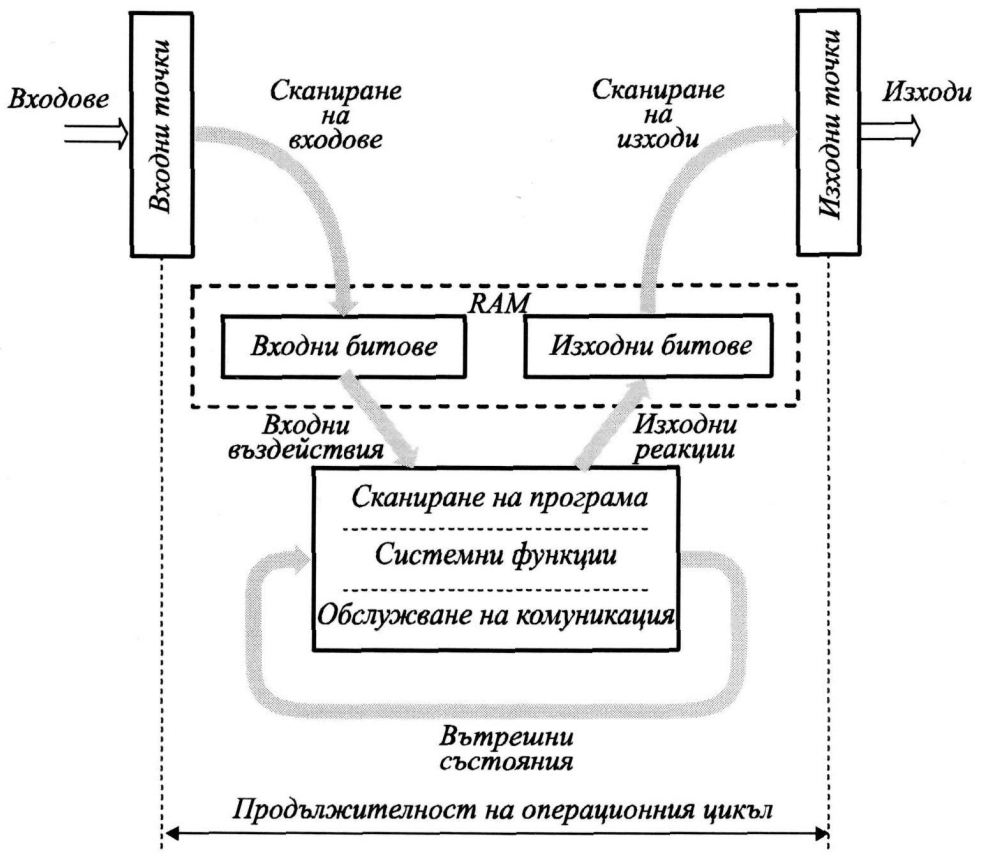
Фиг. 1.3. Оперативен цикъл на промишлен контролер.

Фаза 2 е сканиране на програмата (Program Scan). През тази фаза се изпълняват всички инструкции от логическата програма по отработване на входните въздействия и формиране на изходните реакции. Програмируемите логически контролери са последователностни устройства, в които при формирането на изходните реакции освен входните променливи участват и вътрешни за контролера променливи (таймери, броячи, компаратори и т.н.). В много ограничени случаи, програмируем логически контролер може да изпълнява само комбинационни логически функции (изходните реакции се определят еднозначно само от входните въздействия).

Фаза 3 се нарича сканиране на изходите (Output Scan). Физическите места, през които промишленият контролер изпраща изходните сигнали към управляемия обект се наричат изходни точки (Output Points). Както при входните точки, така и за всяка изходна точка е определен по един бит в оперативната памет – т.нар. изходен бит (Output Bit). Дейността по сканирането на изходите извършва прочитане на изходните битове и прехвърлянето на информацията от тях в изходните точки. Отново, в оперативната памет се създава „образ", сега вече на изходните реакции в дадения момент (Process Image Outputs).

Фаза 4 е предназначена да обслужи комуникацията (Service Communication). В тази част на оперативния цикъл се извършва комуникация с други устройства, например програмиращо устройство, централен компютър, устройство за човеко-машинен интерфейс (операторски пулт, панел, терминал) и др. В повечето програмируеми логически контролери, комуникацията се обслужва главно от операционната система на контролера, според специални ди-

ративи, записани в управляващата програма. Някои контролери позволяват директно управление на комуникацията от потребителската програма чрез съответната системна функция.



Фиг. 1.4. Продължителност на операционния цикъл на промишлен контролер.

Фаза 5 е предназначена за изпълняване на системни функции (Housekeeping and Overhead). Към тази дейност влизат разнообразни действия, по-важни от които са:

– управление на паметта;

– управление на вътрешните регистри на микропроцесора;

– обслужване на вградените специални апаратно-програмни функции (Firmware Function) като броячни входове, импулсни входове и изходи, допълнителни комуникационни портове и др.;

– извикване на вградени логически функции, които имитират работата на хардуерни устройства, използувани за целите на логическото управление като устройства за времезадържане (таймери), устройства за отброяване на външни или вътрешни за контролера събития (броячи) и др.;

– тестване на апаратно-програмните функции на промишления контролер (Self Diagnostic Test).

Фиг. 1.4 дава нагледна представа за дейностите, извършвани от промишления контролер в отделните фази на оперативния цикъл.

Организация на паметта при промишлените контролери

За разлика от микропроцесорните системи с общо предназначение, изграждането на паметта на промишлените контролери се подчинява преди всичко на режимите на работа, които те трябва осигуряват и които са съобразени със специфичните условия на работа и изпълняваните задачи. Основните режими, в които трябва да обезпечават промишлените контролери са 4:

– режим на програмиране на промишления контролер (Download);
– режим на нормална работа (Run Mode);
– режим на отпадане на захранващото напрежение (Power Down);
– режим на възстановяване на захранващото напрежение (Power Up).

Постоянната памет на съвременните промишлените контролери е от EEPROM или FLASH тип. Една част от нея, която съдържа системните функции, може да бъде и от ROM или EPROM тип. Предназначението на постоянната памет е да съдържа потребителската програма и системните и потребителски данни (статични системни и потребителски даннови секции), които не се променят по време на работа. С използването на репрограмируеми постоянни памети по технологията FLASH не се затрудняват промените в програмното осигуряване при неговата настройка.

Оперативната памет в промишлените контролери съдържа т.нар. динамични даннови секции. Това са променливите данни на операционната система (системна информационна област) и данните (информационните области), осигуряващи обмена на информация между различните задачи в потребителската програма – образ на входовете и изходите, таймери, броячи, памет на променливите, вътрешни релета и др. В някои промишлени контролери, в оперативната памет се помества и работната програма, т.е тя се използува и като програмна памет.

Оперативните памети в промишлените контролери са от статичен тип. Тъй като информацията в RAM се изгубва при отпадане на захранващото напрежение (аварийна ситуация), се налага да се вземат специални програмно-апаратни мерки по възстановяване на работата на контролера след възстановяване на захранването. В зависимост от конкретния обект, управлението трябва да се възстанови от момента, предшестващия аварията или по друг указан алгоритъм.

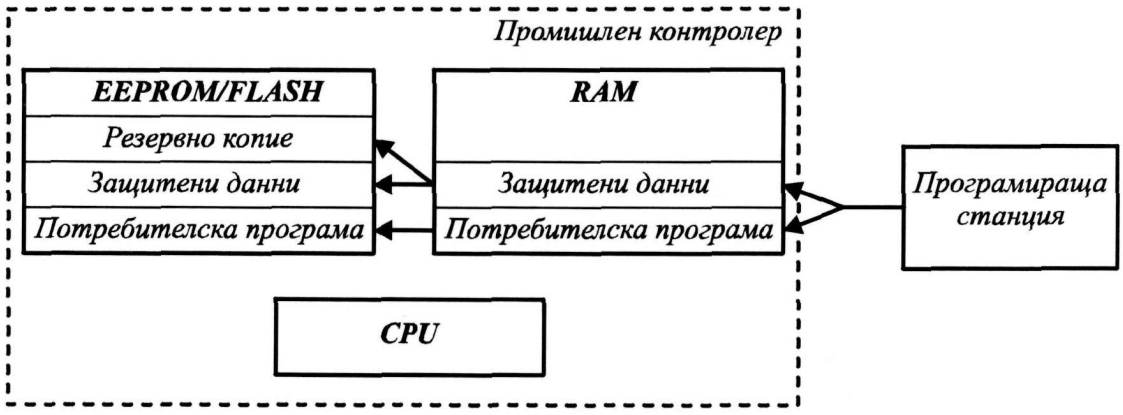
Апаратното осигуряване за отработването на аварийните ситуации с отпадането на захранващото напрежение при повечето производители включва защита на част от оперативната памет с батерия или кондензатор, т.нар защитени данни (Retentive Data). Обикновено, в тази част влизат цялата област на системните данни, заявената за защита част от потребителските данни, част от областта на вътрешните релета и част от областта на броячите.

Задължително апаратно осигуряване е наличието на сигнал за отпаднало захранване, малко преди контролерът да го загуби напълно. Такъв сигнал

обикновено се подава на апаратен вход за немаскируемо прекъсване на контролерния процесор и задейства процедурата за отработване на аварийната ситуация.

Програмният механизъм за отработване на аварийните ситуации при различните контролери се различава в някаква степен, но неговите принципи са еднакви.

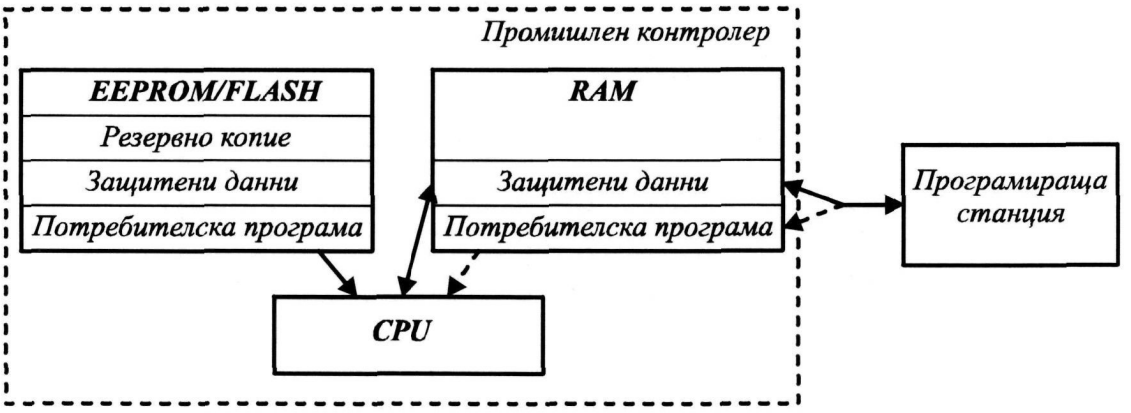
На фиг. 1.5 е даден режимът на програмиране на промишления контролер. Потребителската програма (Program Files) и защитените данни се прехвърлят от програмиращото устройство в RAM паметта на промишления контролер. Оттам потребителската програма и защитените данни се запомнят в енергонезависима EEPROM или FLASH памет. При контролери, които не подържат батерийно резервирани данни в RAM, защитените данни се копират още веднъж в енергонезависимата памет като резервно копие (Backup Data).



Фиг. 1.5. Режим на програмиране на промишлен контролер.

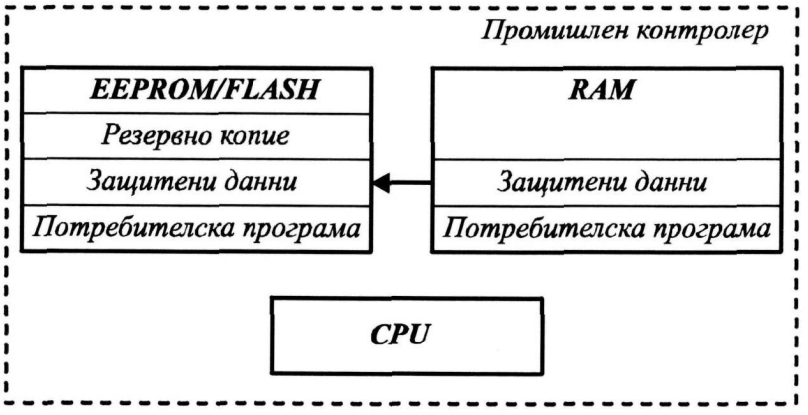
В режимът на нормална работа, промишленния контролер изпълнява потребителската програма по управление от енергонезависимата памет. По време на работата настъпва промяна в областта на защитените данни. Тя се отразява единствено в RAM паметта, тъй като процесорът работи с нея. В този режим програмиращата станция може да чете и променя областта на защитените данни в RAM паметта, т.е може да променя хода и параметрите на управляващата програма без да променя самата програма. Фиг. 1.6. илюстрира режима на нормална работа.

При някои контролери, изпълнението на потребителската програма се извършва от RAM паметта. В този случай програмиращата станция може да променя и самата потребителска програма. Последното не се препоръчва да се извършва в работни условия, но има приложение, когато управляващата програма се тества върху макет на съоръжението или процеса.



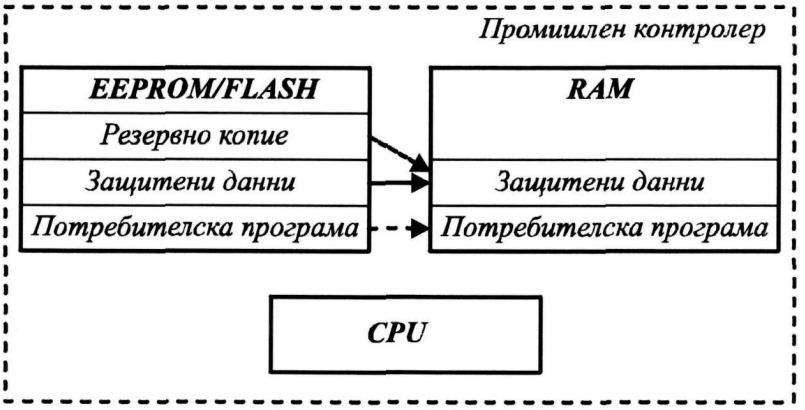
Фиг. 1.6. Режим на нормална работа на промишлен контролер.

Режимът на отпадане на захранващото напрежение се активира, когато промишленият контролер получи сигнал от захранващия блок Power Down. При него се извършва прехвърляне на защитените данни от RAM в енергонезависимата памет – вж. фиг. 1.7.



Фиг. 1.7. Режим на отпадане на захранващото напрежение при промишлен контролер.

Режимът на възстановяване на захранването се активира след начално установяване, при положение, че в енергонезависимата памет е вдигнат флаг за спиране поради отпадане на захранващото напрежение. При контролерите, които работят с потребителска програма в RAM се извършва нейното прехвърляне от енергонезависимата памет. Проверява се защитената RAM за наличие на данни в нея. При спиране за по-дълго време, резервираното батерийно или кондензаторно захранване може също да е отпаднало. След това се проверява за наличието на записани защитени данни в енергонезависимата памет. Ако те са налице се извършва възстановяването им в RAM и привеждане на контролера в нормален режим на работа. Ако по някаква причина те не са се записали, например при бързо отпадане на захранващото напрежение, възстановяването на защитените данни се извършва от резервното копие в енергонезависимата памет.



Фиг. 1.8. Режим на възстановяване на захранването при промишлените контролери.

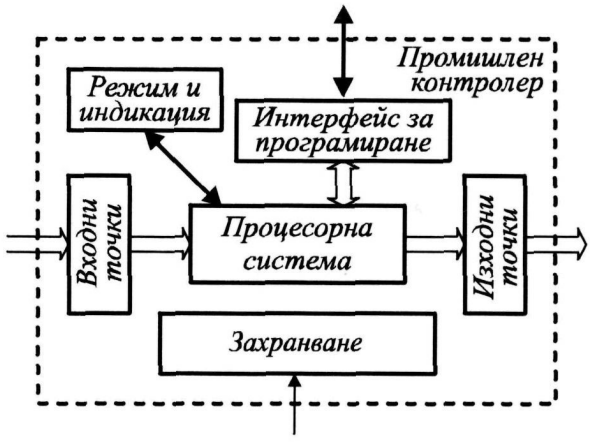
2. АПАРАТНА И ПРОГРАМНА АРХИТЕКТУРА НА ПРОМИШЛЕНИТЕ КОНТРОЛЕРИ

Основните характеристики на промишлените контролери се определят от възможностите им за управление на обекти (машини, съоръжения, и технологични процеси). Те могат да се съсредоточат в три насоки:

1. Максимален брой и тип на входните и изходните точки;
2. Минимална продължителност на входните сигнали;
3. Обем и организация на паметта за работната програма.

Компактна и модулна архитектура на промишлените контролери

Посочените по-горе три основни черти на промишлените контролери са взаимосвързани и пряко влияят и се определят от хардуерната архитектура на процесора. На най-ниско ниво стои компактната архитектура на промишлените контролери. При нея, всички атрибути на контролера са съсредоточени в един физически модул, който в някои случаи не подлежи на разширение. Той съдържа захранване, процесорна система, определени количества входни и изходни точки, интерфейс за програмиране и блок за режим на работа и индикация – вж. фиг. 2.1.



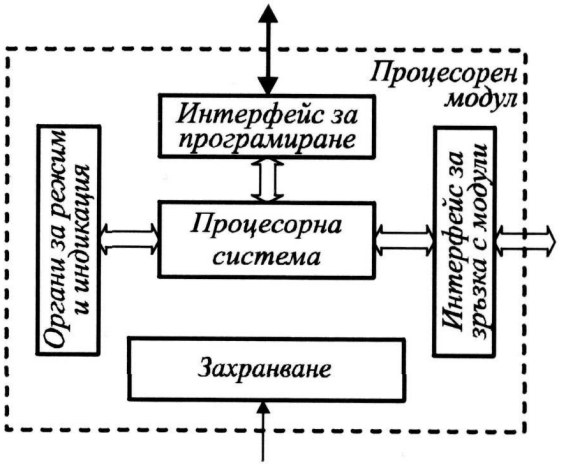
Фиг. 2.1. Компактна архитектура на промишлен контролер.

Промишлен контролер с такава архитектура може да управлява сравнително малки обекти – при тях броят на входно-изходните точки е ограничен (обикновено 8 – 24) и не подлежи на нарастване. Типът на входно-изходните точки също е твърдо определен (с малки изключения). В стремежа си да увеличат приложението на този вид контролери, фирмите производителки предлагат серия компактни контролери, различаващи се най-вече по брой и тип на входно-изходните точки.

Някои производители на серията могат да притежават локален интерфейс за програмиране (клавиатура и индикация), позволяващи им програмирането да се извърши на място, без използването на програмираща станция. Тенденция в развитието на този клас контролери е снабдяването им с интерфейс за индустриална мрежа (например AS-interface), позволяващи им връзка като подчинени устройства с други контролери и участието им в по-голяма система за управление.

Противоположно на компактната архитектура на промишлените контролери стои модулната архитектура. При нея промишленият контролер се изгражда от отделни модули, като потребителят решава колко и какви модули да влезнат в състава на контролера, така че максимално да се удовлетворят изискванията на управляемия обект. В някои контролери модулите се помещават в касета, на дъното на която е разположена системна магистрала за връзка между тях. По този принцип се изграждат големите комплексни системи за управление.

Основният модул в контролер с модулна архитектура е процесорният модул. Той определя характеристиките на промишления контролер като цяло (обем на паметта за работната програма, тип и бързодействие на вградената операционна система, брой на входно-изходните битове, възможност за работа в мрежа, връзка с интелигентни модули и др. Процесорният модул съдържа захранващ блок, самата процесорна система, интерфейс за програмиране, локални органи за задаване на режим на работа и индикация на състоянието и интерфейс за връзка с модулите – вж. фиг. 2.2.



Фиг. 2.2. Процесорен модул.

В големите модулни системи е обособен отделен захранващ блок, работещ с входящо напрежение 220V AC и изработващ операционно напрежение (обикновено 24V DC) за всички модули на контролерите. В такива системи, вътрешните захранващи блокове работят с входящото напрежение и изработват от него необходимите напрежения само за дадения модул.

Процесорният модул на разпределена система се предлага от фирмите производителки в няколко разновидности, различаващи се по своите възможности. Локалните контролно-управляващи органи на процесорния модул включват:

1. Селектор на режима на работа (Mode Selector) чрез които се задават следните режими:

– MRES – функция за начално установяване на модула;

– STOP – Стоп режим, при който програмата не се изпълнява;

– RUN – режим на изпълнение на работната програма.

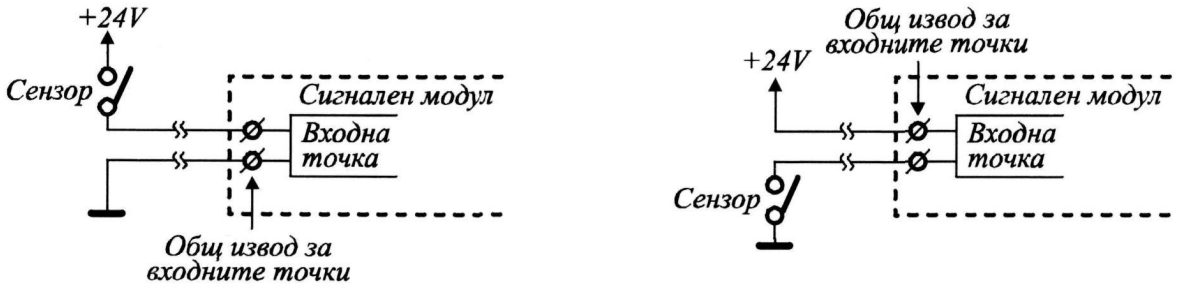
2. Индикатор на състоянията (Status indicator) – обикновено това са светодиодни индикатори, които с постоянно светени или мигане дават информация за текущото състояние на контролера, наличие на грешки и т.н.

Нормално процесорният модул съдържа и интерфейс за директна връзка на разпределени входно-изходни устройства към CPU.

Производителите на промишлени контролери предлагат широка гама от допълнителни входно-изходни модули, съобразени с най-често срещаните сензори и изпълнителни механизми. Те са проектирани по такъв начин, че да минимизират междинните устройства между промишления контролер и управлявания обект. Следващата класификация дава обща представа за различните видове модули:

1. Сигнални модули (SM – Signal Modules)

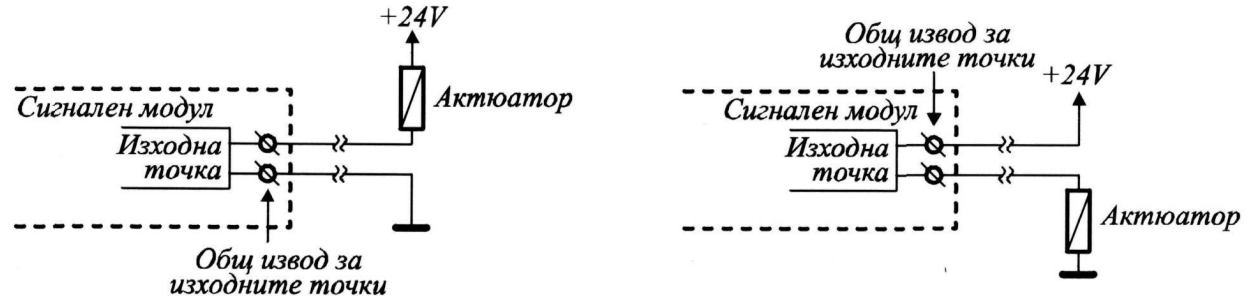
– Цифрови входове – предназначени са за свързване към сензори от логически тип (с две състояния – включено и изключено). Различават се по типа на входните сигнали, които могат да обработват – 24V DC, 120/230V AC, както и по начина на свързване на общия входящ проводник (към нулевия или към захранващия проводник) – вж. фиг.2.3.



а) свързване с обща маса - "PNP вход" *б) свързване с общо захранване -"PNP вход"*

Фиг. 2.3. Свързване на сензори към входни точки в промишлен контролер.

– Цифрови изходи – предназначени са за свързване към изпълнителни механизми от контактен тип (актюатори от логически тип). При този тип модули се наблюдава голямо разнообразие на използваните изходни елементи (релета, биполярни или MOS транзистори), при свързване на товара към масата или към захранващия проводник – вж. фиг. 2.4.



а) свързване с обща маса - "NPN изход" *б) свързване с общо захранване - "PNP изход"*

Фиг. 2.4. Свързване на актюатори към изходни точки на промишлен контролер.

– Аналогови входни модули – това са модули, предназначени да въвеждат в промишления контролер и да дискретизират аналогови сигнали. Различават се

по вида на входната величина (напрежение, ток, съпротивление) по точността на преобразуване, по работен диапазон, честота на дискретизация и др. Съществуват и специализирани аналогови модули за обработка на сигнали от определен тип датчици (термодвойки, PT100, тензометрични мостове и др.)

– Аналогови изходни модули – генерират изходни сигнали напрежение или ток.

Нормално в аналоговите входни и изходни модули са включени допълнително апаратни блокове, извършващи операции по филтриране, мащабиране и линеаризиране на обработваните сигнали.

2. Интерфейсни модули (IM – Interface Modules). Тези модули нямат входно-изходни точки. Служат за осигуряване на връзка между различни касети с модули. Те се използуват тогава, когато необходимата конфигурация на промишления модул не може да се помести в една касета, и се налага да се разположи в повече касети.

3. Функционални модули (FM – Function Modules). Този тип модули са специализирани относно управлението на конкретна задача за управление (броене, позициониране, управление с обратна връзка, регулиране и др.). Често те са интелигентни модули с вграден микроконтролер. Чрез тях се увеличава изчислителната мощност на промишления контролер, без да се увеличава –времето на оперативния цикъл.

4. Комуникационни процесори (CP – Communication Processors). Тези модули са предназначени за изграждане на локални мрежи и разпределени системи за управление с промишлени контролери. Основните интерфейси, използувани в разпределените системи за управление са:

– Point-to-point – връзка само между два кореспондента;

– ASi-bus – връзка между едно главно устройство с подчинени устройства, представляващи сензори и изпълнителни механизми от релеен тип по двупроводна линия, съвместяваща захранване и информационни сигнали;

– Modbus, Profibus, Berges и др. – мрежови интерфейси за връзка между един главен промишлен контролер и подчинени контролери;

– Industrial Ethernet и др – мрежови интерфейси за връзка между равнопоставени кореспонденти, предназначени за изграждане на големи информационно-управляващи системи.

Междинно място заемат контролерите с полу-компактна архитектура. При тях процесорният модул съдържа освен захранващ блок и определено количество входно-изходни точки и може да работи самостоятелно и да изпълнява управленчески задачи. Същевременно в него са предвидени възможности за разширение чрез включването на допълнителни модули, съдържащи допълнителни входни и изходни точки.

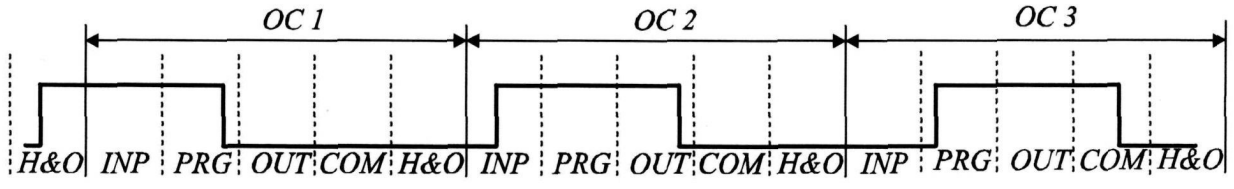
Системите за управление се изграждат по съсредоточена и разпределена структура.

Минимална продължителност на входните въздействия

Начинът на работа на промишления контролер, последователното обхождане на отделните фази, довежда до някои ограничения, които трябва да се имат предвид при избора на подходящ контролер за дадено приложение. Двете основни ограничения които съществуват са минимална продължителност на входните сигнали и минималното време от възникването на входно въздействие и изработването на адекватна изходна реакция.

Изискванията за минимална продължителност на входните сигнали възниква поради факта, че потребителската програма няма достъп до входните точки на контролера, а работи с областта на входните битове и освен това входните битове се опресняват само във фазата на сканиране на входовете. На следващите фигури са показани три примера на постъпващо входно въздействие. Със съкращенията IN, PRG, OUT, COM и SYS са означени съответните фази на оперативния цикъл ОС на контролера. Активирането на входната точка се извършва асинхронно спрямо операционния цикъл на промишления контролер.

На фиг. 2.5. е показан пример при който пристигат входни въздействия с продължителност по-малка от максималната продължителност на оперативния цикъл T_{OCmax}. В първия оперативен цикъл, входното въздействие попада във времето на сканирането на фазата IN и ще бъде отчетено. Във втория оперативен цикъл, входно въздействие със същата продължителност засяга част от фазата на сканиране на входовете и може да бъде или да не бъде отчетено, в зависимост от моментът на стробиране на входната информация. В третия оперативен цикъл входното въздействие не покрива фазата на сканирането на входовете и не може да бъде отчетено.

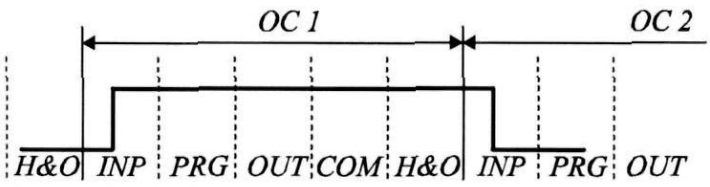


OC - Оперативен цикъл;
INP - Сканиране на входовете;
PRG - Сканиране на програмата;
OUT - Сканиране на изходите;
COM - Обслужване на комуникацията;
H&O - Системни функции.

Фиг. 2.5. Отчитане на входно въздействие с продължителност по-малка от продължителността на оперативния цикъл.

Анализът на тези три случая показва, че за да попадне дадено въздействие във фазата на сканиране на входовете е необходимо то да бъде с продължителност не по-малка от времето на оперативния цикъл. На фиг. 2.6 е показан случай при който продължителността на входното въздействие е точно равна на

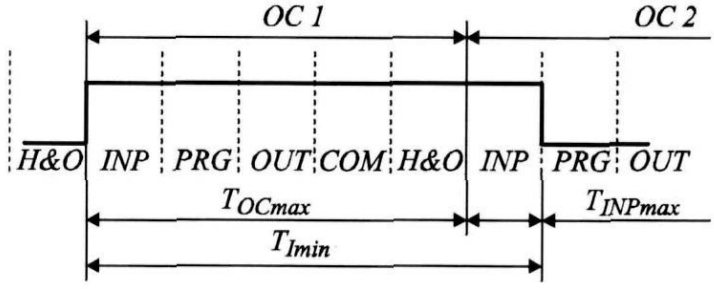
максималната продължителност на оперативния цикъл. Тогава обаче съществуват ситуации, при които входното въздействие може да не „покрие" изцяло фазата на сканиране на сканиране на входовете. Както е показано на фигурата, то се активира в даден момент на първия оперативен цикъл и се дезактивира в същия момент на следващия цикъл, т.е съществува вероятност прочитането (стробирането) на входната информация да се извърши точно в този преходен момент и въздействието да не може да бъде възприето. Ясно е, че продължителността на входното въздействие трябва да е по-голяма от максималната продължителност на оперативния цикъл.



Фиг. 2.6. Отчитане на входно въздействие с продължителност равна на продължителността на оперативния цикъл.

За да се гарантира правилното възприемане на входното въздействие е необходимо да се осигури условието то да покрие изцяло поне една фаза по сканиране на входовете (вж. фиг. 2.7), а това е възможно само ако минималната продължителност на входното въздействие е равна на максималното време на оперативния цикъл плюс максималното време за сканиране на входовете:

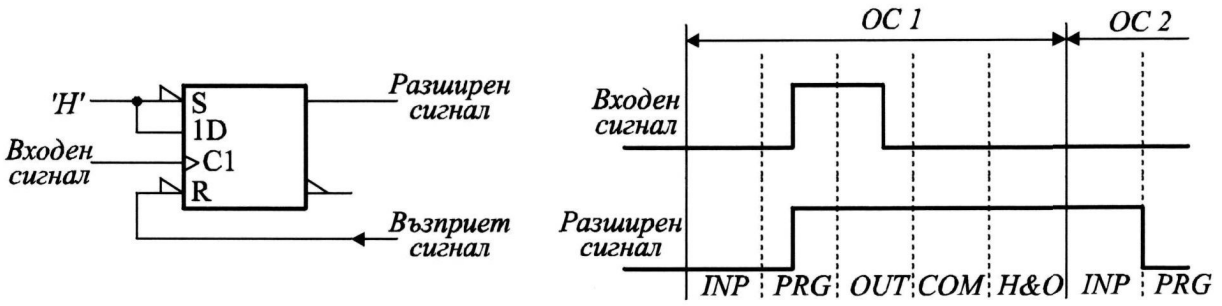
$T_{\text{Imin}} = T_{\text{OCmax}} + T_{\text{INPmax}}$.



Фиг. 2.7. Минимална продължителност на входно въздействие.

От казаното до тук следва, че входни въздействия с продължителност помалка от минималната не могат да бъдат обработени от стандартните средства на операционната система на промишления контролер. За такива случаи се въвеждат специални методи, които най-често се свеждат до два.

Първият използува т.нар. „разтягане на импулса", което се реализира по апаратно-програмен път. Фиг. 2.8 илюстрира този начин на обработване. При активен фронт на входното въздействие, то се запомня в апаратен тригер докато премине фазата на сканиране на входовете, когато тригерът се нулира. Нулирането на тригера трябва да стане по програмен начин, само ако е възприето въздействието.



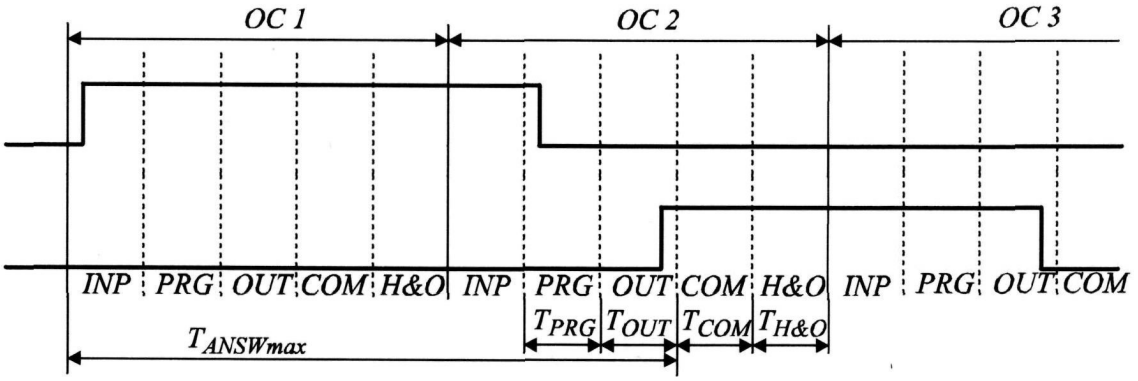
Фиг. 2.8. Апаратно програмен метод за „разтягане" на кратко въздействие.

Вторият начин за възприемане на краткотрайни въздействия е използването на прекъсвания. Входящия сигнал се подава на вход за апаратно прекъсване на микропроцесорната система на промишления контролер. Активирането му предизвиква прекъсване на оперативния цикъл на контролера и извикването на програма, обслужваща това прекъсване (Interrupt Function). Оперативния цикъл се възстановява в местото където е прекъснат, след изпълнение на прекъсващата програма.

Максимално време на реакция

Друг важен параметър на промишлените контролери е т.нар. максимално време на реакция. То се определя от момента на постъпване на входното въздействие до изработването на адекватна изходна реакция. От направените по-горе разглеждания за минимална продължителност на входящото въздействие може да се заключи, че в най-лошия случай активирането на входа ще се отчете във втория оперативен цикъл, т.е. може да се запише:

$$T_{\text{ANSmax}} = T_{\text{Imin}} + T_{\text{PRG}} + T_{\text{OUT}} = 2T_{\text{OC}} - T_{\text{H\&O}} - T_{\text{COM}} .$$



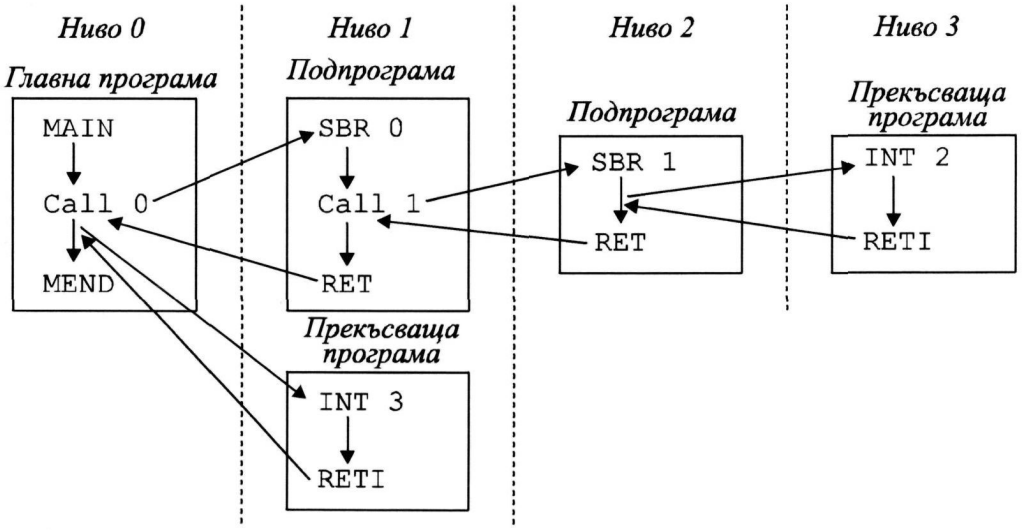
Фиг. 2.9. Максимално време на реакция на промишлен контролер.

Прекъсващи програми при промишлените контролери

Прекъсващите програми осигуряват бърза реакция от страна на потребителската програма. Прекъсванията могат да бъдат външни (предизвикани от

управляемия обект) и вътрешни (предизвикано от вътрешно събитие, като например установяване на таймерен бит и др.).

Организацията на прекъсванията при програмируемите логически контролери е подобно на тези при микропроцесорните системи за обща употреба и е илюстрирана на фиг. 2.10.



Фиг. 2.10. Механизъм на обслужване и нива на вмъкване на подпрограми и прекъсващи програми.

Прекъсванията могат да се възприемат по всяко време на изпълнение на оперативния цикъл, независимо от това дали се изпълнява главната програма или подпрограма. Едновременно с това обаче, обработката на прекъсванията може да се забрани през определено време на работата на контролера.

Когато операционната система на промишления контролер регистрира прекъсващо събитие, тя създава т.нар. точка на прекъсване и предава управлението на програма за обслужване на прекъсването, наречена прекъсваща програма. Процедурата по предаване управлението на прекъсващата програма се нарича влизане в прекъсване.

Прекъсващата програма се изпълнява еднократно, след което операционната система възстановява изпълнението на програмата в точката на прекъсването. Процедурата по възстановяване на изпълнението на програмата се нарича връщане от прекъсване.

Външните прекъсвания се делят на два вида:

– импулсни прекъсвания, които регистрират настъпването на определено събитие, обикновено постъпващо на апаратен вход, свързан към входа за прекъсване на микропроцесора;

– броячни прекъсвания, които се генерират от вградени в промишления контролер апаратно-програмни блокове, като бързи броячи, таймери и т.н.

Външните прекъсвания се регистрират при възникването им по всяко време на операционната система. Тяхното обслужване обаче може да бъде забавено. Повечето от промишлените контролери обработват външните събития само по

време на фазата на сканиране на програмата.

Вътрешните прекъсвания се наричат още системни прекъсвания, тъй като се генерират от събития, възникнали при работата на операционната система. Най-общо те могат да се разделят на три групи:

– интервални (таймерни) прекъсвания, формиращи прекъсвания през зададен интервал от време;

– прекъсвания от грешки, които се генерират при възникване на грешки в потребителската програма (например делене на нула);

– комуникационни прекъсвания, които се използуват за обслужване на комуникацията.

Използуването на прекъсващи програми изисква от програмиста да има предвид следните основни момента:

– прекъсващите програми увеличават оперативния цикъл на контролера;

– операционната система на промишлените контролери има ограничен брой нива на вмъкване на подпрограми и прекъсващи програми в главната програма.

Виртуална даннова област (Data Area)

Подобно на данновата област (оперативната памет) при реалните микропроцесорни системи, в PLC е на разположение виртуална даннова област. Междинният код (междинните инструкции) на PLC работят с виртуалната даннова област. Интерпретаторът е този, който превежда адресите от виртуалната даннова област в реални адреси за конкретната микропроцесорна система в контролера. Основни подобласти на виртуална даннова област са дадени в табл. 2.1

Таблица 2.1.
Типичните подобласти на виртуалната даннова област при промишлените контролери

Наименование на подобластта	Примерно означение	Име
Образ на входовете	I	Process Image Inputs
Образ на изходите	Q	Process Image Outputs
Таймери	T	Timers
Броячи	C	Counters
Вътрешни релета	M	Bit Memory
Памет за променливите	V	Variable Memory
Системна памет / Специални битове	SM	System Memory / Special Bits
Последователностни релета	SRC	Sequence Control Relay

Образ на входния процес (Process Image Inputs). В тази подобласт операционната система записва състоянията на входните точки.

Образ на изходния процес (Process Image Outputs). От тази подобласт операционната система копира изчислените от потребителската програма състоянията на изходните точки.

Таймери (Timers). В тази подобласт се разполагат атрибутите на вградените

програмни таймери. Операционната система на контролера поддържа няколко типа таймери, обикновено с фиксирана база време.

Броячи (Counters). Подобластта съдържа атрибутите на вградените програмни софтуерни броячи. Нормално се поддържат няколко типа софтуерни броячи.

Вътрешни релета (Bit Memory). Тук се запазват състоянията на т.нар. „оперативни" релета, имащи отношение към управлението на обекта, но които не са свързани към конкретен изход. Обикновено в тази област се запомнят и междинните резултати при работата на програмата с променливи от тип бит.

Памет на променливите (Variable Memory). Областта е предназначена за запазване на резултатите от междинни изчисления, извършвани върху числови операнди. Тук обикновено се запазват и необходимите за управлението на обекта числови константи.

Системна памет / Специални битове (System Memory / Special Bits). Чрез тази подобласт се подържа комуникацията между операционната система на контролерите и потребителската програма за управление на обекта. Посредством подобластта потребителската програма може да управлява специалните функции, включени в апаратно програмната част на контролера.

Последователностни релета (Sequence Control Relay). Тази подобласт се използва за организиране на постъпково изпълнение на потребителската програма (работа на краен автомат).

Подобластите на виртуалната даннова област при различните контролери се различават незначително. Например при SIMATIC S7-200 са обособени допълнителни подобласти.

Аналогови входове (Analog Inputs). Подобластта е предвидена за адресиране на многовходови аналогово-цифрови преобразуватели, разположени в основния модул или върху допълнителен хардуерен модул. Стойността на аналоговата величина (например температура, ток, напрежение и др.) се съхранява като 16-битова цифрова стойност. Подобластта е позволена само за четене от потребителската програма.

Аналогови изходи (Analog Outputs). Подобластта е предвидена за адресиране на цифрово-аналогови преобразуватели, разположени върху допълнителни хардуерни модул. Модулът преобразува 16-битова цифрова стойност в ток или напрежение. Подобластта е позволена само за запис от потребителската програма.

Акумулатори (Accumulators). Подобластта се използва за предаване на параметри от и към отделните подпрограми, както и за запомняне на междинни резултати от математически изчисления.

3. ОРГАНИЗАЦИЯ НА ПРОГРАМНОТО ОСИГУРЯВАНЕ НА ПРОМИШЛЕНИТЕ КОНТРОЛЕРИ

Операционната система на промишлените контролери (ОСРВ) е вградена в процесорния модул. Тя осигурява обслужването и изпълнението на потребителската програма. Състои се от две основни части – фиг. 3.1:

Първата част е управляващата част (управляваща програма), която изпълнява следните основни функции:

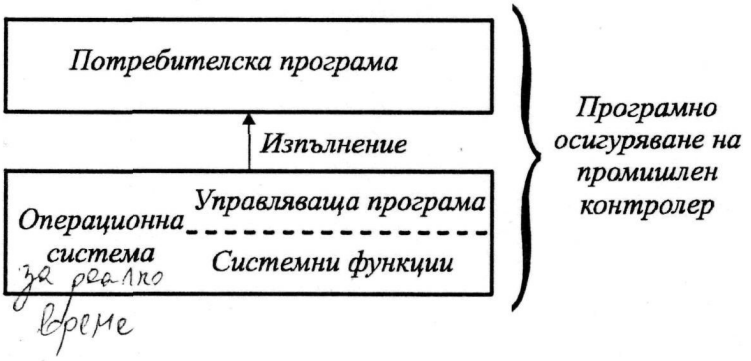
– управление на задачите в потребителската програма, в което влиза начално установяване на потребителските задачи, обслужване на основната задача (т.нар. бавна задача или главна програма) и управление на потребителските прекъсващи програми (т.нар. бързи задачи);

– управление на приоритета на задачите;

– осигуряване на обмен на информация между задачите в потребителската програма;

– циклично обновяване на информацията в потребителските входни и изходни даннови области.

Втората част е системна част (системни функции), която предоставя на потребителя програмни и апаратно-програмни функции като таймери, броячи и др.



Фиг. 3.1. Операционна система на промишлен контролер.

В зависимост от метода, по който се изгражда, разработва и програмира програмното осигуряване на промишления контролер, в операционната му система могат да бъдат включени още две части:

– обработваща част (интерпретатор), която превежда инструкциите на потребителската програма на обектов код за конкретния микропроцесор в промишления контролер;

– помощна част (помощни функции), съдържаща набор от готови подпрограми за най-често употребявани функции (например математически).

Програмиране и изпълнение на потребителската програма

Съществуват два различни метода за програмиране и изпълнение на потре-

бителската програма. Първият метод се използува при микропроцесорните системи с общо предназначение. Той изисква разработка на програмното потребителско осигуряване на развойна станция, неговото директно транслиране в обектов код за микропроцесорната система и зареждането му в промишления контролер – вж. фиг. 3.2.

В много случаи от развойната система се зарежда и самата операционна система за реално време.

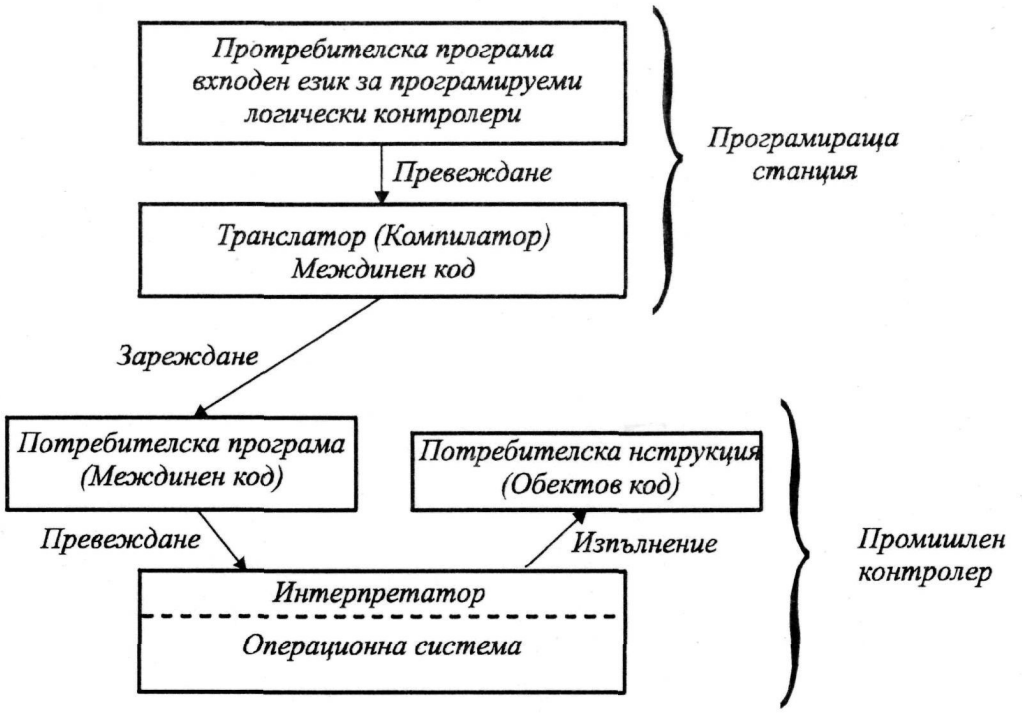
Фиг. 3.2. Използуване на транслатор за превеждане на потребителска програма.

При втория метод програмното осигуряване на промишления контролер се разработва на програмираща станция и се транслира до междинен код, който се зарежда в контролера. В процеса на изпълнението на потребителската програма междинният код се превежда на обектов код на микропроцесорната система в контролера от вграден в ОСРВ интерпретатор – вж. фиг. 3.3. При този метод интерпретаторът представлява третата основна част на ОСРВ.

Интерпретаторът поема инструкция от междинен код, транслира я в обектов код за микропроцесора и след това го изпълнява.

По принцип интерпретаторът е по-бавен от транслатора, тъй като в хода на работата се налага и извършване на процеса по „превеждане" от междинен код на обектов код на микропроцесора.

Съществуват промишлени контролери, при които интерпретирането е поверено на отделен апаратно-програмен процесор (подобно на комуникационните процесори), наречен езиков процесор.



Фиг. 3.3. Използуване на интерпретатор при изпълнение на потребителска програма.

Междинен код за програмируеми промишлени контролери.

Междинният код представлява набор от инструкции за работа с виртуална микропроцесорна система. Инструкцията (Statement) е закодирана съвкупност от действия, които трябва да извърши процесорът след превеждането й от интерпретатора. Всяка инструкция съдържа информация за вида на конкретната операция, формата на информацията, адреса на информацията и др. Според организацията си инструкциите се делят на триадресни, двуадресни, едноадресни и безадресни. Всяка инструкция има свое мнемонично описание, използувано от транслатора в междинен код.

Класическата триадресна инструкция съдържа код на операцията (КОП) и три операнда, като третият операнд най-често е резултатът от операцията между първите два операнда – вж. фиг. 3.4.

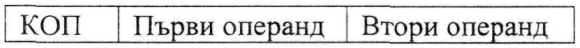
КОП	Първи операнд	Втори операнд	Трети операнд

```
FOR IND, INIT, FNL
...
...
...
NEXT
```

Фиг. 3.4. Пример за триадресна инструкция в промишлен контролер SIMATIC S7-200, която организира циклично изпълнение на всички инструкции, записани между FOR и NEXT, започвайки от начална стойност INIT на променливата IND до крайна стойност FNL.

Триадресните инструкции заемат най-много памет за своята организация. В промишлените контролери техният брой е ограничен и те се използуват само за специални цели. Някои промишлени контролери не поддържат триадресни инструкции.

Двуадресните инструкции съдържат само код на операцията и два операнда – фиг. 3.5.




```
MOVB INA, OUTA
```

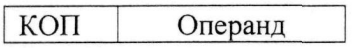
Фиг. 3.5. Двуадресна инструкция в SIMATIC S7-200, прехвърляща съдържанието на байт от адрес INA в адрес OUTA.

Съществуват и безадресни инструкции, които съдържат само код на операцията и се използуват за управление на програмата – фиг. 3.7.

```
BE
```

Фиг. 3.7. Безадресна инструкция в SIMATIC S7-300, задаваща край на блок.

Повечето от използуваните инструкции са едноадресни. Те съдържат код на операцията и един операнд – фиг. 3.6. С тях се минимизира обема на обектовия код. Това най-силно зависи от архитектурата на интерпретатора.




```
SWAP INA
```

Фиг. 3.6. Едноадресна инструкция в промишлен контролер SIMATIC S7-200, която разменя младшия и старшия байт на дума с адрес INA.

В общия случай задаването на операндите се извършва поотделно с формат и адрес. В повечето от контролерите обаче, форматът е общ за всички операнди и се задава чрез кода на операцията. В мнемониката на инструкциите форматът се указва с подходяща буква. Повечето от производителите са приели следните видове означения:

без означение – вит (bit);

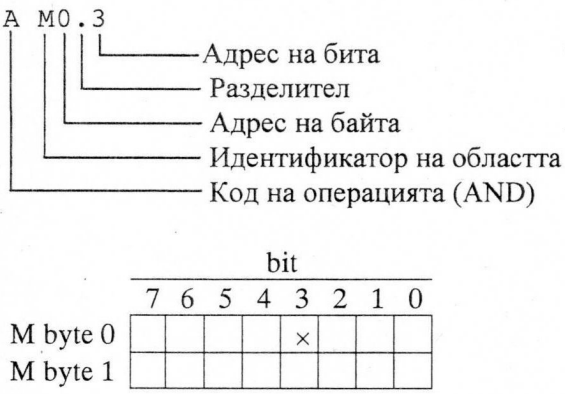
B – операндът е байт (byte);

W – операндът е двубайтова дума (word);

D – операндът е четирибайтова дълга дума (double word);

R – операндът е дробно число, записано в плаваща запетая (real).

Адресирането на операндите в различните промишлени контролери се извършва по възприет от фирмата производител начин. Например, при контролерите на SIMATIC адресирането на бит информация се извършва с указване на областта, номера на байта и номера на бита – вж. фиг. 3.5.



Фиг. 3.8. Неявно адресиране на операнди при SIMATIC.

Инструкциите на виртуалната микропроцесорна система се разделят по вид на пет групи:

1. Инструкции от контактен тип (контактни инструкции). Това са инструкции за обработка на отделен бит. С тях се обработва т.нар. „контактни" битове от данновата област, съответстващи на логически елементи с две състояния – включено и изключено. Такива са инструкциите за двоичните логически операции AND, OR, XOR и др. Към този тип инструкции спадат и инструкциите, чрез които се кодират операциите за логическо установяване Set и нулиране Reset на бит от данновата област.

2. Инструкции за работа с числови операнди, представени чрез байт, дума, дълга дума и числа в плаваща запетая. Делят се на три вида:

– математически инструкции, чрез които се извършват математически операции над операндите;

– логически инструкции, към които се причисляват операциите по преместване и ротиране на операндите;

– инструкции за преобразуване на типа на информацията, като например преобразуване от двоичен в двоично-десетичен вид и т.н.

3. Инструкции за сравняване, които сравняват операндите и изработват като изход еднобитов резултат. Резултатът от сравнението може да се използува като бит в изразите на контактните инструкции.

4. Инструкции за управление на хода на потребителската програма като условни разклонения, прескачане на отделни части от програмата, предаване на управлението, извикване на подпрограми и др.

5. Инструкции за ползване на системните функции, имитиращи работата на хардуерни устройства като таймери, броячи и др. Към тях се отнасят и инструкциите за управление на специализирани хардуерни блокове и др.

Виртуален процесор при промишлените контролери

Както беше посочено, инструкциите на междинния код са предназначени за обработване от виртуална микропроцесорна система. Интерпретаторът превежда тези инструкции на езика на реалната микропроцесорна система, т.е. той е този, който служи за връзка между виртуалната и реалната микропроцесорни системи. При тази концепция, потребителят няма достъп до реалната система. Той комуникира с виртуалната и трябва да познава само нейната структура.

Виртуалните микропроцесорни системи на различните промишлени контролери са сравнително близки помежду си и имат подобна идеология, което дава възможност за сравнително лесно усвояване на определен тип промишлен контролер.

Виртуалният процесор на микропроцесорната система участва в интерпретирането на програмата със своята виртуална структура. Виртуалната структура няма нищо общо със структурата на реалния процесор, но съдържа подобни блокове. Тя включва различни програмни модели, които отразява начина на създаване на отделните структурни елементи от потребителската програма. Различните модели взаимно се допълват и са предназначени за обработката на различен по вид информация. Съвременните промишлени контролери най-често ползуват два отделни програмни модела, условно наречени стеков и акумулаторен.

Стековият модел се използува при работа с променливи от типа бит, предимно при приложението на контактните инструкции. В общ преглед, това е най-разпространения модел, прилаган при организирането на логическите операции. На практика той представлява изграждане на еднобитово аритметично-логическо устройство чрез логически стек от типа LIFO. Фиг. 3.9. илюстрира логическия стек.

Ниво 0	RLO	Връх на стека
Ниво 1	S1	Клетка 1
Ниво 2	S2	Клетка 2
Ниво 3	S3	Клетка 3
Ниво 4	S4	Клетка 4
Ниво 5	S5	Клетка 5
Ниво 6	S6	Клетка 6
Ниво 7	S7	Клетка 7
Ниво 8	S8	Клетка 8

Фиг. 3.9. Логически стек.

Повечето от инструкциите, реализиращи двоични логически операции работят главно с върха на стека или с върха на стека и следващото ниво 1. Ниво 0 се нарича „Резултат от логическата операция" (RLO).

Класическият стеков модел при повечето промишлени контролери включва стек с широчина 1 бит, но някои контролери ползуват организация на логическия стек с широчина 1 байт.

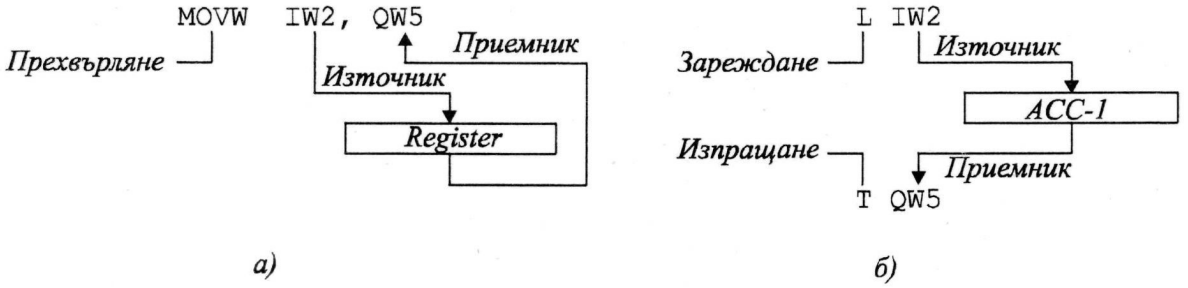
Пример за работа с логически стек е следващата програма за контролер SIMATIC S7-200, реализираща уравнението:

$$Q1.0 = (I0.1 + \overline{I1.1}) * (\overline{M1.2} + I1.2) * (T32 + \overline{I1.3})$$

```
LD   I0.1   Зарежда RLO с бита I0.1, стекът се измества надолу.
ON   I1.1   Операция ИЛИ между RLO и инверсията на бита I1.1, резултат – в RLO.
LDN  M1.2   Зарежда RLO с инверсията на бита M1.2, стекът се измества надолу.
ON   I1.2   Операция ИЛИ между RLO и бита I1.2. Резултат – в RLO.
ALD         Операция И между RLO и ниво 1, резултат в RLO, стекът се измества нагоре.
LD   T32    Зарежда RLO с инверсията на бита M1.2, стекът се измества надолу.
```

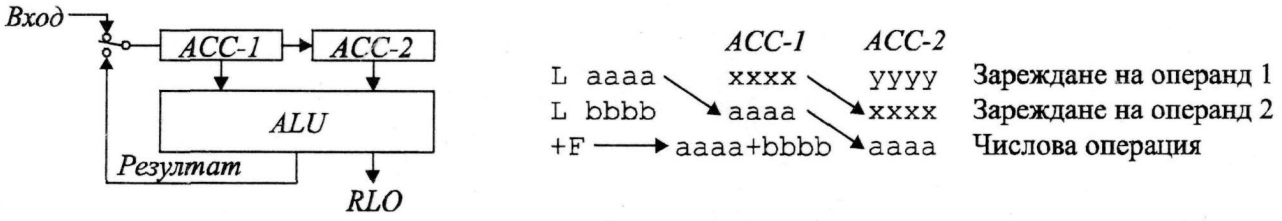
```
ON   I1.3   Операция ИЛИ между RLO и инверсията на бита I1.3. Резултат – в RLO.
=    Q1.0   Копира RLO в бита Q1.0
```

Акумулаторният модел се използува при числовите инструкции, които работят с променливи от типа байт, дума и двойна дума. Програмистът разполага с виртуално ALU, един или няколко потребителски виртуално достъпни регистри, наречени акумулатори (ACC). Някои акумулаторни модели използуват регистри за временно съхранение на информацията, които са недостъпни за потребителската програма. Класическият акумулаторен модел поема целия информационен поток. Фиг. 3.10 илюстрира този модел.



Фиг. 3.10. Акумулаторен модел за изпълнение на числови инструкции: а) – чрез потребителски недостъпен регистър; б) – чрез един виртуален акумулатор.

Често срещан и същевременно много удобен за работа е акумулаторния модел с два акумулатора – вж. фиг. 3.11. Двата достъпни за потребителя акумулатора се зареждат с операндите. Тяхното съдържание се изпраща в ALU, където се извършва указаната математическа операция, след което резултатът се връща в ACC-1.



Фиг. 3.11. Акумулаторен модел с два акумулатора за числови инструкции.

Както реалните микропроцесори, така виртуалните притежават дума на състоянието (Status word). Битовете в тази също виртуална дума представляват флагове, които се установяват съобразно резултата от изпълняваната инструкция. Флаговете се установяват най-вече от числовите инструкции. При инструкциите за сравняване се използуват различни методи за свързване на получения резултат с останалите видове инструкции. Като правило обаче, резултатът от операциите за сравняване се записва в RLO бита на логическия стек.

Флаговете във виртуалната дума на състоянията се ползуват като условие

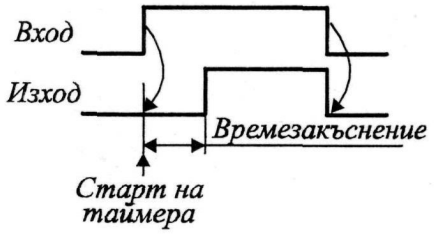
ИСТИНА или ЛЪЖА за реализиране на условни разклонения. Механизмът им обаче на ползуване е различен при различните контролери. По-голяма част от инструкциите за разклонения при промишлените контролери ползуват като условие за разклонение само RLO битът на логическия стек.

Една част от промишлените контролери ползуват само инструкции за разклонение по RLO бита. При тях, за да се реализира разклонение по битовете в думата на състоянието, е необходимо тези битове да бъдат обработени чрез контактните инструкции за формиране на RLO бит, т.е. допуска се побитова обработка с контактни инструкции на думата на състоянието.

При някои промишлени контролери не се допуска работа на контактните инструкции върху думата на състоянието. Тези контролери поддържат система от инструкции за условни преходи, директно зависещи от флаговите битове във виртуалната дума на състоянието.

Системни програмни и апаратно-програмни функции

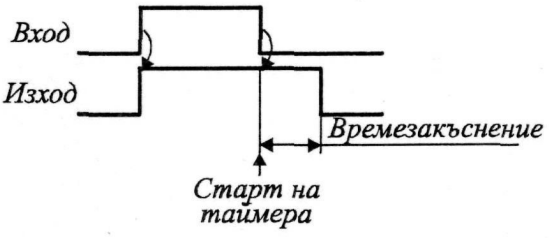
Основните системни функции, които се предлагат на оператора на промишлен контролер са таймерите и броячите. Таймерите са функции на операционната система, които симулират по програмен път апаратни устройства за времезакъснение, използувани в системите за управление.



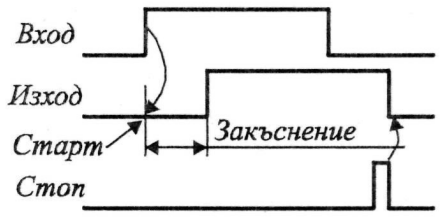
Фиг. 3.12. Таймер със закъснение при включване – TON, TIM, TMR.

Таймер със закъснение при включване – On-Delay Timer. Действието на този тип таймери е пояснено от фиг. 3.12. При активирането на входния сигнал, таймерът се стартира, изчаква зададеното закъснение и активира изхода си. Дезактивирането се извършва с дезактивирането на входа.

Таймер със закъснение при изключване – Off-Delay Timer. Този таймер действа по противоположен начин на предишния и неговото действие е показано на фиг. 3.13. При активирането на входния сигнал, таймерът включва изхода си. При дезактивирането на входния сигнал, таймерът се стартира и изключва изхода си след изтичане на зададеното времезакъснение.



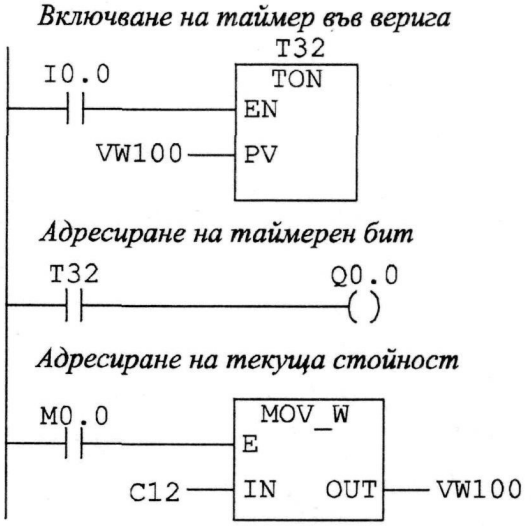
Фиг. 3.13. Таймер със закъснение при изключване – TOF.



Фиг. 3.14. Акумулаторен таймер – RTO, TMRA.

Задържащ (акумулаторен таймер) – този таймер има поне два входа. Таймерът се стартира при активиране на входния сигнал и след зададено закъснение включва изхода си. Изходът на таймера се изключва при пристигане на нулиращ сигнал – фиг. 3.14.

Основните параметри, които определят работата на таймера са стартиране и времезакъснение. Фиг. 3.15 илюстрира приложението на таймер.



Фиг. 3.15. Включване на таймер в управляваща програма.

Времезакъснението се определя от зададената стойност на PV по таймерната база (1ms, 10ms, 100ms, 1s). Обикновено таймерите се организират на базата на системно (недостъпно за потребителя) прекъсване. Както се вижда, всеки таймер се характеризира с текуща стойност и таймерен бит.

Съществуват таймери, които работят в режим на изваждане. При тях, предварително зададената стойност се зарежда в таймера в момента на стартиране, а таймерния бит се установява в 1, когато текущата стойност на таймера стане 0.

Таймерите се характеризират с две променливи от различен тип:

– текуща стойност (Current value), която представлява 16-разредна променлива от Integer тип;

– таймерен бит (Timer bit), който отразява състоянието на таймера и се установява в 1, когато текущата стойност на таймера е стане равна или по-голяма от предварително зададеното времезакъснение.

Достъпът до двата типа променливи се осигурява посредством използуване на съответния таймерен адрес, при което инструкциите с еднобитов операнд адресират таймерния бит, а инструкциите с операнд дума – текущата стойност.

Броячите са другите софтуерни функции в промишлените контролери, които емулират апаратни устройства за преброяване на събития. Под събитие се разбира преход от логическа нула в логическа единица на зададен информационен бит. Софтуерните броячни функции могат да емулират следните видове броячи:

– сумиращ брояч (CTU, CTN, C или CTR);

– изваждащ брояч (CTD);

– реверсивен брояч (UDC или CTUD).

Фиг. 3.16. Включване на реверсивен брояч в управляваща програма.

Работата на брояча се определя от броячен вход C (при реверсивните броячи входовете са два – за събиране CU и за изваждане CD), нулиращ вход R и зададена стойност PV. Подобно на таймерите, броячите се характеризират с две променливи от различен тип:

– текуща стойност (Current value), която представлява 16-разредна променлива от Integer тип. В нея се натрупват преброените събития;

– броячен бит (Counter bit), който отразява състоянието на брояча и се установява в 1, когато текущата стойност на брояча стане равна или по-голяма от предварително зададената стойност PV.

Фиг. 3.16 илюстрира приложението на реверсивен брояч. Трябва да се има предвид, в най-общия случай броячите могат да реагират на събития с честота поне два пъти по-ниска от честотата на оперативния цикъл. За преброяването на по-бързи събития някои фирми вграждат хардуерно изградени броячи, наречени бързи броячи (HSC или CTH).

Помощна част (помощни функции)

Помощните функции представляват набор от подпрограми, които извършват най-често срещаните математически функции (например sin, exp., log и др.). Те са вградени в операционната система и потребителят има достъп до тях чрез способа на извикване на подпрограми. При извикването им трябва да се спазват определените от производителя правила като начин на задаване на входните и изходните параметри, формата на данните и др.

4. ЕЗИЦИ ЗА ПРОГРАМИРАНЕ НА ПРОМИШЛЕНИТЕ КОНТРОЛЕРИ

Потребителските програми за системите за управление се създават на развойна станция. При програмируемите логически контролери се ползува понятието програмираща станция (развойна станция е по-широко понятие). Тя може да бъде персонален компютър с необходимото програмно осигуряване или специализиран фирмена програмираща станция. И в двата случая това са крос системи.

Класификация на езиците за програмиране на PLC се извършва според степента на детайлизация в процеса на програмиране. По този признак те се делят на:

а) езици от високо ниво:

– алгоритмични езици (C, PLM, BASIC, PASCAL);

– обектно-ориентирани езици (езици за проектиране на автоматизирани системи за управление), базиращи се на възможностите за представяне на алгоритъма на работа на логическо управляващо устройство във вид на ориентиран граф – GRAFSTEP на Telemecanique, HIGHGRAPH на Siemens, JARUS и т.н. Тези езици се ползват за програмиране на отделни части (обекти) от системите за управление. Те се предлагат от фирмите като надстройка на езиците за програмиране от ниско ниво.

б) езици от ниско ниво:

– езици на релейно-контакторни символи;

– езици но функционални електрически схеми;

– езици на логически булеви изрази;

– езици на мнемонични команди.

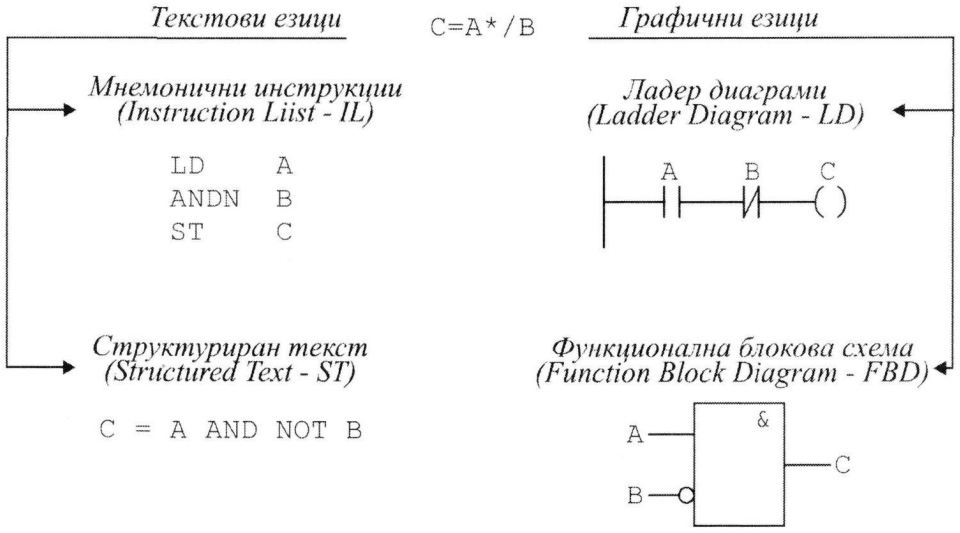
Понастоящем разнообразието на входни езици за програмиране на промишлени контролери е много голямо. Почти всяка фирма, производителка на PLC, предлага свой език за програмиране. Създаденият през 1993 г. стандарт IEC 1131 от някои производители на програмируеми логически контролери (Allen Bradley, Eberle, Kloeckler-Moeler, Omron, Pepper & Fuchs, Shettcontrol, Telemecanique и др.) е стремеж за уеднаквяване на езиците за програмиране.

Стандарт за програмни езици за PLC – IEC 1131/1993

Стандартът IEC 1131 дефинира синтаксиса, семантиката и различните приложения на езиците за програмиране в системите за индустриална автоматизация, независимо от базовата структура на хардуера и вградената операционна система на потребителя. С това се създава възможност за използуване на веднъж научения език с широк спектър от PLC системи, базирани на стандарта.

Класификация на езиците за програмиране на PLC според стандарта

IEC 1131 е даден на фиг. 4.1.



Фиг. 4.1. Дефинирани от стандарта IEC 1131/93 езици за програмиране на PLC.

Езиците за програмиране са взаимосвързани, т.е. дава се възможност за автоматично транслиране от един вид в друг.

Ладер диаграми (Ladder Diagram – LD)

Ладер диаграмите са графичен език за програмиране на PLC, чието начало е поставено от първите производители на програмируеми контролери в САЩ. Този език "копира" електрическата схема на изгражданите на базата на релейно-контакторни схеми устройства за управление. Ладер диаграмата представлява графично описание на логически инструкции, в които булевите променливи са изразени чрез "контакти". Състоянията на отделните битове се определя от две основни инструкции –"изпълни при затваряне" (examine if close –| |–) и "изпълни при отваряне" (examine if open –|/|–). Тези две инструкции напълно съответстват на релейните схеми "нормално отворен контакт" и "нормално затворен контакт". Всъщност те изразяват по какъв начин участват булевите променливи в логическото уравнение – с правата или с инверсната си стойност.

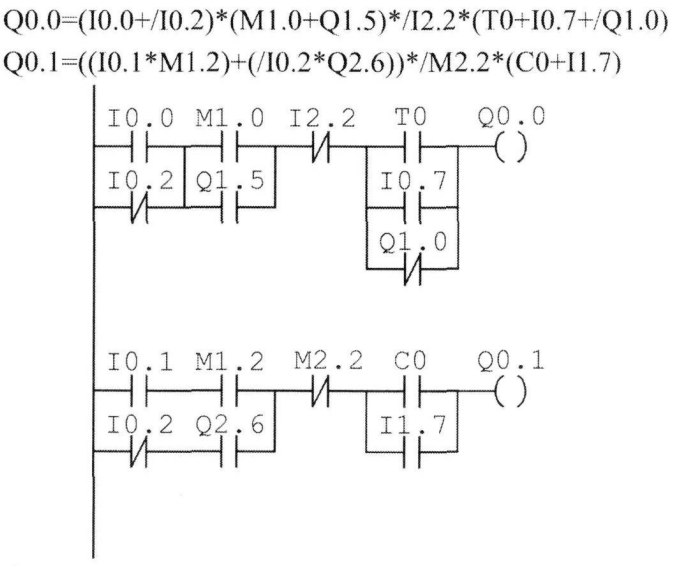
В релейно-контакторните схеми, управлението на всеки изпълнителен механизъм се извършва от отделна електрическа схема, започваща със захранваща линия, преминаваща през паралелно или последователно свързани контакти (в съответствие с логическата функция на управлението) и завършват с т.нар. "бобина". Когато състоянията на включените във веригата контакти позволят протичането на ток към бобината, изпълнителният механизъм се активира.

По аналогия с релейно-контакторните схеми, всяко логическо уравнение се представя чрез последователни или паралелни връзки между логическите променливи ("контакти") и формира т.нар. "стъпало" (contact network) в ладер диаграмата. Началото на всяко стъпало започва с условна "захранваща линия"

(power line). Резултантното състояние на всеки изходен (или междинен) бит се представя чрез "бобина" (coil –( )), чийто логически смисъл е присвояване на стойност.

Отделните стъпала в ладер диаграмите се обработват от операционната система на промишления контролер последователно отгоре надолу.

Логическата комбинация между отделните контакти в ладер диаграмите се нарича "тестова зона" или "логически условия", а участващите контактни инструкции – "входни инструкции". На фиг. 4.2 е показан пример на реализиране на две уравнения като стъпала в ладер диаграма.



Фиг. 4.2. Ладер диаграма.

Резултатът от логическата операция може да служи като условие за извършването на определена логическа операция. За тези цели се ползват т.нар. "специални бобини" (special coil –(xxx)). Те могат да бъдат:

1. Функции на схемите:

– установяване на бит – Set –(S) ;

– нулиране на бит – Reset –(R) .

2. Функции за организиране на програмата:

– безусловен преход към етикет – Jump to Label –(J) ;

– извикване на подпрограма с номер n – Call Subroutine –(Call n) ;

– връщане от подпрограма – Return from Subroutine –(Ret) ;

– край на потребителска програма – End of Program –(END) .

Стандартът IEC1131 позволява в качеството на "бобина" да се таксува много по-общи неща, затова и условната "бобина" се нарича още "изходна инструкция", което означава, че в зависимост от резултата на изпълнението на логическото уравнение могат да се изпълняват значително по-сложни инструкции, като например "бокс инструкции" (box).

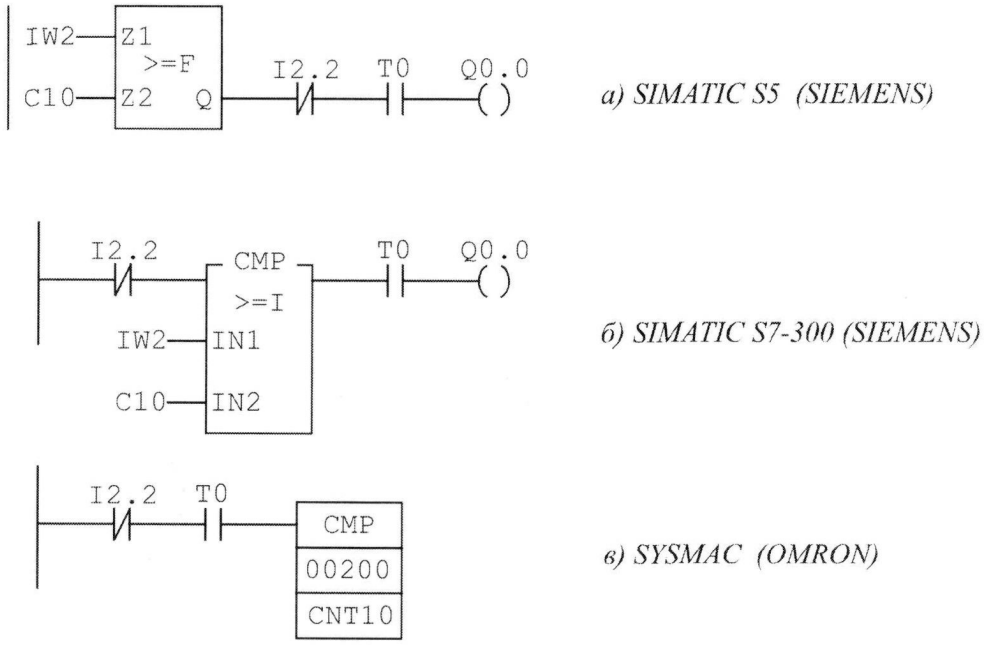
Бокс инструкциите съчетават действието на контактните и числовите инст-

рукции. Те се разделят на две основни групи:

– бокс изходни инструкции за системни функции (system function box), които са вградени в операционната система, като таймери, броячи и др.

– бокс изходни функции за операционни символи (operational box), които представляват логически и математически операции над числови операнди, както и операции за прехвърляне на променливи в различните даннови области.

Нормално бокс инструкциите могат да се използват само като изходни инструкции. Изключение е бокс инструкцията за сравнение (comparison box instruction), която извършва операция над числови операнди и формира двоичен (булев) резултат – вж. фиг. 4.3.



а) SIMATIC S5 (SIEMENS)

б) SIMATIC S7-300 (SIEMENS)

в) SYSMAC (OMRON)

Фиг. 4.3. Различни случаи на приложение на бокс-инструкция за сравнение.

В зависимост от избраната стратегия за програмиране при различните промишлени контролери, тази инструкция може да се намира на три различни места в ладер диаграмите:

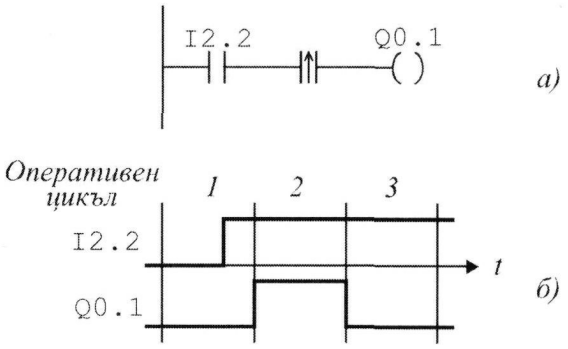
а) в началото на ладер диаграмата, непосредствено след захранващата линия. В този случай резултатът от сравнението се възприема като първа контактна инструкция и участва в по-нататъшното сканиране на логическото уравнение от стъпалото;

б) на всяка позиция от текстовата зона. Резултатът от изпълнението на инструкцията за сравнение е еквивалентен на резултата от сканирането на контактна инструкция;

в) в края на ладер диаграмата, като инструкция от типа на "специална бобина". В този случай логическото уравнение в стъпалото служи като разрешение за сравнение и резултатът от сравнението се записва в определени флагове

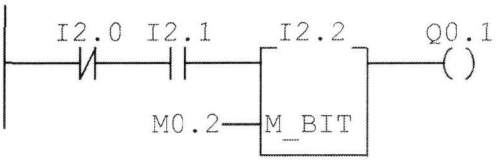
на системната област, които могат да се сканират като контактни инструкции в следващи стъпала на ладер диаграмата.

Особеностите на работа на промишлените контролери изискват използването на специални инструкции за отчитането на преход (фронт на импулс) – съответно нарастващ фронт (rising edge – |↑|) и спадащ фронт (falling edge – |↓|). Пример за такава инструкция в ладер диаграмите е показана на фиг. 4.4. Инструкцията е безадресна, т.е. отнася се към целия резултат от решението на ладер диаграмата преди нея. Преходът се отчита в резултата на текущия оперативен цикъл спрямо този в предишния. Ако е необходимо да се отчете преход в единичен контакт, необходимо е контактът да се постави в началото на стъпалото и веднага да следва инструкцията за преход.



Фиг. 4.4. Приложение на инструкция за преход: а) – включване в ладер диаграма; б) – въздействие.

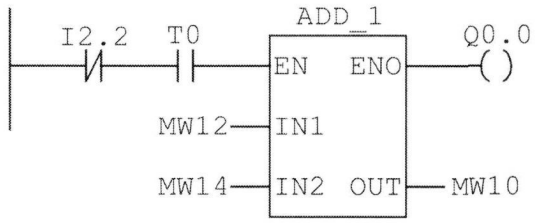
За отчитане на преход в единичен контакт вътре в стъпало на ладер диаграма, някои промишлени контролери притежават специални инструкции. Пример е показан на фиг. 4.5 (SIMATIC S7-300), където вътрешно реле M0.2 запомня положителен преход на входния бит I2.2 в предходния оперативен цикъл и го прилага в текущия.



Фиг. 4.5. Бокс-инструкция за отчитане на преход на единичен контакт.

Стандартът IEC 1131 предвижда включването в стъпалата на ладер диаграмите на функции и на функционални блокове. Ако блокът притежава вход за разрешаване Enable – EN (разрешение на изпълнение на блока при логическо условие RLO = 1), то включването на блока може да се извърши като бокс-инструкция от типа на специалните бобини. Ако блокът притежава изход от типа бит, който се установява в 1 или 0, в зависимост от изпълнението на програмираната в него функция, този изход може да се използува като разрешение Enable Output – ENO за понататъшното сканиране на стъпалото. В този случай блокът може да се включи в тестовата зона на стъпало от ладер диаграма. Ако функционален блок не притежава входове EN или ENO, неговото изпълнение може да се организира чрез извикването му като подпрограма (Call Subroutine).

По същия механизъм могат да се включват и числови инструкции в тестовата зона на ладер диаграмите, което увеличава възможностите за съчетаване на числови и контактни инструкции.



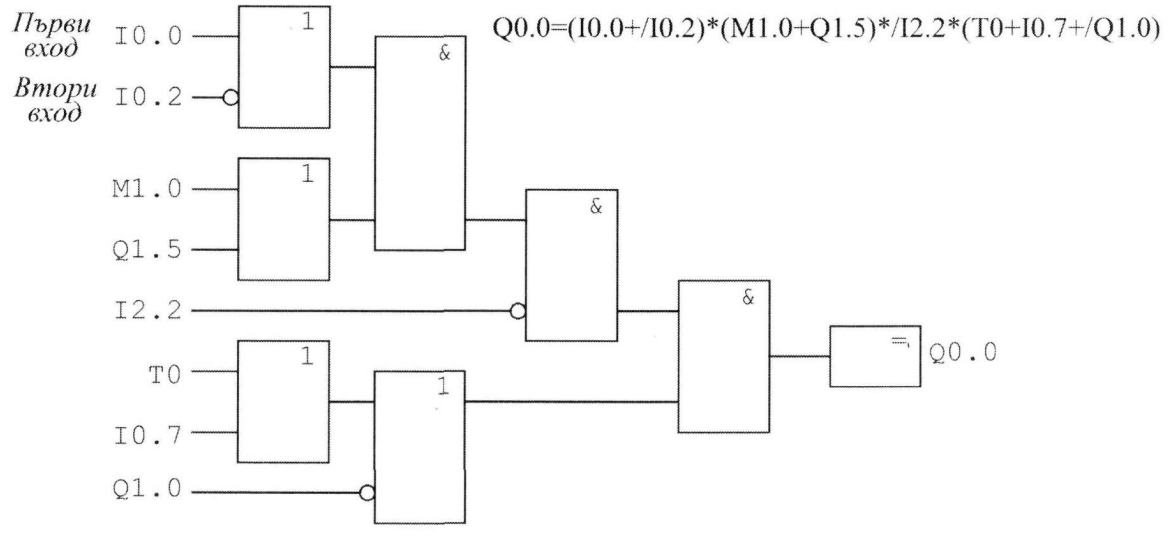
Фиг. 4.6. Включване на числова инструкция в тестова зона на ладер диаграма

Пример за включване на числова инструкция в тестова зона е даден на фиг. 4.6. за SIMATIC S7-300 където, при изпълнението на условието от предходната част на уравнението, се извършва събиране на две числа, резултатът се записва като числова променлива и ако събирането е извършено без грешка (няма препълване) се активира изходната бобина.

Функционални блокови схеми (Function Diagram Block – FDB)

Езикът на функционалните блокови схеми е графичен език за програмиране на промишлени контролери, използуващ логически елементи (Logic boxes) за представяне на управляващите уравнения. Началото му е поставено от европейските производители на PLC.

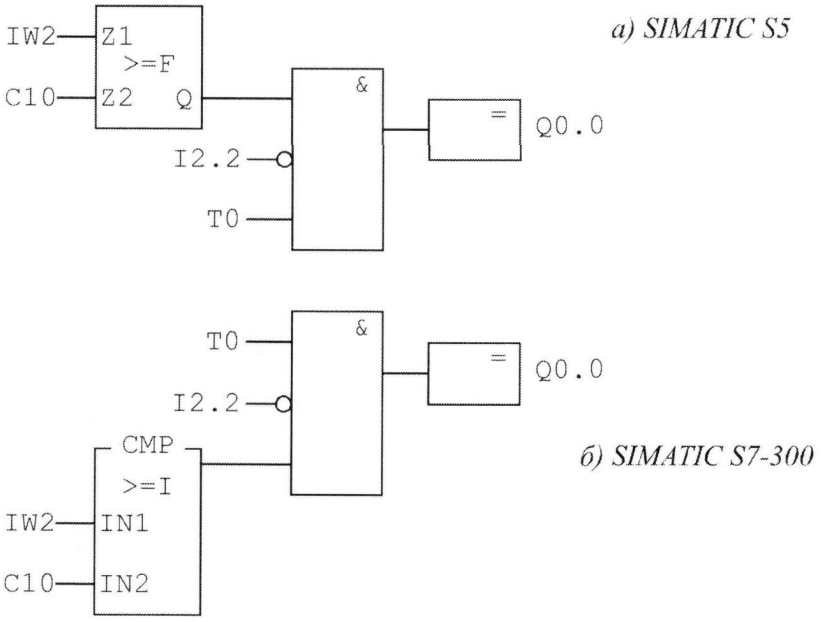
Всяко логическо уравнение се кодира самостоятелно и се оформя като отделен логически сегмент (Logical Segment). Уравнението се кодира чрез последователност от логически инструкции, всяка от които е представена чрез логически елемент. Използуваните логически елементи при езикът на функционалните блокови схеми са И, ИЛИ, ИЗКЛЮЧВАЩО ИЛИ (AND, OR, XOR) и техните комбинации, а графичните символи и означения са заимствани от логическата символика според стандарта IEEE Std. 91-1984. Като пример на фиг. 4.7 е представено кодирането на логическо уравнение (същото както първото уравнение в ладер диаграмата от фиг. 4.2) чрез функционална блокова схема.



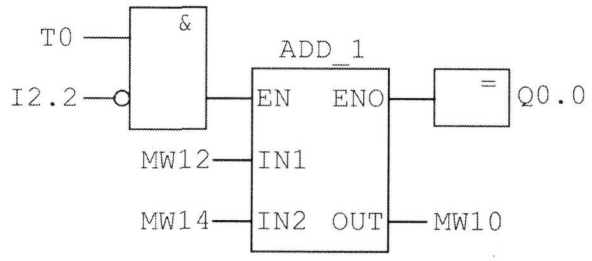
Фиг. 4.7. Кодиране на уравнение чрез функционална блокова схема.

Един логически сегмент от функционалната блокова схема съответства на едно стъпало от еквивалентна ладер диаграма. Последователността на обработка на уравнението в логическия сегмент е отляво надясно и отгоре надолу. Например първият логически елемент от представеният сегмент има два входа. Първият вход зарежда RLO бита със стойността на променливата I0.0. Извършва се логическата функция между инверсната стойност на втория вход (променливата I0.2). Резултатът се съхранява временно в RLO бита и се предава за обработка към следващия логически елемент.

Логическите инструкции при функционалната блокова схема се наричат "двоични логически инструкции" (Binary Logic Instruction). Те съответстват на контактните инструкции при ладер диаграмите. Еквивалент на бобина е инструкцията за присвояване (Assignment, =).



Фиг. 4.8. Включване на инструкции за сравнение във функционална блокова схема.



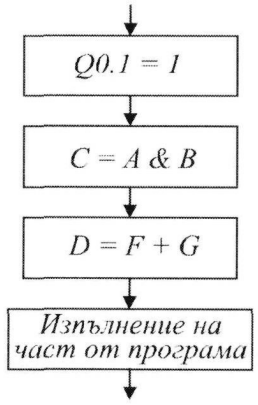
Фиг. 4.9. Включване на функционален блок за събиране във FDB.

Подобно на ладер диаграмите, при функционалните блокови схеми също могат да се включват бокс-инструкции. Фиг. 4.8 илюстрира приложението на бокс-инструкцията за сравнение приложена в началото и на произволно място в логическия сегмент, а фиг. 4.9 – функционален блок за събиране, приложен като бокс-инструкция.

Структуриран текст (Structured Text – ST)

Езикът на структурирания текст е представител на алгоритмичните езици. Той се базира на възможността за представяне работата на системата за управление чрез блокова схема на алгоритъма за управление (Flowchart). Той най-добре и лесно работи с логически и аритметични операции, операции за обработка на данни, както и осигурява лесен достъп до всички адресируеми области от промишления контролер.

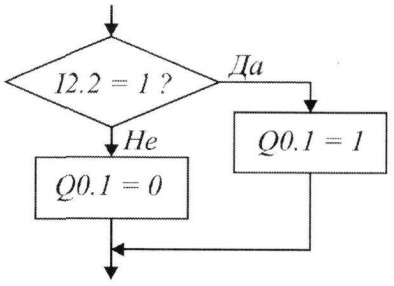
Отделните елементи от блоковата схема на алгоритъма на работа се кодират директно с инструкции, представляващи текстови изрази (Literal Phrases). Текстовите изрази са три типа – безусловен израз, условен израз и цикъл.




```
Set Q0.1;
C := A&B;
D := F+G;
Exec FB10;
```

Фиг. 4.10. Безусловен израз в структуриран текст.

Безусловният израз (Action Phrase) позволява извършването на едно или няколко действия върху различен тип обекти. Отделните действия могат да бъдат описани с един оператор, а може да се приложи и съставен оператор (верига от оператори). Безусловните изрази се прилагат за описание на линеен участък от алгоритмичната блокова схема. Пример за безусловен израз с верига от оператори е показан на фиг. 4.10. При промишлените контролери безусловните изрази, които адресират данновите области, използват мнемонични означения на съответните инструкции.

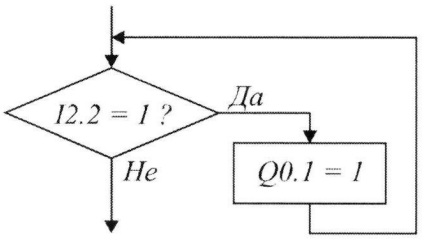



```
If (I2.2=1)
  Then Set Q0.1;
  Else Reset Q0.1;
```

Фиг. 4.11. Условен израз за структуриран текст.

Условните изрази (Condition Phrase) позволяват извършването на действия в зависимост от резултата от определена проверка. Условните изрази се реализират чрез класическата If, Then, Else конструкция – фиг. 4.11.

Циклите (Iterative Phrase) позволяват непрекъснато изпълнение и многократно повторение на действия, до изпълняване на определено условие. За организирането на цикли се използват класическите `For ... Next` и `While ... Do` конструкции от алгоритмичните езици – пример на фиг. 4.12.




```
While (I2.2=1)
  Do Set Q0.1;
```

Фиг. 4.12. Обособяване на цикли в структуриран текст.

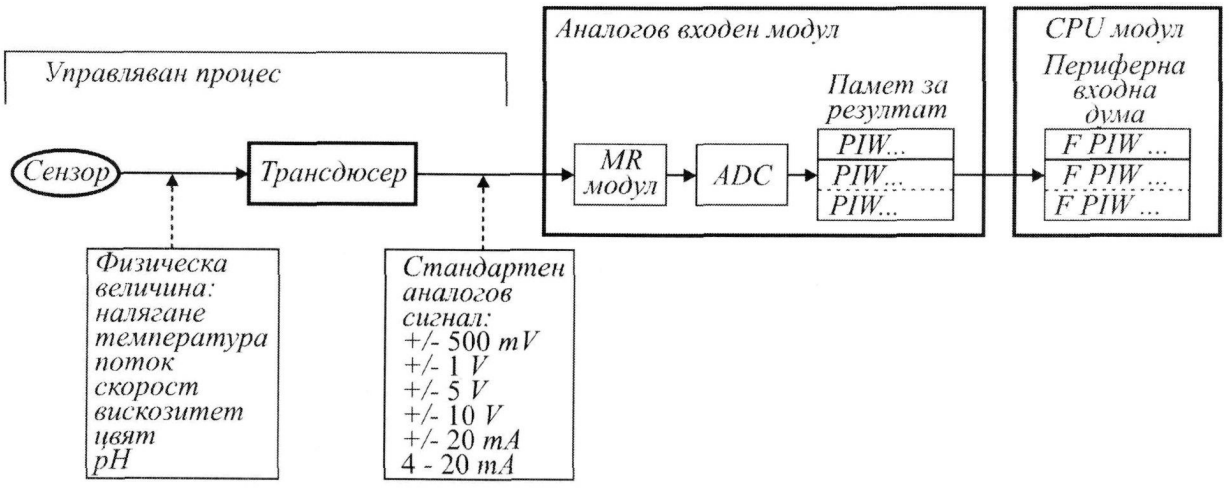
Прилагането на структурирания текст съкращава значително времето за програмиране, в сравнение с използването на мнемоничните инструкции, ладер диаграмите и функционалните блокови схеми. Като недостатъци могат да се изтъкнат по-големия обем на изпълнимия код, получен след компилирането на програмата, както и затрудненатa настройка и диагностика на работната програма.

5. ВЪВЕЖДАНЕ И ИЗВЕЖДАНЕ НА АНАЛОГОВА ИНФОРМАЦИЯ В ПРОМИШЛЕНИТЕ КОНТРОЛЕРИ

Въвеждане на аналогова информация

Във всеки производствен процес има известен брой физични величини, които трябва да се обработят за целите на автоматизацията. Това са налягане, температура, скорост, ротационна скорост, pH стойност, вискозитет и др.

Приложението на аналоговите входове в промишлените контролери може да се види от фиг. 5.1.



Фиг. 5.1. Възприемане на аналогова информация от промишлен контролер.

Сензор (Възприемател) – възприема и реагира на промени в измерваната физическа величина: налягане; сила; ъглово завъртане; електрическа проводимост и др.

Трансдюсер (Измервателен преобразувател) – трансформира промените от сензора в стандартни аналогови електрически сигнали, като например: ±500 *mV*, ±10 *V*, ±20 *mA*, 4 ÷ 20 *mA* и др. Тези сигнали се подават на аналоговите входни модули.

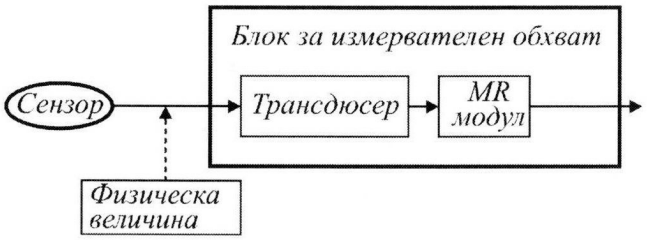
Аналогов входен модул

Аналоговият модул преобразува аналоговата стойност от измервателния преобразувател и я запомня в цифров вид в буферна памет. Нормално аналоговия модул се състои то три блока:

Модул за измервателен обхват (Measured Range Module) – това е блок, който преобразува входящата в аналоговия модул стандартна електрическа величина в аналогова величина, съответстваща на входа и обхвата на използвания

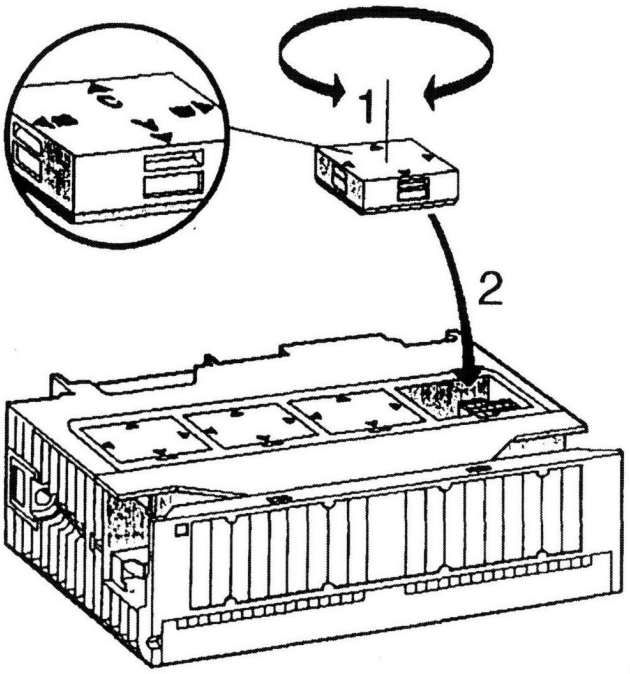
аналогово-цифров преобразувател. Това е един от важните блокове в аналоговия модул – т.нар. ***Нормиращ преобразувател***.

Някои модули за измервателен обхват включват в себе си и измервателен преобразувател, позволяващ им директно да поемат сигнал от сензор. Обикновено това са преобразуватели за най-често срещаните аналогови величини в производствените процеси – съпротивление (обхват 150 Ω, 300 Ω, 600 Ω и т.н.) и температура (термоелементи, Pt100, Ni100 и др.). Блоковата схема на такъв блок за измервателен обхват е показана на фиг. 5.2.



Фиг. 5.2. Блок за измервателен обхват, съдържащ вграден трансдюсер.

Аналогов модул с модул за измервателен обхват от серията S7-300 е показан на фиг. 5.3.



Фиг. 5.3. Модули за измервателен обхват.

В модула за измервателен обхват се съдържат кодиращи ключета за задаване на обхвата. Настройките за различните типове на измерване и измервателни обхвати са фиксирани. На някои модули, няколко канала са групирани заедно (2, 4 или 8). В този случай, настройката на кодиращите ключета се прилага за

цялата група входове.

Аналоговите модули без кодиращи ключета имат различни изводи за измерване на напрежение и ток. Типът на измерване може да се укаже като се опроводи съответният извод.

Аналогово-цифров преобразувател (ADC) – Преди да се обработят в CPU, аналоговите сигнали се преобразуват в цифров вид. Използват се аналогово-цифрови преобразуватели, работещи на тегловния принцип (10-12 разреда) или от интегриращ тип (12-14 разреда).

Цифров буфер (Result Memory) – Резултатът от преобразуването се съхранява в буфера на аналоговия модул и остава там, докато не се презапише с нова стойност. При S7-300 преобразуваната аналогова стойност може да се прочете с инструкцията "L PIW...".

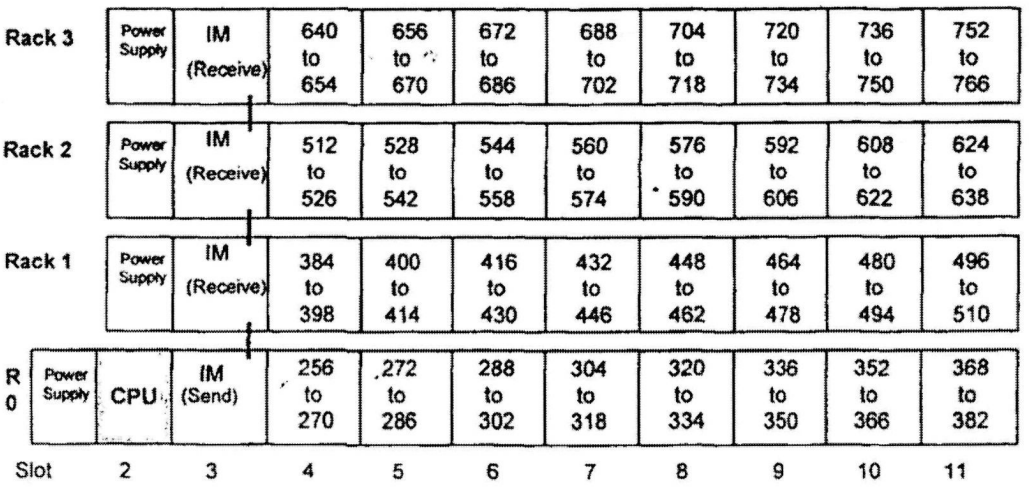
Сканиращ цикъл – Аналоговите модули имат собствен сканиращ цикъл. Времето за сканиране е времето, необходимо на модула да обработи по веднъж всички активирани входове. Времето за сканиране на аналоговите входове може да се задава. В края на сканиращия цикъл модулът може да стартира хардуерно прекъсване (в случай, че е избрано време на сканиране по-голямо от 1 ms). За намаляване на собственото време на сканиране, неизползваните входове трябва да се дезактивират софтуерно и да се свържат хардуерно накъсо.

Управление на времето на сканиране може да се извърши само при ползването на аналогово-цифрови преобразуватели от интегриращ тип. То се извършва чрез намаляване на времето на интегриране и директно рефлектира върху точността на преобразуване. Следната таблица за сигналния модул SM331 показва връзката между времето на интегриране, разрешаващата способност и подтискане на нискочестотни смущения.

Време на интегриране (ms)	Разрешаваща способност (bits)	Подтискане на нискочестотни смущения(Hz)
2.5	9 + знаков бит	400
16.6	12 + знаков бит	60
20	12 + знаков бит	50
100	14 + знаков бит	10

Образ на аналоговите входове в паметта на промишления контролер – Всеки аналогов вход разполага с определени клетки в паметта на контролера. Входните аналогови клетки се обновяват от изходния буфер на аналоговия входен модул по време на фазата "сканиране на входовете" на оперативния цикъл. Ако времето на собственото сканиране на аналоговия входен модул е по-голямо от това на оперативния цикъл на контролера, обновяването се извършва със старите стойности. Възможно е обновяването да се организира чрез прекъсване от страна на аналоговия модул и да се изключи от фазата "сканиране на входовете".

Адресна област в CPU модула – S7-300 има специална адресна област за аналоговите входове и изходи, която е отделна от адресната област на цифровите входно изходни модули (PII/PIQ). Тази адресна област се разполага от байт 256 до байт 767. Всеки аналогов канал заема 2 байта. Достъпът до аналоговите модули се осъществява чрез инструкциите Load и Transfer. Например инструкцията "L PIW256" прочита първия канал на първия модул в шина 0. При S7-400, адресната област за аналоговите модули започва от байт 512 и зависи от позицията на модула в монтажния ред.



Фиг. 5.4. Адресиране на аналоговите модули в S7-300.

Аналоговите стойности от измерването в се представят като Integer в допълнителен код. Ако бит No.15=0 стойността е положителна, а ако бит No.15=1 е отрицателна. Когато разрешаващата способност на аналогов модул е помалка от 15 бита, аналоговата стойност се записва в акумулатора ляво-

ориентирана. Неизползваните по-малко значещи битови позиции се запълват с "0".

При аналоговите модули на S7-300 разрешаващата способност се дефинира косвено чрез избиране време на интегриране в инструмента "HWConfig".

В зависимост от типа на модула са възможни от 8 до 15 битови A/D (D/A)преобразуватели. Времето на преобразуване зависи от използваната процедура на преобразуване в модула (процедура на интегриране, последователна апроксимация). Например модулът SM344 има време на преобразуване 5 *ms* за всичките 4 входни канала.

Представяне на аналоговата стойност в различните обхвати

Обхват	Напрежение		Ток		Съпротивление		Температура напр. Pt100	
	Изм.обхват ± 10V	Единици	Изм.обхват 4 .. 20mA	Единици	Изм.обхват 0...300Ohm	Единици	Изм.обхват -200...+850°C	Единици
Препълване	>= 11.759	32767	>= 22.815	32767	>=352.778	32767	>= 1000.1	32767
Разширен обхват	11.7589 ⋮ 10.0004	32511 ⋮ 27649	22.810 ⋮ 20.0005	32511 ⋮ 27649	352.767 ⋮ 300.011	32511 ⋮ 27649	1000.0 ⋮ 850.1	10000 ⋮ 8501
Номинален обхват	10.00 7.50 ⋮ -7.5 -10.00	27648 20736 ⋮ -20736 -27648	20.000 16.000 ⋮ 4.000	27648 20736 ⋮ 0	300.000 225.000 ⋮ 0.000	27648 20736 ⋮ 0	850.0 ⋮ -200.0	8500 ⋮ -2000
Разширен обхват	- 10.0004 ⋮ - 11.759	- 27649 ⋮ - 32512	3.9995 ⋮ 1.1852	- 1 ⋮ - 4864	Отрицат. стойности не са възможни	- 1 ⋮ - 4864	- 200.1 ⋮ - 243.0	- 2001 ⋮ - 2430
Препълване	<= - 11.76	- 32768	<= 1.1845	- 32768		- 32768	<= - 243.1	- 32768

Кодирането на симетричните напреженови или токови обхвати се извършва в обхвата от -27648 до +27648. Валидни са следните симетрични обхвати: ±80mV; ±250mV; ±500mV; ±1 V; ±2.5V; ±5V; ±10V; ± 3.2 mA; ±10mA; ±20mA.

Кодирането на асиметричните напреженови или токови обхвати се извършва в обхвата от 0 до +27648. Валидни са следните асиметрични обхвати: 0 до 2V; 1 до 5 V; 0 до 20 mA; 4 до 20 mA.

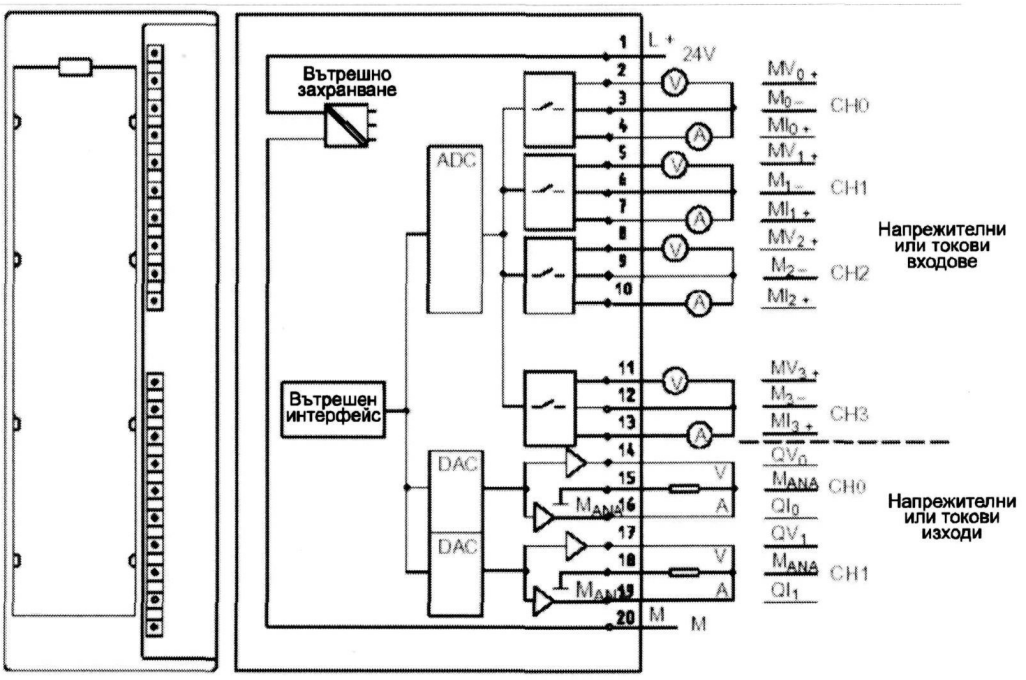
Кодирането на обхватите на съпротивлението се извършва в обхвата от 0 до +27648. Обхватите са: 0 до 150 Ohm; 0 до 300 Ohm; 0 до 600 Ohm.

Температурите се измерват със съпротивителни термометри или термодвойки. Кодирането на температурния обхват се извършва след десетократно разширяване на обхвата:

Сензор:	Температурен обхват:	Обхват след разширяването:
Pt 100	от -200 до +850°C	от -2000 до +8500
Ni 100	от -60 до +250°C	от -600 до +2500
Термодвойка тип K	от -270 до +1372 °C	от -2700 до +13720
Термодвойка тип N	от -270 до +1300 °C	от -2700 до +13000

Термодвойка тип J	от -210 до +1200 °C	от -2100 до +12000
Термодвойка тип E	от -270 до +1000 °C	от -2700 до +10000.

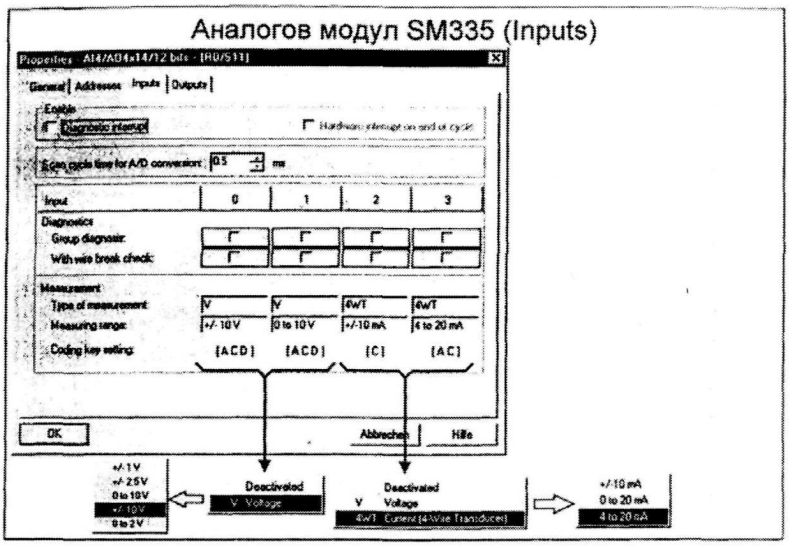
Аналогови сигнални модули при S7-300



Фиг. 8. Сигнален модул SM 334 AI 4/AO 2 x 8/8 bit

Програмирането на аналоговите сигнални модули се извършва в средата на Step-7.

Аналогови входове на сигнален модул при S7335



Времето за сканиране е времето необходимо на модула да обработи всички активирани аналогови входове веднъж. Настройката на времето за сканиране на A/D преобразуването може да бъде от 0,5 *ms* до 16 *ms*. Когато обработи всички аналогови входове, модулът може да стартира хардуерно прекъсване (само ако е избрано време за сканиране по-голямо от 1 *ms*).

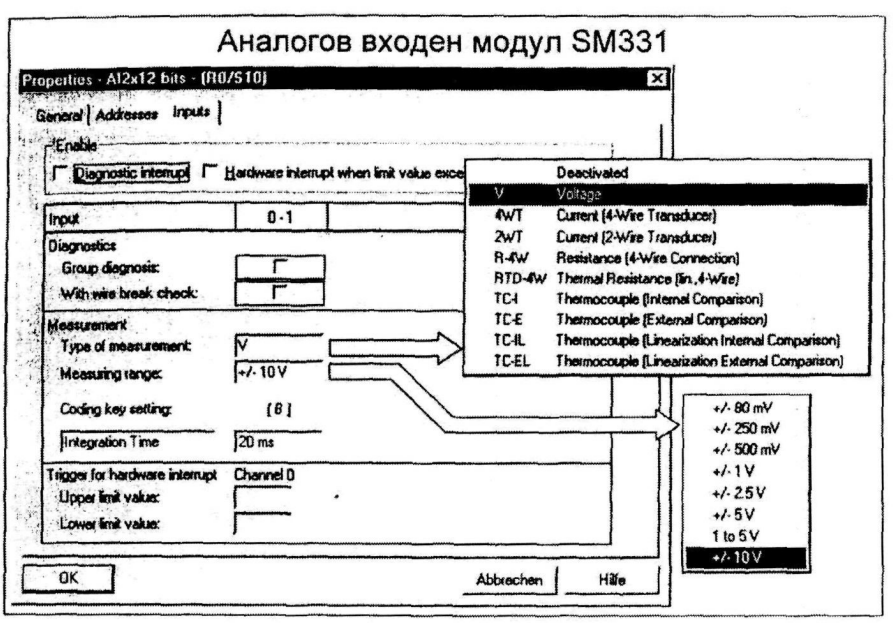
Неизползваните входове трябва да се свържат хардуерно накъсо и да се деактивират софтуерно. Деактивираните аналогови входове намаляват времето за сканиране.

Когато се избере типът на измерване и измервателния обхват се показва необходимата настройка на кодиращото ключе на модула с измервателен обхват.

Аналоговите входове на SM 335 имат разрешаваща способност от 13 бита + знак, а аналоговите изходи 11 бита + знак.

Когато е активирана опцията за диагностично прекъсване и възникне хардуерна грешка, като напр., прекъсване на захранване се изпълнява диагностично прекъсване (OB 81). Допълнително, в полето "Diagnostics" трябва да се посочи кои входове се наблюдават. Възможно е да се провери прекъсване на кабел само при входове със сигнал 4 до 20 mA.

Аналогови изходи на сигнален модул при S7-335



Параметри — За аналоговите входни модули могат да бъдат установени две групи параметри чрез инструмента "HW Config":

Диагностично прекъсване — *Diagnostic Interrupt (за модула като цяло): Ако е активирано полето "Group Diagnosis" и се появи диагностично събитие, съответната информация се въвежда в областта диагностични данни на модула и се стартира диагностично прекъсва-

не (OB82). Аналоговият модул може да регистрира следните диагностични събития:

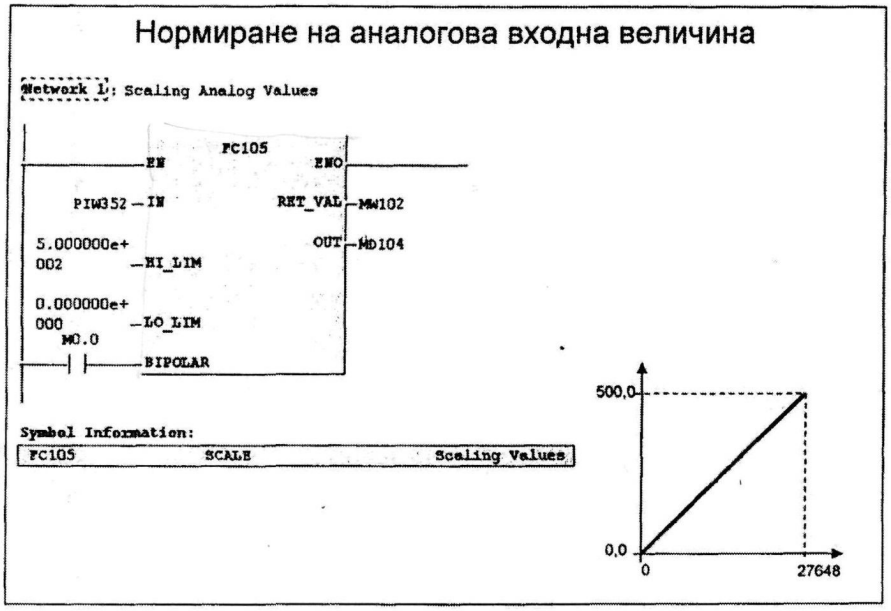
- Грешки при конфигуриране/присвояване на параметри
- Прекъсване на кабел (ако е активирано полето "With Wire Break Check")
- Сигнал над измервателния обхват
- Сигнал под измервателния обхват
- Липса на захранващо напрежение L+

Хардуерно прекъсване при превишаване на гранична стойност *Hardware Interrupt when Limit Value Exceeded. Ако входната стойност превиши обхвата зададен чрез полетата "Upper Limit Value" и "Lower Limit Value", модулът стартира хардуерно прекъсване. Забележка: Само сигнала на първия канал в дадена група може да бъде наблюдаван за превишаване на зададените гранични стойности!

Вид измерване *Type of Measurement: При активиране на полето се показват възможните типове на измерване (voltage, current...). За неизползваните канали трябва да се избере опцията "deactivated" и хардуерно да се свържат към нулата на модула.

Измервателен обхват *Measuring Range: При активиране на полето се показват възможните измервателни обхвати за избрания тип на измерване.

Нормиране на аналогова входна величина

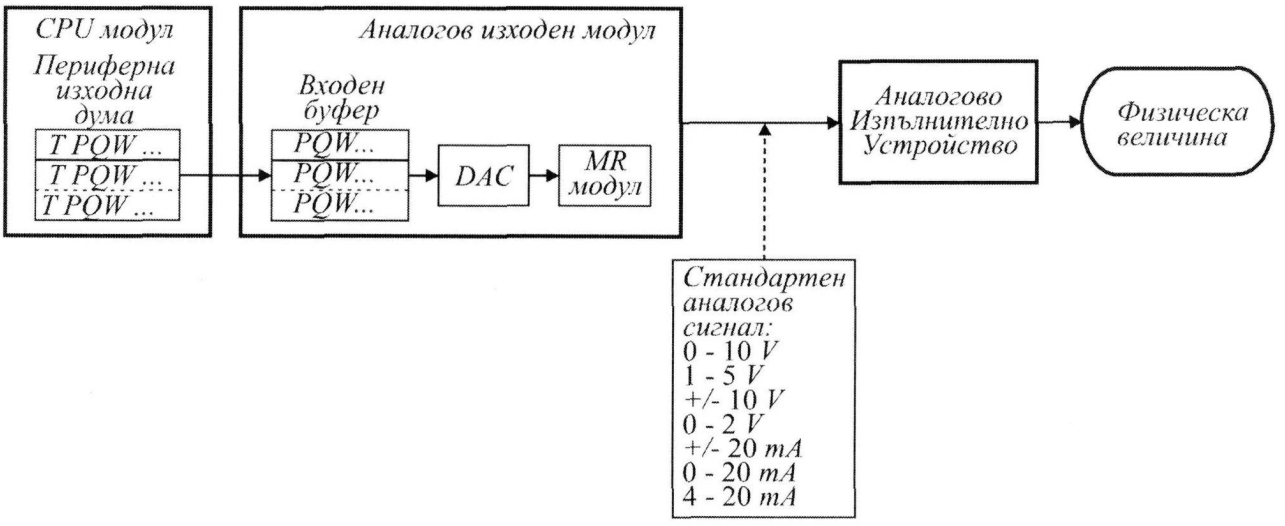


Аналоговият модул кодира аналоговата стойност 10 V като целочислената стойност 27648. Тази стойност трябва да се преобразува в физичната величина

"литри". Това преобразуване е известно като "мащабиране" или "нормиране" на аналоговата стойност. Програмно мащабирането се извършва чрез стандартният блок FC 105 в Step-7.

Аналоговата стойност на вход IN може да се прочете директно от модула или от област данни в INTEGER формат. Входовете LO_LIM (ниска граница) и HI_LIM (висока граница) се използват за дефиниране границите на преобразуване. В примера прочетената стойност се мащабира в обхвата от 0 до 500 литра. Мащабираната стойност (физическо количество) се записва като реално число на изхода OUT. Входът BIPOLAR определя, дали трябва да се преобразуват и отрицателни стойности. В горния пример, флаговият бит M0.0 има състояние "0", което указва, че входната стойност е униполярна. Изходът RET_VAL има стойност "0", ако функцията е изпълнена без грешки.

Извеждане на аналогова величина



Представяне на аналоговата стойност за аналогов изход

Обхват	Единици	Напрежение			Ток		
		Изходни обхвати:			Изходни обхвати :		
		0 to 10V	1 to 5V	± 10V	0 to 20mA	4 to 20mA	± 20mA
Препълване	>=32767	0	0	0	0	0	0
Разширен обхват	32511 : 27649	11.7589 : 10.0004	5.8794 : 5.0002	11.7589 : 10.0004	23.515 : 20.0007	22.81 : 20.005	23.515 : 20.0007
Номинален обхват	27648 : 0 : - 6912 - 6913 : - 27648	10.0000 : 0	5.0000 : 1.0000	10.0000 : 0 : -10.0000	20.000 : 0	20.000 : 4.000	20.000 : 0 : -20.000
		0	0.9999 : 0		0	3.9995 : 0	
			0			0	
Разширен обхват	- 27649 : - 32512			- 10.0004 : - 11.7589			- 20.007 : - 23.515
Препълване	<=- 32513			0			0

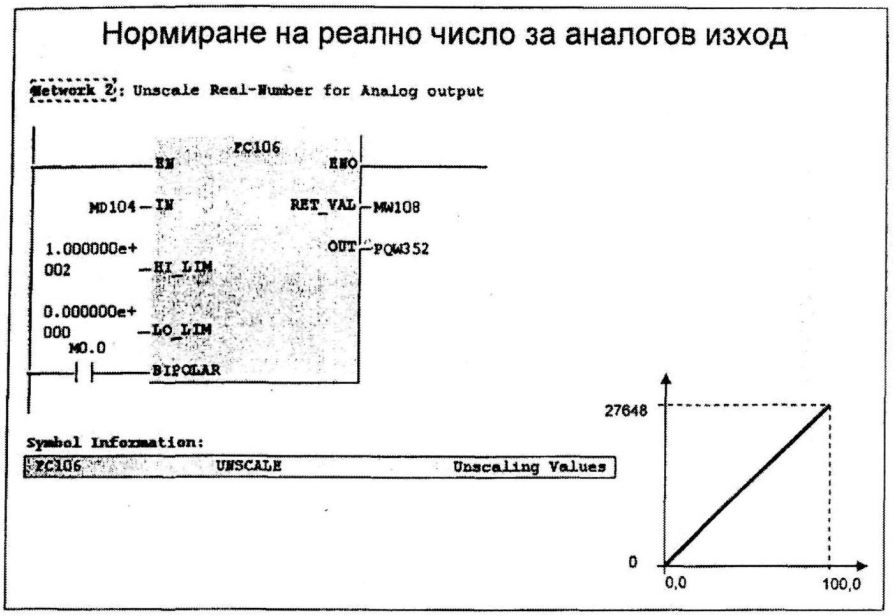
За асиметрични напреженови или токови обхвати, обхватът от -27648 до +27648 се преобразува в следните симетрични обхвати: ±10V; ± 20mA.

За симетрични напреженови или токови обхвати, обхватът от 0 до +27648 се преобразува в следните асиметрични обхвати: 0 до 10V; 1 до 5V; 0 до 20mA; 4 до 20mA.

Ако стойността,която трябва да се преобразува превиши граничните стойности на обхвата (препълване), на аналоговия изход се присвоява стойност (0V, 0mA).

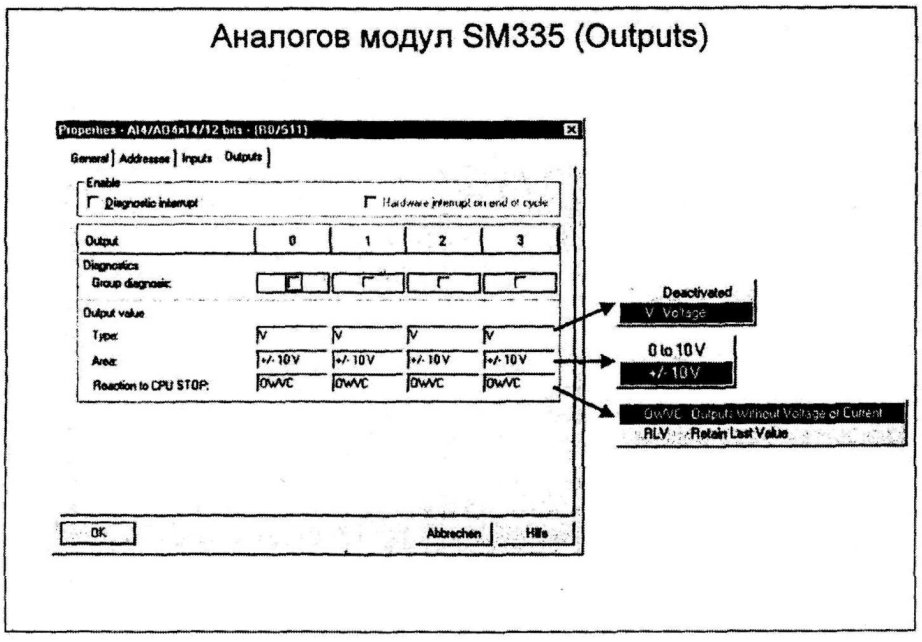
Съдържанието на изходните аналогови клетки си прехвърля във входния буфер на аналоговия изходен модул по време на фазата "сканиране на изходите" на оперативния цикъл. Нормално, собственото сканиране на аналоговите изходи е по-малко от времето на оперативния цикъл, поради което не се предвижда процедура за прекъсване от собствения цикъл на обновяване.

Нормиране на аналогова изходна величина

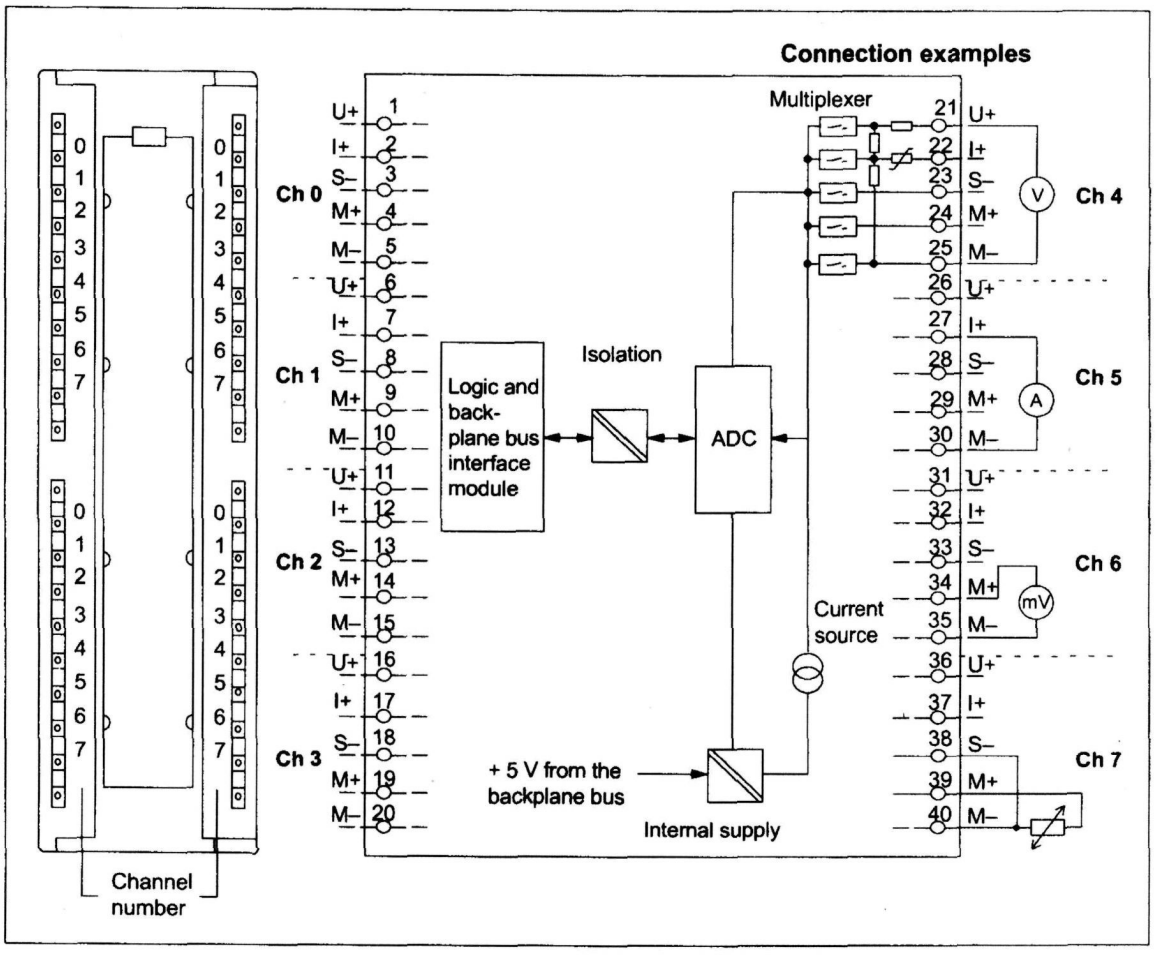


Нормирането на реално число за аналогов изход ще бъде пояснено с пример, където потребителската програма изчислява аналогова стойност в обхвата от 0 до 100.0% и тази стойност трябва да се изведе чрез аналогов изходен модул.

Стандартният блок FC106 се използва за мащабиране (преобразуване на реално число от 0 до 100.0% в 16-битово цяло число от 0 до 27648). Мащабираната аналогова стойност на изхода OUT може да се зареди под формата на 16-битово цяло число директно в аналоговия изходен модул или в даннова област.



Неизползваните изходни канали трябва да останат отворени хардуерно (не свързани накъсо, както аналоговите входове) и да бъдат деактивирани софтуерно.



Module View and Block Diagram of the Analog Input Module SM 331; AI 8 × 13 Bit

6. ПРОМИШЛЕНИ КОНТРОЛЕРИ SIMATIC – ОБЩ ПРЕГЛЕД

Въвеждане и извеждане на цифрова информация в промишлените контролери

Програмируемите логически контролери SIMATIC се изграждат на модулен принцип от произволно комбиниращи се модули с малки размери. Това дава възможност за изграждане на управление, което съответства точно на конкретната задача за реализиране. В зависимост от възможностите за решаване на задачи за управление от различен тип, контролерите SIMATIC се произвеждат в три класове – нисък, среден и висок.

Промишлени контролери от серията S7-200

Типичен представител на контролерите от нисък клас е SIMATIC S7-200. В конструктивно отношение S7-200 е изграден на модулен принцип и се използва предимно при автоматизацията на машини и съоръжения с ниски изисквания. CPU S7-200 комбинира в себе си микропроцесор, захранващ блок и входно/изходни устройства. Различните му модификации се различават по:

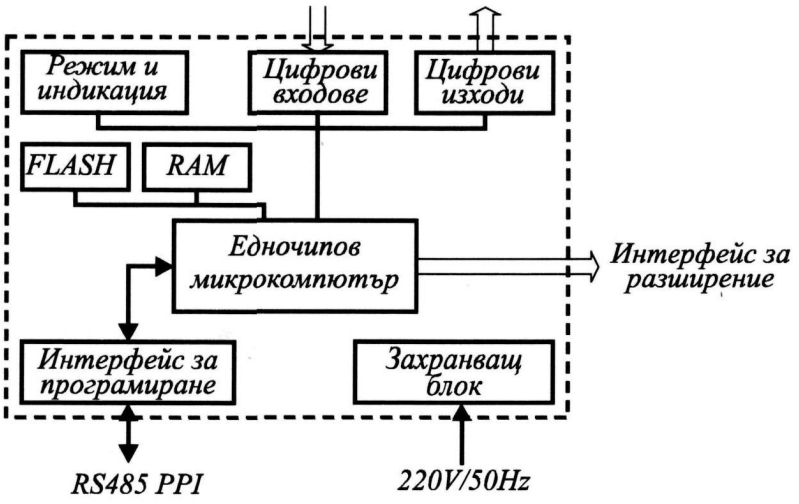
– различната изчислителна мощност на вградения микропроцесор;

– обема на паметта предназначена за потребителска програма;

– обема на потребителски достъпните даннови области адресируеми от операционната система;

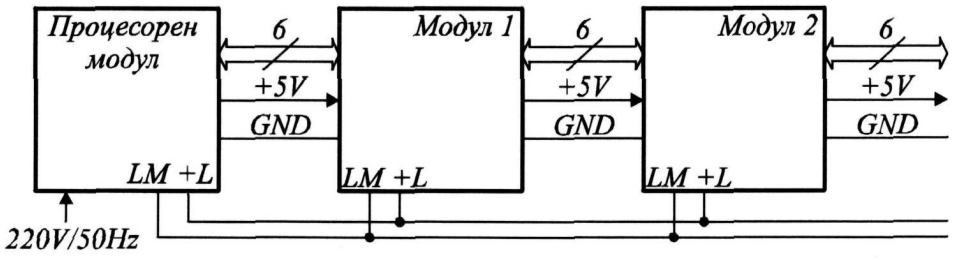
– броя и типа на вградените в модула входове и изходи.

Броят на входовете и изходите може да се разширява посредством допълнително включени модули. Изключение от това прави CPU-221, към което не могат да се свързват допълнителни модули. За останалите представители на серията S7-200, броят на допълнително включените модули е различен. Така например CPU-222 може да обслужва до 2 допълнителни модула от различен тип, CPU-224 и CPU-226 могат да обслужват до 7 такива модули.

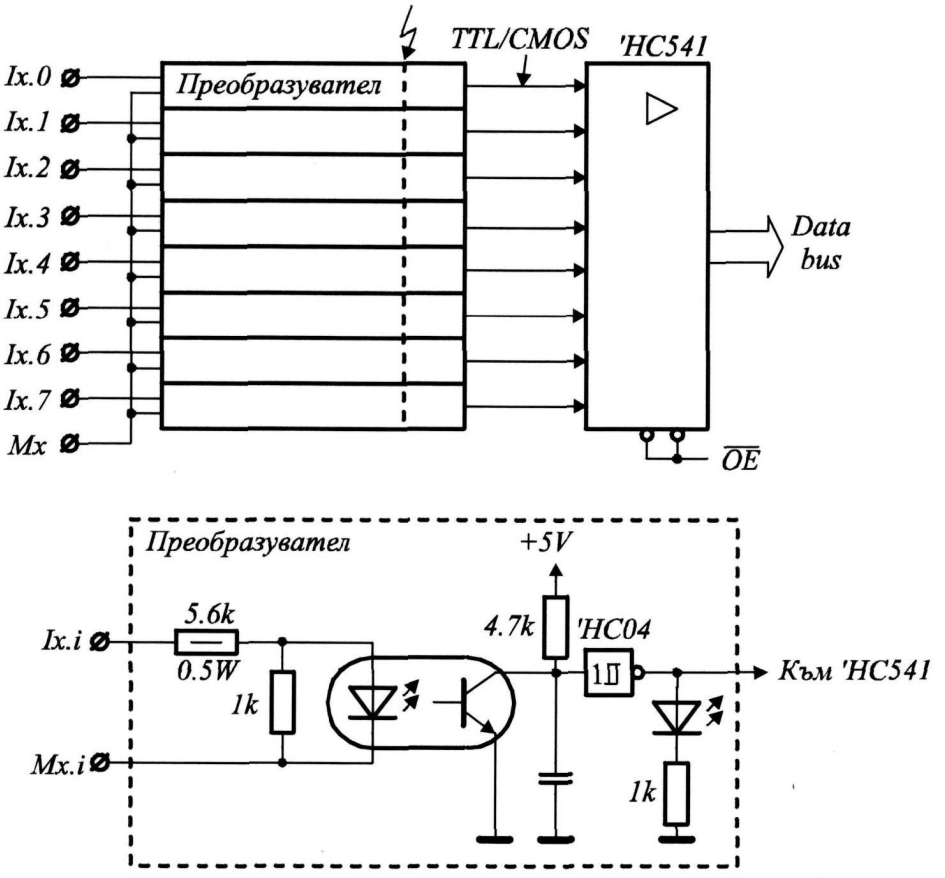


Фиг. 6.1. Типичен процесорен модул на SIMATIC S7-200.

Съществува широка гама от входни, изходни и входно/изходни модули, които са съобразени с най-често срещаните сензори и актюатори. Тези модули са реализирани по начин, елиминиращ нуждата от поставяне на междинни устройства за управлявания процес. Допълнителните модули към S7-200 се класифицират по предназначение и могат да бъдат сигнални, специализирани и модули от тип комуникационен процесор. Връзката между процесорния модул и допълнително включените модули е верижна, т.е. всеки от модулите е същевременно и разширителен – вж. фиг. 6.2.



Фиг. 6.2. Връзка между модули в SIMATIC S7-200.



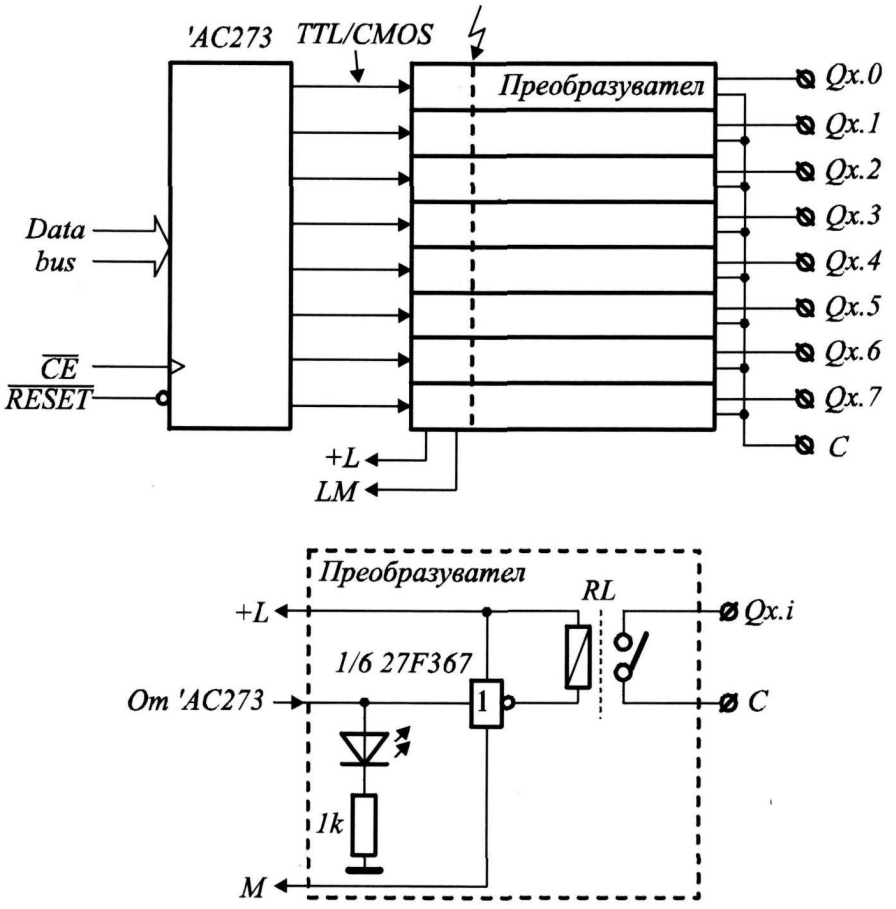
Фиг. 6.3. Структура на входен модул S7-200.

Сигналните цифрови разширителни модули могат да съдържат само цифрови входове, само цифрови изходи или комбинация от определен брой цифрови

входове и изходи. Цифровите входове са предназначени за връзка със сензори от логически тип. Цифровите изходи са предназначени за свързване към изпълнителни механизми от контактен тип. Изходните елементи, които се използват за тяхното реализиране, са от релеен или транзисторен тип.

На фиг. 6.3. е представена типична схема на реализиране на цифрови входове при S7-200. Галваничното развързване е с оптрони и е приложено поотделно за всеки вход. Едновременно с това обаче цифровите входове са групирани по октави (в група по осем), като само по един извод (едноименни) от всеки вход е изведен поотделно на входна точка, а другите изводи са свързани в обща входна точка *M*, която се свързва към минуса на захранването на сензорите. Включваните сензори трябва да са свързани по схема „общо захранване".

Обикновено сигналните разширителни модули се означават с „EM", което означение идва от английското им наименование Expanded Module. Обикновено модулите с релейни изходи имат като добавка в означението си RELAY. Като пример за това може да се даде разширителния модул EM223 RELAY, който включва в състава си 16 цифрови входа и 16 цифрови изхода от релеен тип.



Фиг. 6.4. Структура на изходен модул на SIMATIC S7-200.

S7-200 разполага с широк спектър от сигнални модули включващи само цифрови входове (EM221), само цифрови изходи (EM222), както и такива, кои-

то включват цифрови входове и изходи (EM223).

На фиг. 6.4. е представена типична схема на реализиране на релейни цифрови изходи при S7-200. Галваничното развързване е реализирано чрез релета във всеки изход. Подобно на входовете цифровите входове също са групирани по октави, като само по един извод от всеки изход е изведен поотделно на изходна точка, а другите изводи са свързани в обща изходна точка *C*. Включваните актюатори могат да са свързани по схема „общо захранване" или „обща маса", като изпълнението на свързването трябва да е еднакво за цялата октава.

Специализираните разширителни модули на S7-200 са предназначени за връзка със сензори и изпълнителни механизми, изискващи допълнителна обработка на входната или изходната информация. В общия случай това са аналогови входове и изходи, с включени допълнително апаратни блокове за филтриране, мащабиране и линеаризиране на обработваните сигнали. Аналоговите разширителни модули на SIMATIC S7-200 включват от 4/0, 4/1, 0/2 входа/изхода. Към специализираните модули се включени и модулите за точно температурно измерване с точност до 0,1°C, RTD модули за съпротивителни температурни датчици, TC модули за измерване на полуполярни термоелементи, както и позициониращи модули.

Разширителните модули от тип „комуникационен процесор" са предназначени за изграждане на локални мрежи и разпределени системи за управление. SIMATIC S7-200 има няколко възможности за комуникация. CPU разполага с интегриран сериен интерфейс RS485, а опционно се предлагат разширителни модули за AS-iterface и ProfiBus-DP (само като подчинен модул). Интегрираният RS485 има следните комуникационни възможности:

– комуникация като master или slave в мрежа PPI;

– участник в MPI мрежа само като slave;

– комуникация с уреди за визуализация и контрол (Text Display, Touch Panel);

– програмиране на S7-200, чрез програмираща станция или персонален компютър посредством PC/PPI кабел.

Вграденият в SIMATIC S7-200 интерфейс се характеризира със:

– скорост на обмен на данни от 9,6 до 187,5 kB/s;

– режим на работа като RS485 или RS232 чрез опционният PC/PPI кабел;

– максимален брой участници в мрежата до 126 (до 32 за сегмент).

Промишлени контролери от серията S7-300

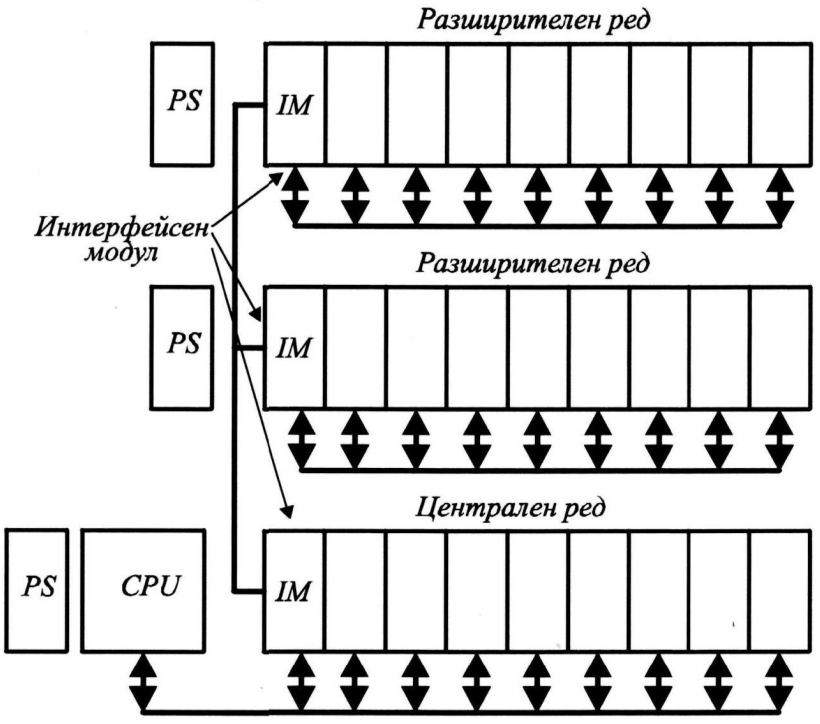
Програмируемият логически контролер SIMATIC S7-300 е типичен представител на контролерите от среден клас. Той се използва за различни приложения в автоматизационната техника. Модулната конструкция без вентилатор, лесното реализиране на децентрализирано управление и лесното и удобно обслужване правят SIMATIC S7-300 решение за най-разнообразни автоматизаци-

онни задачи в следните браншове: производствена техника, автомобилна индустрия, общо машиностроене, специалното машиностроене, хранително вкусовата промишленост и др.

За изграждане на контролера на разположение са няколко централни процесора с различни възможности, част от които са дадени в табл. 1.

Табл. 1. Централни процесори при S7-300.

CPU	Интегрирана периферия	Интегрирани технологични функции	Интегрирани интерфейси
CPU312IFM	Цифрова	Броене	MPI
CPU312C	Цифрова	Броене	MPI
CPU313			MPI
CPU313C	Цифрова, аналогова	Броене	MPI
CPU313-2PtP	Цифрова	Броене	MPI, PtP
CPU313C-2DP	Цифрова	Броене	MPI, DP
CPU314		Броене	MPI
CPU314IFM	Цифрова, аналогова		MPI
CPU314C-2PtP	Цифрова, аналогова	Броене, позициониране	MPI, PtP
CPU314C-2DP	Цифрова, аналогова	Броене, позициониране	MPI, DP
CPU315		Броене, позициониране	MPI
CPU315-2DP			MPI, DP
CPU316-2DP			MPI, DP
CPU318-2DP			MPI, DP

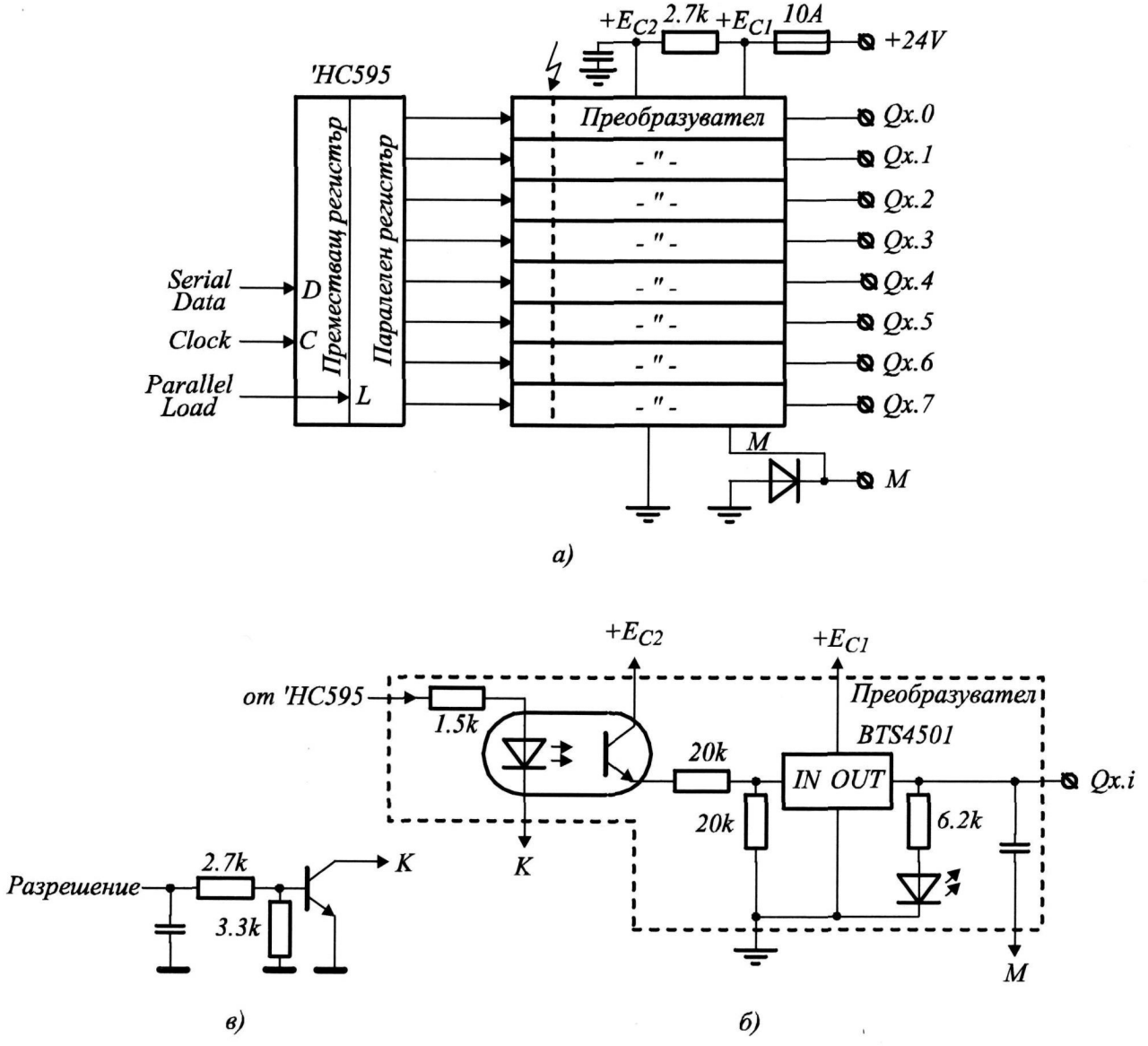


Фиг. 6.5. Разширяване на SIMATIC S7-300 в монтажни редове.

CPU модулите могат да бъдат със или без интегрирана периферия. Обемът на работната памет за потребителската програма варира от 6KB (при

CPU312IFM) до 512kB (при CPU318-2DP). Всяка инструкция заема около 3 байта от паметта. Така обемът на инструкциите при CPU318-2DP може да достигне до 84K инструкции.

Връзката между централния процесор и модулите в SIMATIC S7-300 е организирана по паралелна информационна шина. Паралелната шина има значително по-голяма скорост на обмен на информацията, спрямо последователната. За разлика от верижната връзка, паралелната дава възможност отстраняването на даден модул, без да се нарушава връзката между останалите.



Фиг. 6.6. Структура на изходен модул при SIMATIC S7-300.

Към централния процесор на SIMATIC S7-300 могат да бъдат свързани до 8 модула в един монтажен ред. Ако автоматизационната задача изисква използването на повече от 8 модула, то системата може да бъде разширена с допълнителни монтажни редове. Комуникацията между отделните монтажни редове се

реализира от т.нар. интерфейсни разширителни модули. В системата S7-300 могат да се включват максимално 32 разширителни модула, като всеки разширителен ред включва до 8 модула. При пространствено раздалечени съоръжения отделните монтажни редове могат да бъдат изградени на разстояние 10m един от друг. Фиг. 6.5 илюстрира свързването на модулите на S7-300 чрез организирането на монтажни редове.

С прилагането на едноредова структура, в която всеки един разширителен модул има по 16 входа и 16 изхода, се получава контролер с 256 входа и 265 изхода. При многоредовата структура максималният брой на входовете и изходите е 1024.

Структурата на типична организация на цифрови изходи при SIMATIC S7-300 е представена на фиг. 6.6. Тя е подобна на използуваната при S7-200, с тази разлика, че изходните битове се извеждат последователно в регистър за прехвърлянето им изходните точки. Галваничното развързване е с оптрони във всеки изход. Изходите са от електронен тип, като актюаторите се свързват по схема „общо захранване".

На фиг. 6.7 е показано изпълнението на цифрови входове при S7-300. Информацията от всяка входна точка преминава през преобразувател (фиг. 6.7.б), който изработва на изхода си CMOS логически нива. Преобразувателят не е галванично развързан. Основното звено в него е компараторът LM2901, който със своя хистерезис въвежда шумоустойчивост по амплитуда спрямо входния сигнал. Групата от осем цифрови входове могат да поемат сигнал от сензори от контактен тип, свързани в схема „общо захранване". Получените от осемте входни точки данни се преобразува в последователен вид и по сериен канал се въвеждат в системата. В серийния канал е въведено галванично развързване. За неговото реализиране са използувани само три оптронни вериги (фиг. 6.7.в) – за паралелното зареждане в преместващия регистър, за тактовия сигнал и за последователно извежданите данни.

SIMATIC S7-300 може да бъде участник в две различни в йерархично отношение мрежи:

– полева комуникационна мрежа;

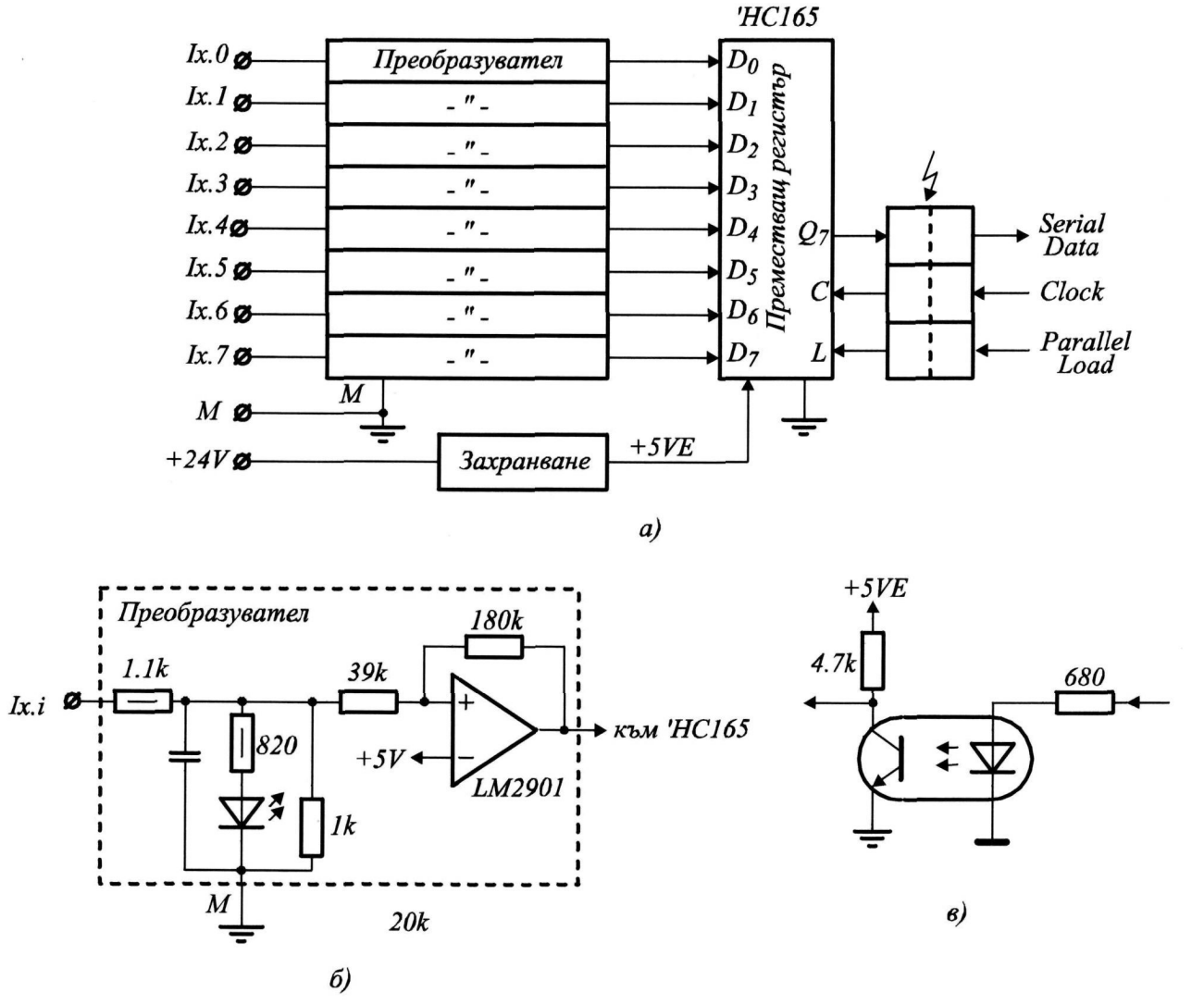
– даннова комуникационна мрежа.

Полева комуникация е за свързване на сензори или изпълнителни механизми към един централен процесор. Свързването се осъществява чрез интегрирани интерфейси (вградени в процесорния модул) или чрез допълнителни модули от тип комуникационен процесор (CPs). Полева комуникация се подържа от мрежите AS-interface и PROFIBUS-DP.

Данновата комуникация служи за обмен на данни между автоматизиращи устройства или между автоматизиращо устройство и персонален компютър. Данновата комуникация се реализира чрез Multi Point Interface (MPI), ProfiBus-DP и Industrial Ethernet.

Директно интегриран в CPU е многоточковият интерфейс MPI. В MPI мре-

жата могат да се включат до 125 участника със скорост на предаване 187,5 kBit/s. Този интерфейс може да се използва да обмен на данни между различните управления, като по този начин едно CPU може да има достъп до входовете и изходите на друго управление. Използването на този интерфейс предоставя възможности за програмиране и наблюдение на работата на контролера в реално време. Освен това той дава възможност един програматор да обслужва няколко CPU, както и няколко програматора да имат достъп до едно CPU.



Фиг. 6.7. Реализиране на цифрови входове при SIMATIC S7-300.

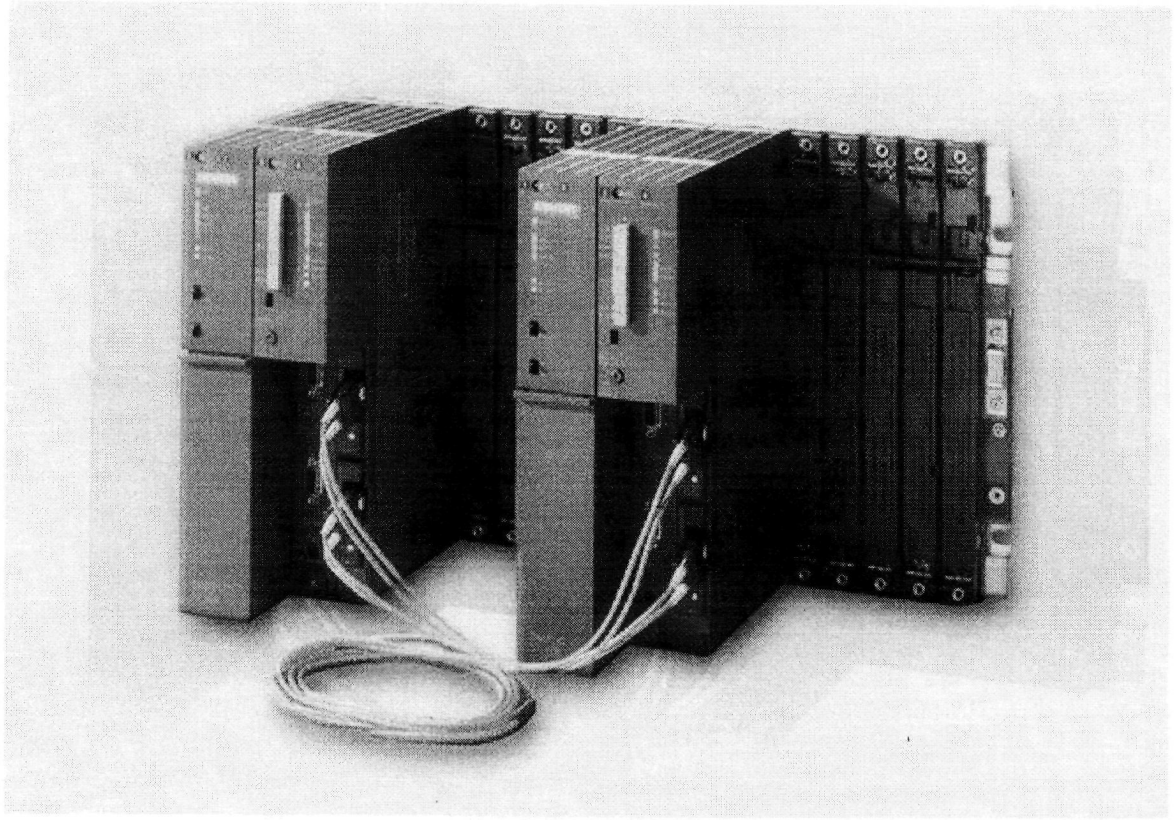
Някои от представителите на фамилията SIMATIC S7-300 имат, интегриран в CPU ProfiBus-DP интерфейс. Той се използва при изграждането на децентрализирани структури на управление. SIMATIC S7-300 може да бъде както slave, така и master в ProdiBus-DP мрежите.

Point to Point е допълнителен сериен интерфейс, интегриран в някои CPU, който е предназначен за свързване на периферни устройства, като скенери, принтери, панели за визуализация

SIMATIC S7-300 се програмира с базовия софтуер STEP 7. Наборът от команди наброява до 350 команди, които включват функции за 32 битова аритметика с плаваща запетая.

Промишлени контролери от серията S7-400

SIMATIC S7-400 програмируем логически контролер от висок клас. S7-400H е контролер, който е с повишена надеждност и е предназначен за управление на непрекъсваеми процеси. Изграден е изцяло на модулна основа, а характерното тук е че модулите могат да бъдат подменяни по всяко време на работа на системата без да е необходимо нейното рестартиране. Освен това няколко CPU могат да работят паралелно в една система – мултипроцесорна работа, с което се повишава функционалността на работа. Броят на процесорните модули може да се комбинира от два до четири процесорни модули (Multicomputing). Обменът на данни на CPU с входовете и изходите на контролера се осъществява по 'P-Bus' и 'K-Bus' магистрали. Чрез тях се реализира значително по-висока скорост на обмен, както и предаването на предаването на големи пакети от данни към функционалните и интелигентните модули.



SIMATIC S7-400 има функции за комуникация по Industrial Ethernet, ProfiBus и MPI, които са интегрирани в операционната система на контролера. Могат да работят и като интегрирано решение за приложения Motion Control. Проектирането се извършва включително и чрез базираните на STEP 7 езици от

високо ниво. Връзката към света на информационните технологии се осъществява чрез Ethernet и включва организирането на Web-страници и E-mail.

Времето за изпълнение на 1K инструкции при S7-400 (~0,1ms) е значително по-малко спрямо S7-300 (~0,6ms). Увеличена е и паметта за данни до 2MB както и тази за потребителската програма също до 2MB (CPU417).

Състав на SIMATIC S7-400

Системата S7-400 се състои от:

– Монтажна касета

– Захранващ модул (PS) осигуряващ оперативно напрежение

– Централен процесорен модул (CPU) с интегрирана връзка към децентрализирана периферия

– Сигнални модули (SM) за цифрови (DI/DO) и аналогови (AI/AO) входове/изходи

– Комуникационни модули (CP) за връзка в мрежа или точка в точка

– Функционални модули (FM) за претенциозни задачи като броене и позициониране

Конструкция на SIMATIC S7-400

S7-400 се изгражда модулно от произволно комбинируеми модули. Монтажните касети са с вградена даннова шина, върху които се окачват и монтират модулите. Системата притежава дефинирана дълбочина на вграждане, като всички кабелни връзки и куплунзи се вграждат в модула и се защитават чрез кабелен капак. При изграждането не е необходимо да се съблюдават правила за спазване на конкретно монтажно място и сигналните модули могат да бъдат подменяни под напрежение, т.е. без спиране на процеса и производствена пауза. Тъй като на монтажната касета не се намират активни елементи се постига най-висока степен на надеждност за цялата система. Всички хардуерни настройки се извършват софтуерно.

Разширителните модули са капсуловани и могат да работят както при нормални, така и при тежки индустриални условия, а краткото време за обработка на инструкциите правят SIMATIC S7-400 приложим за автоматизационни задачи от висок клас. Охлаждането на всички модули е без вентилатор, което освен че снижава нивото на шума, повишава надеждността на експлоатацията им.

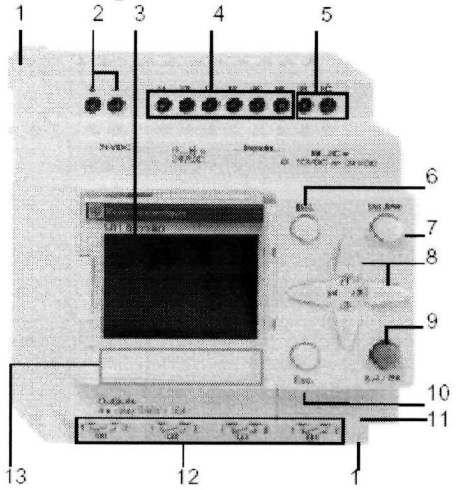
7. ПРОМИШЛЕНИ КОНТРОЛЕРИ TELEMECANIQUE – ОБЩ ПРЕГЛЕД

Серията контролери на Telemecanique включва програмируеми логически модули с различни възможности даващи по-лесна адаптация при изграждане на управление. Контролерите са разделени в три класа: нисък, среден и висок в зависимост от тяхното предназначение и възможности. Към контролерите от нисък клас спадат контролерите от гамата Zelio Logic и Modicon.

Промишлени контролери от серията Zelio Telemecanique

Контролерите от серията Zelio Logic са от фамилията SR1 са от марката Telemecanique на фирмата Schneider Electric. Те се наричат още „логически релета" (Zelio-Logic relays). Тези логически релета притежават характерните входове и релейни изходи както типичните PLC, но могат да не бъдат свързвани в мрежа и представляват напълно автономни програмируеми компактни контролери. Различните модификации включват от 6 до 12 входа за свързване на сензори и бутони, 8 бутона предназначени за програмиране и настройка, екран за визуализация, от 4 до 8 изхода за управление на изпълнителни механизми. Всеки контролер от гамата Zelio Logic има вграден интерфейс за връзка с програматор.

На фиг.1. е показан външния изглед на PLC от фамилията SR1.



1 - Капак.
2 - Изводи за захранване.
3 - LCD дисплей (4 линии, 12 символа).
4 - Изводи за вход.
5 - Изводи за аналогови входове (0-10V или 24V DC).
6 - Бутон за изтриване или отменяне на избор.
7 - Бутон за вмъкване на нов ред.
8 - Клавиши за посока или входни клавиши в RUN режим.
9 - Бутон за избор или потвърждение.
10 - Бутон за отказ.
11 - Извод за патрон с EEPROM или за кабел за връзка с PC.
12 - Изводи за изход на релето.
13 - Област за маркиране.

Фиг.1. PLC SR1 Zelio-logic relay – общ изглед.

Контролерите Zelio имат следните видове интегрирани функции:
– 8 закъснителни блока, регулируеми от 10 ms до 100h;
– 8 брояча от 0000 до 9999;
– 15 помощни релета;
– при някои от контролерите има вграден седмичен часовник, а при други

има допълнителни аналогови входове.

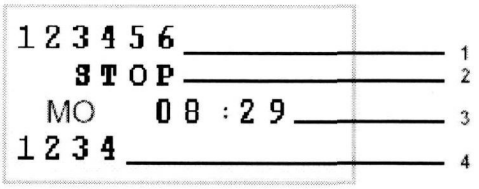
Изходите на контролерите от гамата Zelio Logic могат да бъдат както релейни, така и транзисторни. Релейните изходи са с номинален ток 8А, а транзисторните до 0,5А (общ колектор или общ дрейн).

Цифровите входове на Zelio Logic могат да бъдат за постоянно напрежение 24V или за променливо напрежение 100 – 240V. Аналоговите входове могат да измерват входна аналогова величина в обхват от 0 до 10 V. При някои от модулите, част от функционалните клавиши (Z1-Z4) могат да се конфигурират за работа като стандартни входове.

За програмирането, настройката и диагностиката на контролерите от гамата Zelio Logic се използва програматор. Като такъв най-често се използва персонален компютър, на който е зареден програмният продукт Zelio Soft, работещ в среда Windows. Този програмен продукт има вграден симулатор, чието използване позволява да се открият и коригират грешките в алгоритъма на работа на потребителската програма, преди да бъде заредена в контролера.

Част от контролерите от гамата Zelio Logic се произвеждат без екран за визуализация и функционални клавиши.

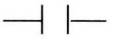
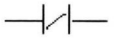
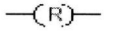
Основният екран на Zelio-Logic relay е показан на фиг.3.



1 – Моментно състояние на входовете.
2 – Индикация за избран режим RUN или STOP.
3 – Индикация за параметър (за релетата с часовник денят и времето за с подразбиране).
4 –Състояние на изходите.

Фиг.2. Основен екран на Zelio-Logic relay.

Таблица1.

Function		Electrical Scheme	Ladder Language	Zelio Relay Symbol	Notes
Contact	N.O. SPSTNO			Ix ▲	I corresponds to the real image of the contact connected to the input of the module.
	N.C. SPSTNC			Ix ▲	i (I) corresponds to the reversed image of the contact connected to the input of the module.
Standard Coil				Qx	The coil is energized when the contacts to which it is connected are closed.
Latch Coil (Set)				SQ	The coil is energized when the contacts to which it is connected are closed. It remains energized when the contacts re-open.
Unlatch Coil (Reset)				RQ	The coil is de-energized when the contacts to which it is connected are closed. It remains inactive when the contacts re-open.

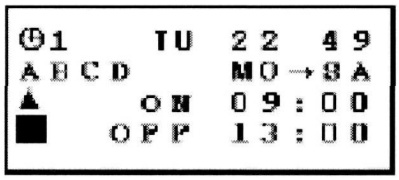
ОПЕРАЦИОННИ СИСТЕМИ ЗА РЕАЛНО ВРЕМЕ. ОСНОВНИ ПОНЯТИЯ

1. Структура на изчислителната система

А. Хардуер
Б. Софтуер
   1. Потребител
   2. Приложни програми
   3. Други системни програми
   4. Операционна система
   5. Техническо обезпечение

Основните символи използвани при програмирането на Zelio-Logic relay са показани в табл. 1.

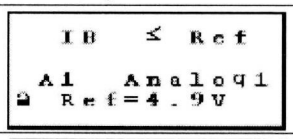
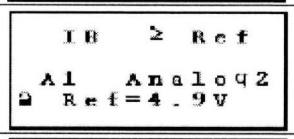
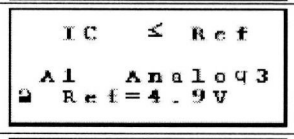
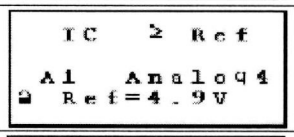
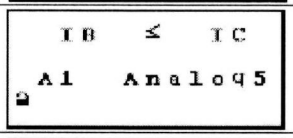
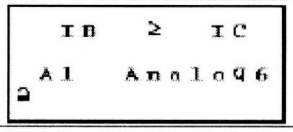
Някои PLC от фамилията SR1 притежават четири часовника, програмируеми за работа 24 часа – 7 дни. На фиг.3 е показан номерът на часовника, който се използва, денят и часът на програмиране, периодът в дни за които е активен, както и часът на включване и изключване.



Фиг.3. Часовников екран на Zelio-Logic relay.

В табл.2. са описани възможните функции на аналоговите входове (компаратори) за някои от представителите на SR1, които са със захранване 24V DC. Стойността на входния аналогов сигнал може да се изменя от 0 до 10 V.

Таблица 2.

Функция	Описание
	Контакт A1 е затворен когато стойността на аналоговия вход е IB не надвишава опорното напрежение, въведено в полето Ref (в случая 4.9V).
	Контакт A1 е затворен когато стойността на аналоговия вход е IB е равна или надвишава опорното напрежение, въведено в полето Ref (в случая 4.9V).
	Контакт A1 е затворен когато стойността на аналоговия вход е IC не надвишава опорното напрежение, въведено в полето Ref (в случая 4.9V).
	Контакт A1 е затворен когато стойността на аналоговия вход е IC е равна или надвишава опорното напрежение, въведено в полето Ref (в случая 4.9V).
	Контакт A1 е затворен когато стойността на аналоговия вход е IB е равна или надвишава стойността на аналоговия вход IC.
	Контакт A1 е затворен когато стойността на аналоговия вход е IB е равна или надвишава стойността на аналоговия вход IC.
	Контакт A1 е затворен когато стойността на аналоговия вход е IB е между IC-H и IC+H. H (хистерезисът) е въведен в полето H (в случая 4.9V).

Таймери в софтуерния продукт Zelio-soft – общо описание

Функционаленият блок на таймера в софтуерния продукт Zelio-soft позволява да бъдат извършвани манипулации и действия с времето. Този блок съдържа:

1. Вход за нулиране (Reset) – RT. Използван като „бобина", този сигнал нулира текущата стойност на таймера или деактивира контакта T, изпълняващ функцията на изход. Под бобини се разбират вътрешни релета, изходни релета, таймери, броячи.

2. Команден вход – TT, използван като бобина – неговото действие зависи от типа на използвания в приложението таймер.

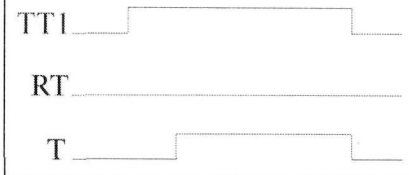
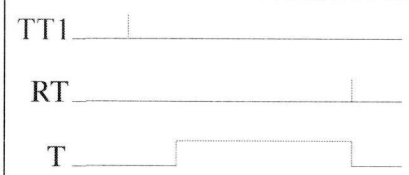
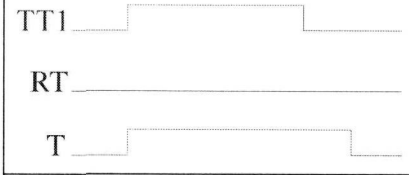
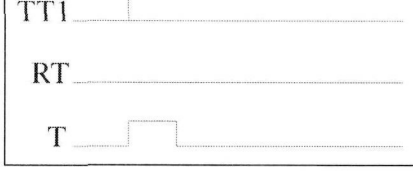
3. Изход T (нормално отворен) или t (нормално затворен).

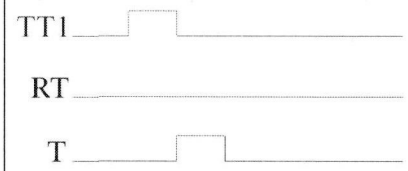
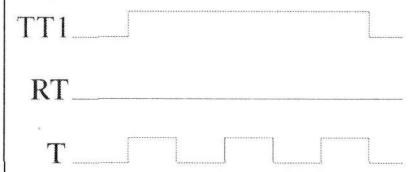
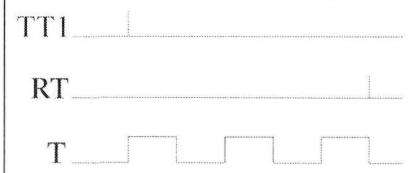
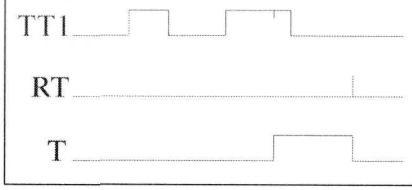
Стойността t=00.00 се нарича „преизбрана" (preselection) стойност. Измервателната единица за време при тази стойност се означава с S и тук е възможен следния избор:

– 1/100 от секундата : 00.00 s (максимум : 99.99)
– 1/10 от секундата : 000.0 s (максимум : 999.9)
– минути : секунди : 00 :00 M : S (максимум : 99:59)
– часове : минути : 00 : 00 H : M (максимум : 99:59).

Съществува възможност функционалният блок да бъде заключен чрез заключващия параметър, означен с . В тази ситуация преизбраната стойност вече не се появява в менюто за параметри („PARAMETER") и тя не може да бъде променяна. Типовете таймери в Zelio-soft са показани в табл. 3.

Таблица 3.

Вид	Коментар
	Тип A: Таймер със закъснение при включване. Пример: закъснение в затварянето на контактора с цел ограничаване на тока.
	Тип a: Таймер със закъснение при включване по фронт и с изключване при фронт на Reset
	Тип C: Таймер със закъснение при изключване. Пример: поддържане въртенето на вентилатора след изключване на мотор.
	Тип B: Калибриращ импулс по управляващия нарастващ фронт (временен контакт). Пример: включване на светлинен източник посредством бутон с ключ за осветление.

Вид	Коментар
	Тип W: Калибриращ импулс по управляващия спадащ фронт. Пример: затваряне на врата.
	Тип D: Симетрично премигване (повтарящи се цикли). Пример: погрешно показание с премигващ светлинен индикатор.
	Тип d: Симетрично премигване, стартирано по входен нарастващ фронт и спирано с Reset. Пример: команда за спиране на импулсна поредица след преустановяване на захранването.
	Тип T: Стартира се при постъпване на управляващ импулс с по-голяма продължителност от зададена, нулира се с Reset. Пример: изисква поставяне на филтър, когато препоръчителното време е надхвърлено.

Съществуват 8 типа таймери. Всеки тип има свои функционални особености, които намират различно приложение. В табл.3. е представено детайлно описание на таймерите.

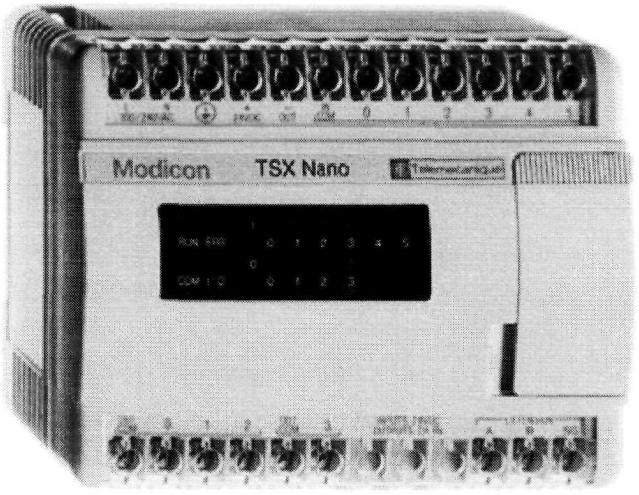
Приложната програма за управление на таймера е представена чрез Диаграма-стълба (Ladder Diagram, LD). Диаграмата е аналогия на електрическа схема с контакти и релета (релейно-контактна схема). В най-лявата и най-дясната част на диаграмата се намират вертикални „захранващи шини". В пространството между тях се разполагат мрежи, състоящи се от свързани помежду си, а също със захранващите шини елементи.

Промишлени контролери от фамилията Modicon TSX Nano

Програмируемите логически контролери от фамилията Modicon TSX Nano се използват малки автоматизирани системи като: преси, помпи, металорежещи машини и машини за хранително вкусовата промишленост. Базовата им конфигурация включва от 6 до 14 цифрови входа и от 4 до 9 цифрови изхода. Обикновено контролерите се захранват с променливо напрежение от 110 до 220 V. Входовете на Modicon TSX Nano се захранват от контролера с постоянно напрежение 24V и имат консумация до 150 mA. Изходите са релейни с номинален ток 1A. Входовете за сензори могат да работят с положителна и отрицателна логика и да възприемат импулси с честота то 10 kHz. Програмирането на контролерите се извършва посредством терминал по физическия стандарт RS 485.

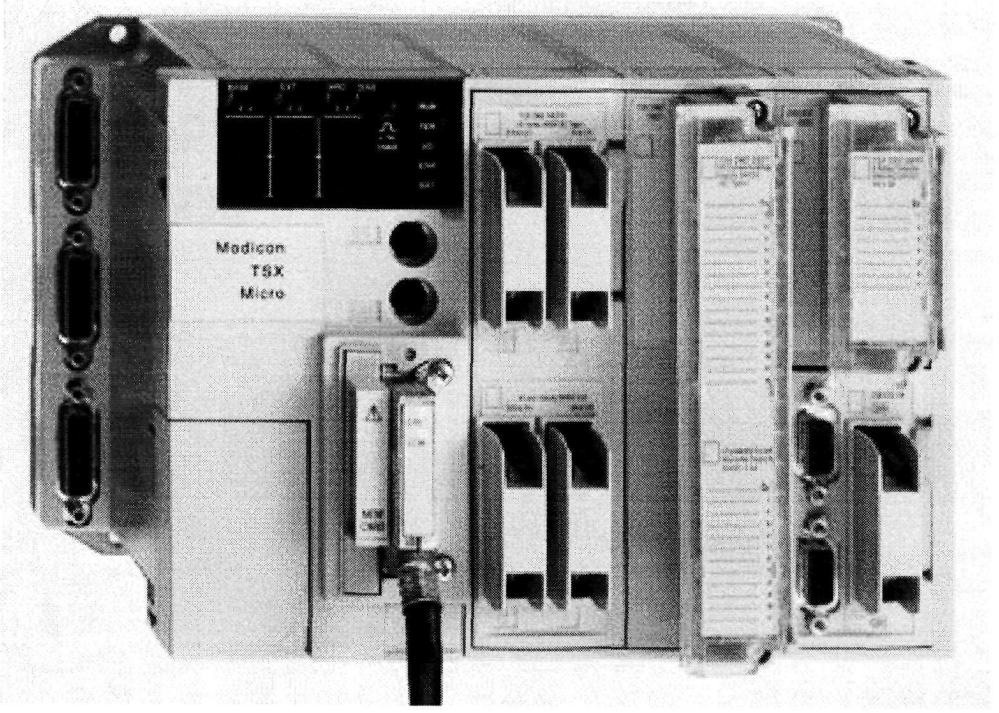
Програмно вградените функции в контролера включват:
– 16 регистъра за дата и час;
– 32 таймера разрешаваща способност 10ms, 100ms, 1s и 1 min;
– 16 реверсивни брояча (0 до 9999);
– 4 двубайтови регистъра.

Потребителската програма се записва в RAM и EEPROM, като информацията в RAM паметта може да се запази до 30 дни с помощта на батерия.



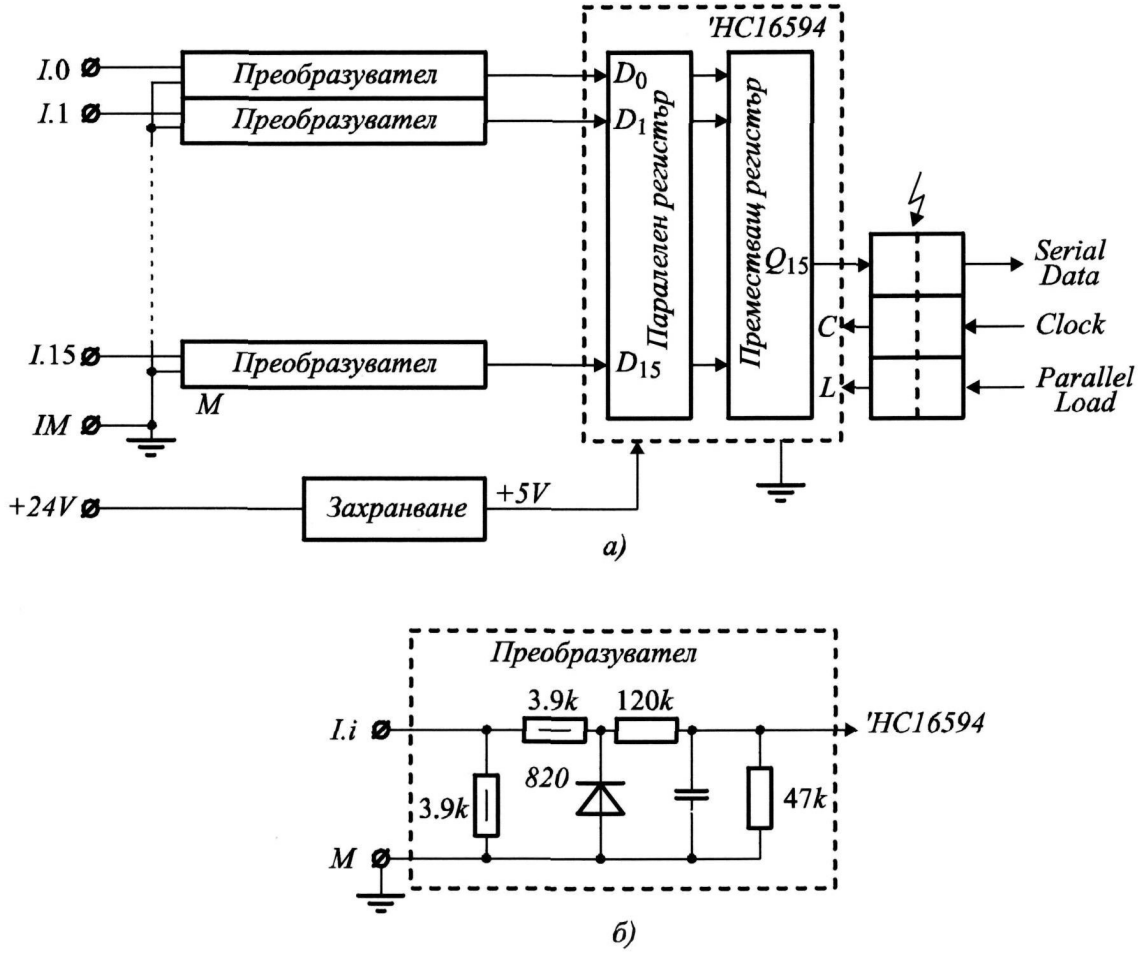
Фиг.4. Общ вид на промишлен контролер от фамилията Modicon TSX Nano.

Промишлени контролери от фамилията Modicon TSX Micro



Фиг.5. Общ вид на промишлен контролер от фамилията Modicon TSX Micro.

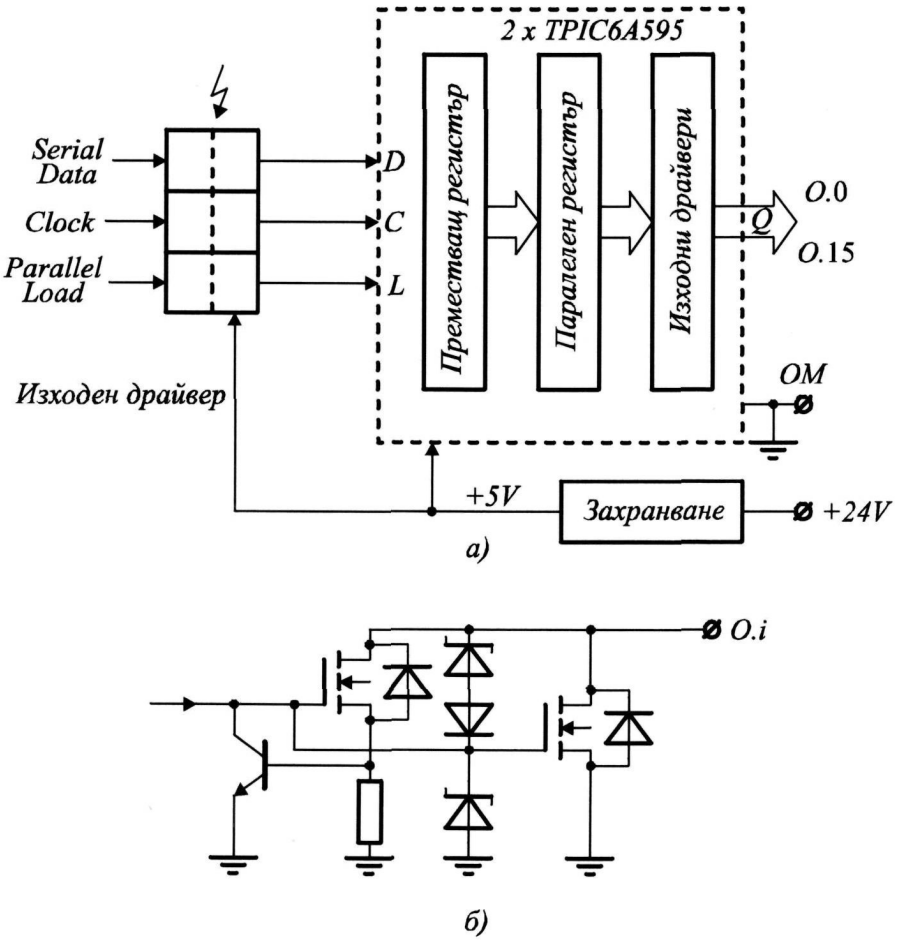
Modicon TSX Micro е типичен представител на контролерите от среден клас на Telemecanique. Той може да управлява до 250 дискретни входа / изхода, 40 аналогови входа / изхода и да се свързва към индустриални мрежи. Този тип контролери се изграждат изцяло на модулна основа. В CPU модула на контролера е вграден физическият интерфейс RS 485. Допълнително да се включват модули за комуникация към индустриални мрежи ModBus и ModBus Plus. За връзка със сензори и изпълнителни механизми има специално разработени модули с вграден AS – Interface.



Фиг.6. Реализиране на цифрови входове при Modicon TSX Micro.

На фиг.4 е показано изпълнението на цифрови входове при Modicon TSX Micro. Информацията от всяка входна точка преминава през преобразувател, който изработва на изхода си CMOS логически нива. Преобразувателят не е галванично развързан. Групата от шестнадесет цифрови входове могат да поемат сигнал от сензори от контактен тип, свързани в схема „общо захранване" (PNP тип). Получените от шестнадесетте входни точки данни се преобразува в последователен вид и по сериен канал се въвеждат в системата. В серийния канал е въведено галванично развързване. За неговото реализиране са използувани само три оптронни вериги – за паралелното зареждане в преместващия ре-

гистър, за тактовия сигнал и за последователно извежданите данни.



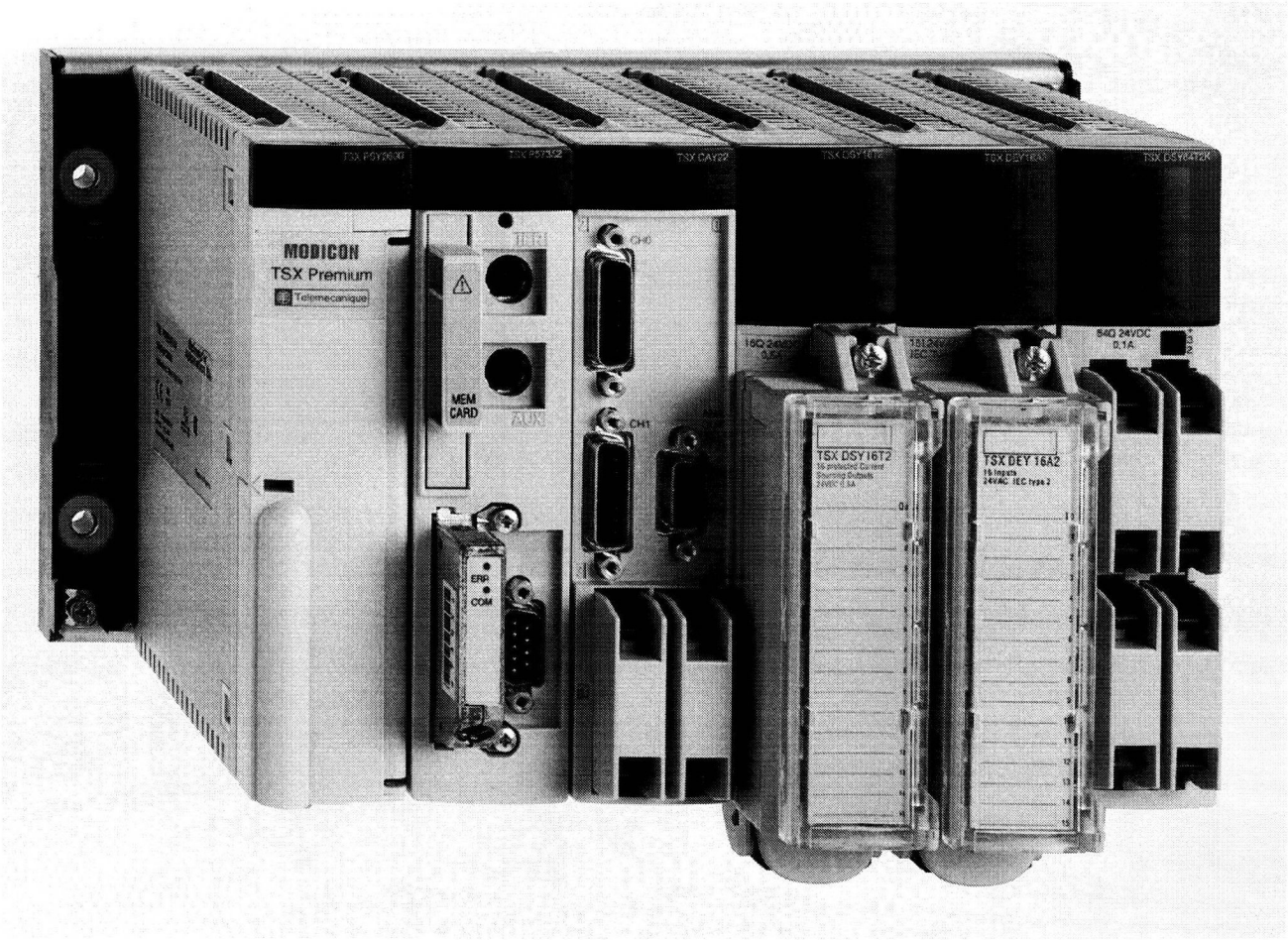
Фиг.7. Структура на изходен модул на Modicon TSX Micro.

На фиг.5. е представена типична схема на реализиране на транзисторни цифрови изходи при Modicon TSX Micro. Изходите се извеждат чрез специализирана интегрална схема TPIC6A595. Галваничното развързване е реализирано едновременно за група от шестнадесет изхода чрез оптронно развързване в последователния вход, входа за зареждане и тактовия вход на специализираната схема. Включваните актюатори трябва да са свързани по схема „общо захранване" (NPN тип).

Промишлени контролери от фамилията Modicon TSX Premium

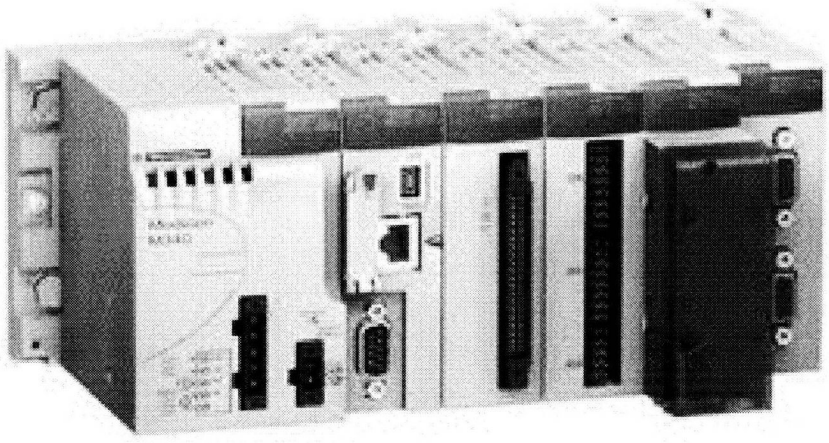
Modicon TSX Premium е типичен представител на контролерите от висок клас на Telemecanique. Той може да управлява до 4000 цифрови входа / изхода, до 256 аналогови входа / изхода. Към контролера се предлагат разширителни модули за управление на непрекъснати процеси, за измерване на тегло, както и модули с вградени бързи броячи. (могат да броят по-кратки събития отколкото времето за един оперативен цикъл). Този клас контролери могат да се свързват

в глобални и локални мрежи EtherNet/IP, Modbus TCP/IP, Modbus, and Modbus Plus, Interbus-S, Profibus-DP, CANOpen, FIPWAY, FIPI/O and Unitelway.



Фиг.8. Общ вид на промишлен контролер от фамилията Modicon TSX Premium.

Промишлени контролери от фамилията Modicon TSX Quantum

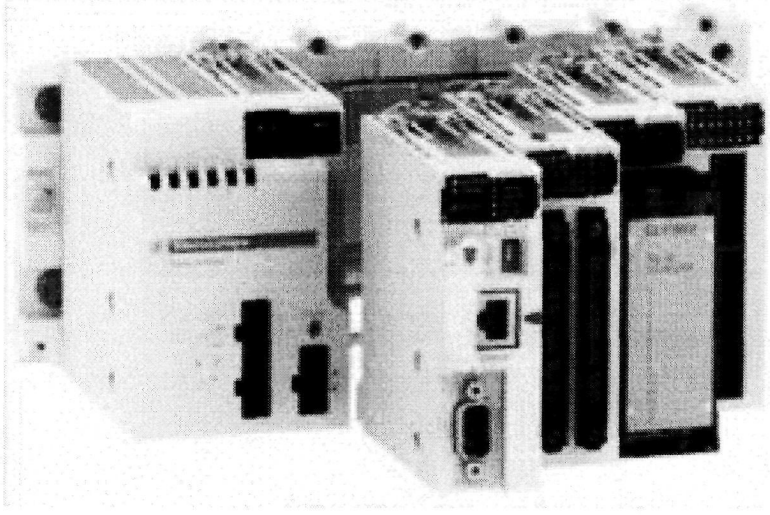


Фиг.9. Общ вид на промишлен контролер от фамилията Modicon TSX Quantum.

Modicon TSX Quantum е представител на контролерите от най-висок клас на Telemecanique. Изгражда се изцяло на модулна основа. Може да управлява до 64000 цифрови входа / изхода, до 72 аналогови модула. Всеки модул притежава самостоятелна автономност и висока надеждност. Този клас контролери могат да се свързват в глобални и локални мрежи Modbus®, Modbus Plus, Modbus Ethernet TCP/IP and EtherNet/IP.

Промишлени контролери за честотно управление на двигатели Altivar

Telemecanique предлага и програмируеми логически контролери специализирани за честотно регулиране на асинхронни двигатели. Към този тип контролери спадат: Altivar 08, Altivar 28 и Altivar 58. Те се различават по номиналните стойности на тока и напрежението и са предназначени за управление на асинхронни двигатели. Реверсивното действие може да бъде забранено за приложения, където двигателят се върти само в едната посока.



Фиг.9. Общ вид на промишлени контролери за честотно управление на двигатели Altivar.

Литература

Интернет

1. Николов Л., Операционни системи, Издателство Сиела, София, 2001.
2. Луканчевски М., Системно програмиране за едночипови микрокомпютри, Техника, София, 1993.
3. Tanenbaum An., Modern Operating Systems, Prentice Hall, 2001.
4. Liu Jane, Real-Time Systems, Prentice Hall, 2000.
5. Li Qing, Real-Time Concepts for Embedded Systems, CMP Books, 2003.

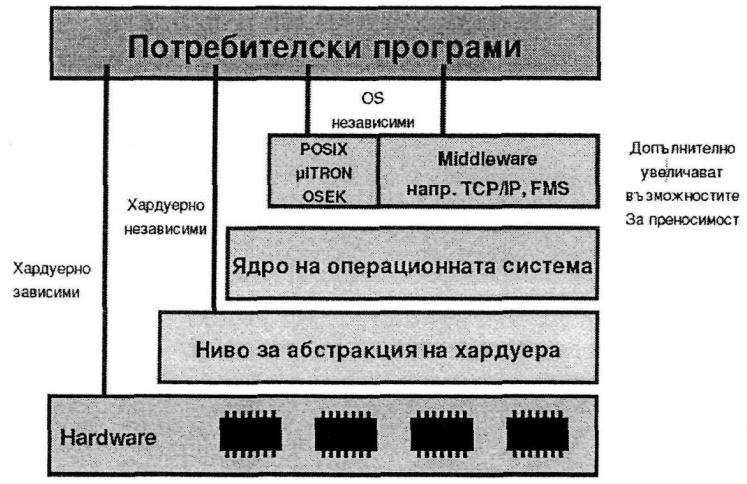
2. Операционна система

ОС представлява съвкупност от системни програми, които осъществяват връзката техническото обезпечение /хардуера/ и потребителските програми.

Характеристики

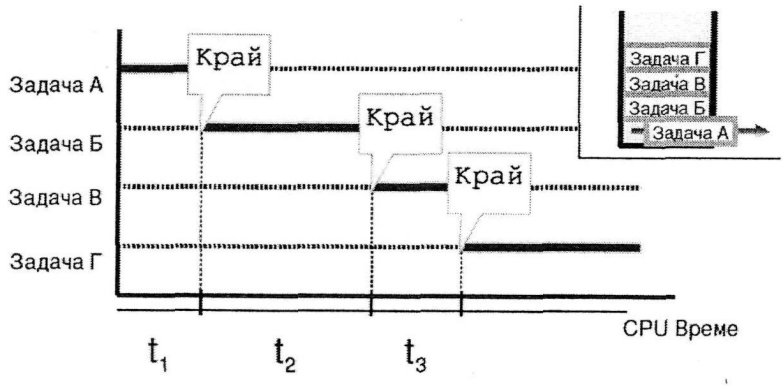
- Операционната система администрира и предоставя достъп до ресурсите на централният процесор, паметта и периферните устройства;
- Операционната система предоставя правилата и примитиви за управление, синхронизация и комуникация между потребителските програми;
- Отговаря за квази-паралелното изпълнение на множество задачи;
- Доставя на потребителя стандартизирани средства за достъп до вътрешни и външни за системата ресурси (файлови системи, комуникационни стекове и т.н.);
- Осигурява среда за тестване и дебъг.

3. Структурна схема на операционна система



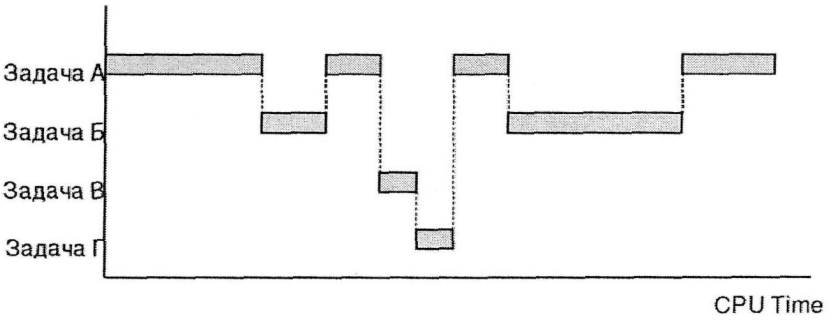
4. Начини за обработка на задачи в ОС

А. Последователно изпълнение



4. Начини за обработка на задачи в ОС

Б. Квази-паралелно изпълнение



С. Паралелно изпълнение - при многопроцесорни системи

5. Обработка на външни събития

Основните типове реакции са:

- Критична реакция в реално време (hard real time)
- Не критична реакция в реално време (soft real time)
- Обработка на събития не в реално време

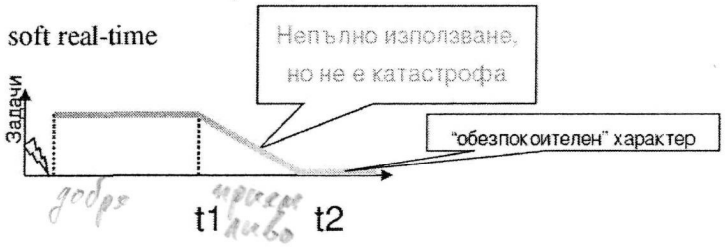
Системи за реално време са тези системи, при които цялостното коректно действие на системата зависи едновременно от нейната **функционална** и **времева коректност**.

Функционална коректност – правилност и точност на получените резултати в резултат на работата на системата.
Времева коректност – резултатите да бъдат получени в рамките на определен (предварително известен) период – т.е. изчисленията да завършат в рамките на този период.

Основни типове реакции на RTOS

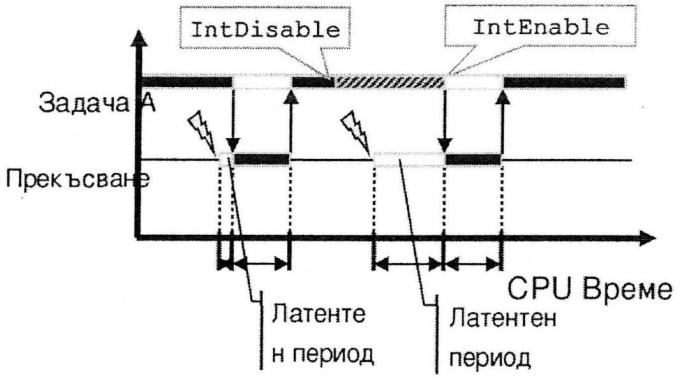
hard real-time

soft real-time



Разликата между **soft real time** и **hard real time** не е във времето за реагиране, а в това какви са последствията.

Обработка на прекъсванията при RTOS



В *първия случай* това е времето за приключване на инструкцията и запазване контекста на задачата.
Във *втория случай* имаме забрана на прекъсванията, което увеличава латентния период. .

6. Изисквания към операционните системи за вградени устройства

- обработка на събития в реално време
- синхронизация на събития
- голяма ефективност и бързодействие (органичен хардуер)
- модулна структура (премахване на ненужните компоненти)
- скалируемост (работа на платформи с различно количество ресурси и скорост)
- директен достъп до хардуера
- ефективна I/O система и драйверите
- предоставяне на специфични за вградените устройства стекове(CAN, DeviceNET ...)

6. Операционна система за реално време. Базисни системни обекти

Задача (Task, нишка)
Задачата е самостоятелна част от програмата, която се конкурира с другите задачи за процесорно време. Тя работи под управлението на операционната система за реално време.

Събитие
Събитието е основна комуникационна единица в операционната система за реално време. Задачите могат да изчакват настъпването на дадено събитие и да генерират настъпването на събитие.

Ресурс
Това е памет, входно/изходен канал, машинното време на процесора и др. Ресурсите са под управлението на операционната система за реално време. Всеки ресурс може да бъде свободен или зает. Задачата може да поиска от операционната система за реално време ресурс, да изчаква освобождаването му, да го използва и да го освобождава.

6. Операционна система за реално време. Базисни системни обекти

Пощенска кутия
Пощенската кутия е структура от данни в която едно съобщение се изпраща от една задача и се получава от друга. Задачите могат да поставят съобщение в пощенската кутия, да вземат съобщение от пощенската кутия и да проверяват състоянието ѝ - празна или не.
Семафор
Семафорът е флаг, който се използва за синхронизиране на задачите във времето. Има две състояния - вдигнато и спуснато. Задачите могат да установяват семафора във всяко от двете състояния и да проверяват текущото му състояние.
Системна единица за време
Системната единица за време е минималното време в рамките на операционната система за реално време и е равна на интервала между прекъсванията на таймера.

Особености при работа на OSRT

Наличие на общи ресурси
Пример за общи ресурси са структурите от данни, дисплеи, входно-изходните устройства и нереентрантните подпрограми, използвани от две или повече подзадачи. Те трябва да са защитени от едновременен достъп за повече от една подзадача.
Превключване между задачите и запазване на контекста
Когато една задача поеме управлението, необходимо е да се запазят системните регистри и някои специфични променливи за предишната задача, така че тя да може след време да продължи работата си от мястото в което е била прекъсната. Това се нарича *превключване на контекста* и е основна част от операционната система за реално време. Обикновено всяка задача има своя област в която запазва контекста си когато бъде прекъсната.
Диспечер
Диспечерът е тази част от ядрото, която определя коя задача да се стартира в следващия момент. Има различни подходи, но при всички случаи трябва да се осигури необходимото време за всички задачи. За тази цел съответните задачи трябва да се намират в състояние на готовност или активност.

Особености при работа на OSRT

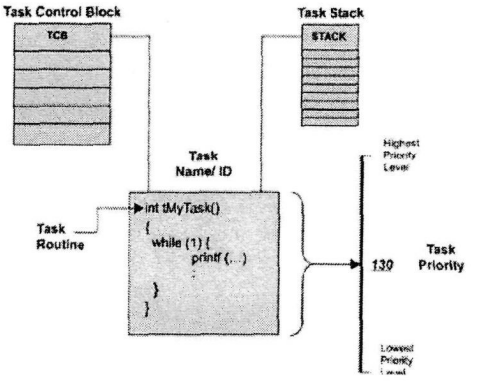
Реентрантност на подпрограмите
В т.нар. preemptive kernel (стратегия с изтласкване) две или повече задачи могат да искат да използват дадена подпрограма. Проблемът е в това. че не може да се предвиди момента, когато изпълнението на подпрограмата ще бъде прекъснато от другата задача. Необходимо е тогава подпрограмата да работи само с локални или пряко предавани променливи, които могат да бъдат запазени в стека на задачата при прекъсване. Такива програми се наричат реентрантни. Не е разрешено използването на глобални променливи или хардуерни ресурси в тялото на подпрограмата.
Приоритети на задачите - число, 1 число => 1 приоритет
Често се налага да се обработват данни или събития в точно определен момент или те ще бъдат изгубени. Тогава се налага да им се отреди по-голям приоритет, така че изпълнението им да бъде гарантирано в рамките на това време. Ядрото на операционната система трябва да може да осигури разпознаването на съответните приоритети и да подреди изпълнението на задачите.

Особености при работа на OSRT

Семафори и синхронизация
Семафорите се използват за синхронизация между задачите чрез предаване на съобщения между тях. Съществуват основно два типа семафори: двоични и броячни. Двоичните семафори позволяват достъп само на една задача до даден ресурс, всички останали задачи изискващи достъп до този ресурс ще бъдат поставени в състояние на изчакване. Броячните семафори поддържат списък със задачите, които трябва да бъдат обслужени.
Блокиране
Възможно е при определени ситуации да се достигне до взаимно блокиране на някои задачи. Например ако дадена задача се нуждае от определен ресурс за завършване на изпълнението си, а този ресурс е блокиран от друга задача, която пък чака завършването на първата, но не го освобождава. Това може да се преодолее с възприемането на динамични приоритети или с инверсия на приоритетите

8. Задачи /Tasks/

Атрибути на задачите
Задачите са обекти които подлежат на диспечеризация. Елементите на задачата са:
Име – Task Name / ID
Приоритет – Priory Level
Контролен блок на задачата – Task Control Block /TCB/
Стек – Task Stack
Функция (процедура)
Самата задача



Системни задачи

- **Инициализираща задача (startup/init)** – инициализира и създава останалите системни задачи.
- **Idle task** (задача за празен ход/незаета задача) – нейното съществуване е важно по 2 причини: да се използва CPU-то, когато няма други задачи Това е задача за изпълнение на специални действия, когато няма други задачи (например за изключване на някои устройства и намаляване на консумацията на енергия). Възможно е ядрото да предостави възможност ние сами да напишем функция и да укажем действията, които се изпълняват от idle задачата. Самата idle задача, обикновено има най-ниският възможен приоритет.
- **Задача за водене дневник на събитията** (logging task).
- **Задача за обработка на изключителните състояния**.
- **Задача на дебъг агента** – тя има за цел да осъществява комуникацията с хост дебъгера, да позволи настройката на софтуера и отстраняването на грешки.

Състояние на задачите

Състоянието на задачите е атрибут задаван и променян непрекъснато от операционната система за реално време.

Блокирано /Blocked/. Задачата в блокирано състояние съществува в паметта на компютърната система, но не участва в конкуренцията за получаване машинното време на процесора. Всички обръщения към нея се игнорират.

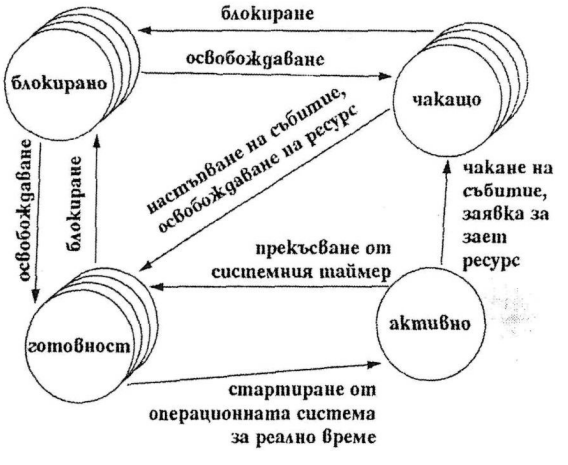
Чакащо /Waited/. Задачата в това състояние изчаква или настъпването на събитие или освобождаването на ресурс.

Готовност /Ready/. Задачата в състояние "готовност" има всички условия за изпълнението си и участва в конкуренцията за получаване на част от машинното време на процесора.

Активно /Running/. Задачата е в състояние "активно" когато се изпълнява от процесора. Броят на активните задачи в рамките на операционната система за реално време е равен на броя на процесорите, т.е. най-често само една задача може да бъде в това състояние.

Граф на състоянията на задачите

Графът на състоянията на задачите определя възможните преходи на задачите от едно състояние в друго. Преминаването на една задача от едно състояние в друго се определя от операционната система за реално време.

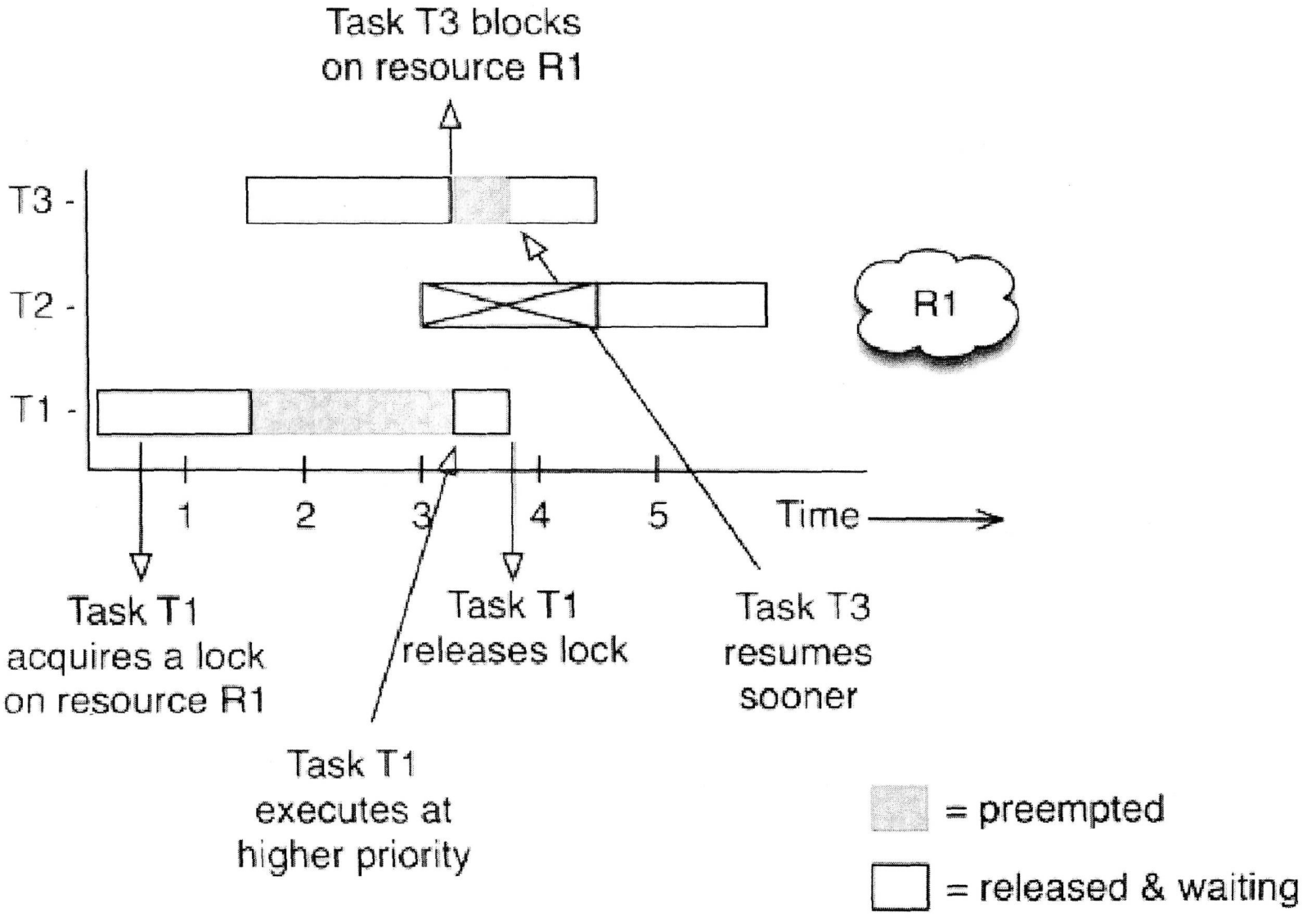


Действия на RTOS при работа със задачи

- създаване и изтриване на задачи
- контролиране диспечеризацията и реда на изпълнение
- получаване на информация за състоянието на задачите

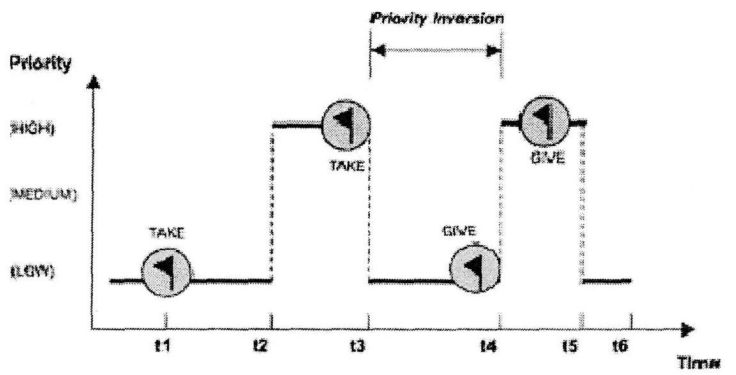
Ситуации възникващи при работа със задачи

Изтичане на ресурси (resource leak) - той се получава в случаите, когато една задача е заявила (и/или заделила) определен набор от ресурси, но не ги е освободила впоследствие. Причините може да са две: грешка при проектирането на самата задача или друга задача я изтрива/терминира без да й даде шанс да освободи ресурсите.Най-често изтичането на ресурси се проявява като изтичане на памет – заделяне на памет, без да се освободи тя след това. Така можем да достигнем до ситуация при която всички ресурси (от дадения тип) ще бъдат изчерпани.


Task T3 blocks
on resource R1
T3 -
T2 -
T1 -
R1
1
2
3
4
5
Time
Task T1
acquires a lock
on resource R1
Task T1
releases lock
Task T3
resumes
sooner
Task T1
executes at
higher priority
= preempted
= released & waiting

Ситуации възникващи при работа със задачи

Обръщане/Инверсия на приоритетите (Priority inversion) се получава в случаите, когато задача с по-нисък приоритет се изпълнява, докато задача с по-висок приоритет я изчаква защото тя държи определени ресурси.



Видове изпълнявани задачи

тип безкраен цикъл – обикновено са задачи които обработват входни и изходни данни. Веднъж създадени и стартирани, тяхното изпълнение продължава, до изричното им терминиране, или спиране на цялата система.

```
EndlessLoopTask ()
{
        Инициализация
        Условие за бекраен цикъл
        {
                Тяло на цикъла
                Едно или няколко блокиращи извиквания
        }
}
```

Видове изпълнявани задачи

тип "изпълнение до приключване на работата" обикновено са най-полезни при инициализиращи действия. Веднъж стартирана, задачата приключва работата (и съществуването) си, щом изпълни всички заложени действия. Типичната структура на този тип задачи има следния вид:

```
RunToCompletionTask ()
{
        Инициализация
        Създаването на задачи тип 'безкраен цикъл'
        Създаване обектите на ядрото
        Изтриване или прекратяване действието на тази задача
}
```

1. Структури от данни за комуникация между задачите

Заданията, работещи под управлението на операционната система за реално време, имат нужда да обменят данни помежду си, тъй като често са част от един конвейер за обработка на потока данни.

За целта системното програмно осигуряване трябва да осигури средствата за обмен на информация между заданията.

средства:

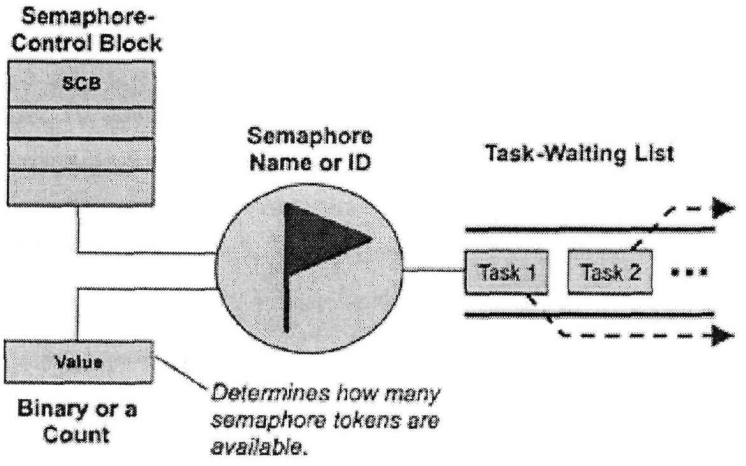
1.1. Семафор

Семафорът е обект на ядрото, който една или повече задачи могат да изискат или да освободят, с цел синхронизация или взаимно изключване.

Елементите на семафорите са:

- Контролен блок – Semaphore Control Block (SCB). Това е структурата, в която се пазят елементите, определящи еднозначно дадения семафор;
- Идентификатор – ID;
- Стойност – бинарна или целочислена;
- Списък на чакащите задачи

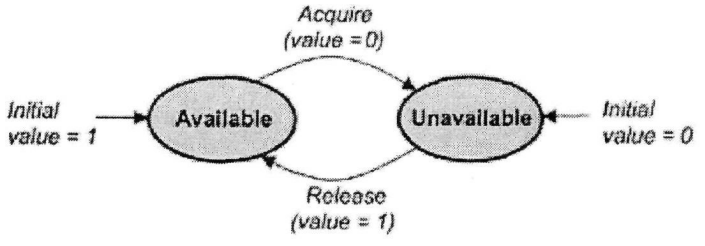
Структура на семафор



Бинарен семафор

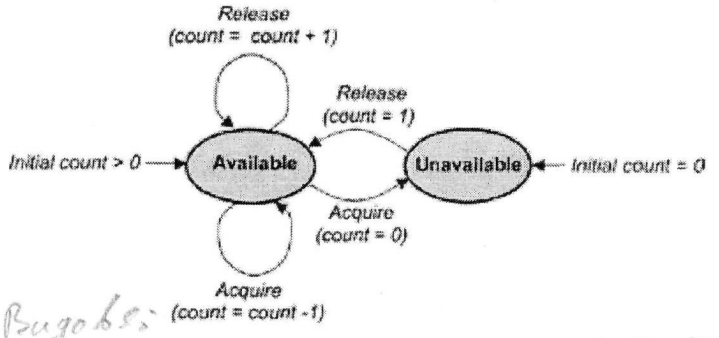
Бинарният семафор има само две състояния.
При стойност 0 семафорът се счита за "зает" (не свободен),
а при 1 за "свободен" ("пълен").

Диаграма на състоянията на бинарен семафор.



Броячен (counting) семафор

Броячният семафор има целочислена стойност и позволява да бъде заеман (респ. освобождаван) множество пъти (в зависимост от стойността).
При стойност 0 семафорът се счита за "зает" (не свободен), а при 1 за "свободен" ("пълен").
Диаграма на състоянията на броячния семафор.



Видове:
1. Ограничен (Не може да приеме ст-т > от число (3))
2. Неограничен (да тегли безкраен ресурс (памет)

рекурсивен

Семафор мютекс (Mutex – Mutual Exclusion)

Рекурсивното заключване е механизъм, при който една задача може многократно да изисква mutex върху, който има собственост. Mutex, който поддържа това свойство се нарича рекурсивен mutex.
Пример за ползата от това рекурсивно заключване е когато няколко функции от една и съща задача изискват съответния mutex.
Защитата от изтриване на задачата е механизъм, който не позволява задачата, която е собственик на mutex-а да бъде преждевременно изтрита (преди да го е освободила).

1 - заключено
0 - отключено

Типичните операции със семафори

➢ **Създаване и изтриване на семафорите.**

В зависимост от подържаните от ядрото на RTOS семафори се създават различни обекти и връзки между тях.

- **Binary –** определя състоянието на семафора и създава task-waiting order; – Задача чакаща
- **Counting** – задава броя и състоянието на семафора и създава task-waiting order; Ред на чакащите
- **Mutex -** създава task-waiting order и защитата от неправомерно изтриване на задачата, притежаваща в даен момент mutex-а.

Типичните операции със семафори

➢ **Изискване (достъп) и освобождаване на семафорите.**

Операциите по освобождаване и изискване на семафорите съществуват в три разновидности (в зависимост от указания период на изчакване):

- **Безкраен интервал на чакане –** задачата изчаква до момента в който операцията може да се извърши, без значение кога ще настъпи този момент.
- **Изчакване за определен период от време –** задачата остава блокирана до освобождаване на семафора или до изтичането на указан интервал от време. След изтичането на този интервал задачата се премахва от списъка с чакащи задачи и преминава в състояние на "готовност" или "изпълнение".
- **Без изчакване –** задачата се опитва да изпълни операцията по изискване, но ако достъпът е забранен, тя се отказва и не изпада в блокирано състояние

Типичните операции със семафори

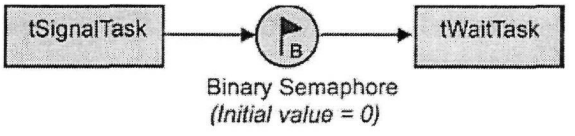
- **Изчистване на списъка с чакащите задачи.**
- **Получаване на информация за състоянието на семафора -** получената информация за състоянието е моментна снимка (аналогично на задачите) и е динамична.

ISR (Процедурите по обработка на прекъсвания) също могат да освобождават бинарни и броячни семафори. Те **НЕ могат** да заключват или отключват мютекси**!**

Употреба на семафорите

- **Изчакване и сигнализиране – Wait-and-Signal**

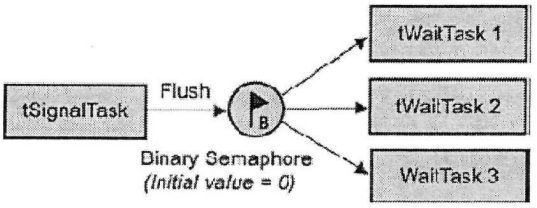
Използва се когато се налага две задачи да си комуникират без да обменят данни помежду си. Например само да се „договорят" за момента на трансфер на контрола.
В този случай се използват бинарни семафори.



Тъй като приоритета на чакащата задача е по-висок, веднага след "освобождаването" на семафора тя преминава в състояние на "изпълнение".

Употреба на семафорите

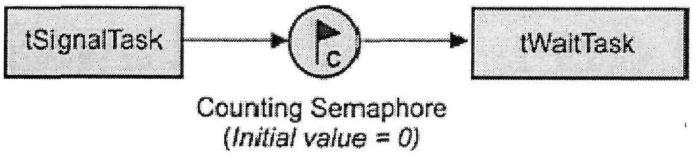
➢ **Изчакване и сигнализиране на множество от задачи.**
При синхронизиране работата на повече от две задачи се използва **Flush operation** с бинарен семафор над task-waiting list.



Диспечера избира коя задача да се изпълни при освобождаване

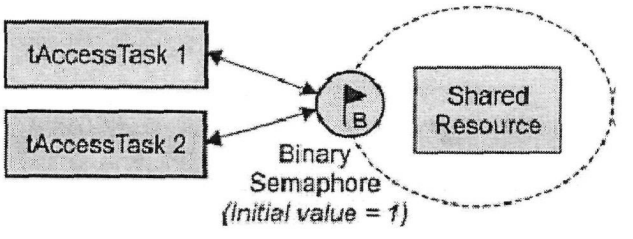
Употреба на семафорите

➢ **Проследяване на събития – Credit-Tracking Synchronization.**
В някои случаи честотата на изпълнение на сигнализиращата задача е по-висока от тази на обработващите задачи. Тогава се използва броячен семафор. Той брои постъпващите сигнали (събития) и изчаква последващата им обработка.



Употреба на семафорите

➢ **Синхронизация на достъпа до единичен споделен ресурс.**

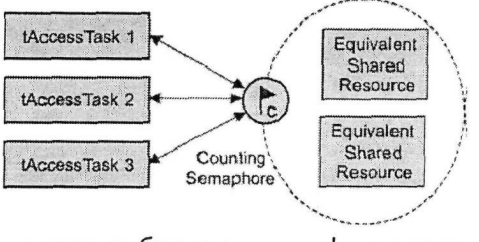


На фигурата е показана реализация на сценария с бинарен семафор. В зависимост от състоянието на ресурса семафора се инициализира с 0 или 1. След това задачите 1 и 2 съответно го изискват или освобождават, при което само една от тях има достъп в момента до ресурса.

Употреба на семафорите

➢ **Синхронизиране достъпа до множество споделени ресурси.**

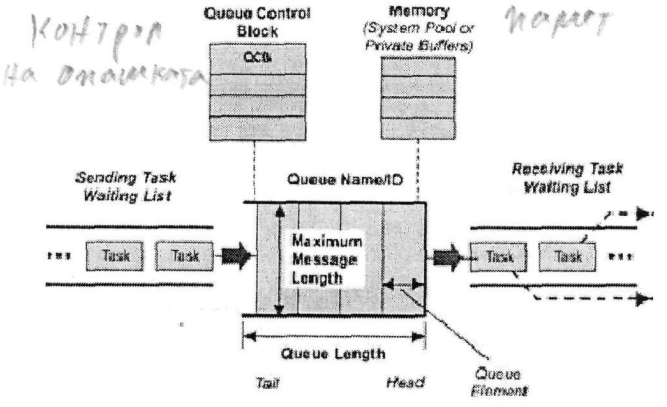
Това е сила **САМО** ако ресурсите са идентични.



Използва се броячен семафор, чиято стойност е установена на максималния брой на идентичните ресурси. Използването на ресурс е изискване на семафора (т.е. намаляване стойността му), а приключването на работата с ресурс е освобождаване (увеличаване стойността на семафора).

1.2. Опашки със съобщения /Message Queues/ Дефиниране на опашка със съобщения.

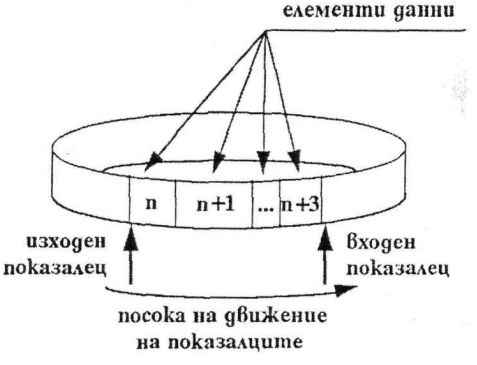
Опашка със съобщения е буфер, в който задачите и ISR изпращат и приемат съобщения (данни).



Опашката притежава няколко компонента, които се използват от ядрото. При първоначалното създаване на опашката се инициализират **Queue Control Block**, идентификатор (**Queue Name - ID**), буферна памет, дължина на буфера и чакащите задачи.

1.2. Опашки със съобщения /Message Queues/ Дефиниране на опашка със съобщения.

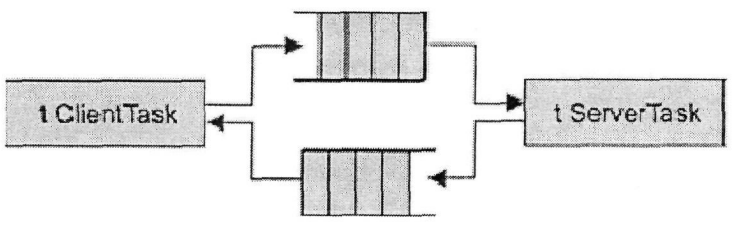
Най-често опашката се управлява като кръгов буфер с указатели – входен и изходен. По своята същност това е един стек FIFO (first in first out - първи влязъл, първи излязъл). Когато опашката е празна входният и изходният указатели имат еднакви стойности.



1.2. Опашки със съобщения /Message Queues/

Дефиниране на опашка със съобщения.

Опашката обикновено реализира връзка между две задания само в една посока - едното задание изпраща съобщения, другото ги чете и обработва. За реализиране на двупосочна връзка са необходими две опашки.



1.2. Опашки със съобщения /Message Queues/

Опашката е важно средство за осигуряване на максимално използване на времето на процесора от заданията и служи като буфер между заданията при случаите на различен темп на постъпване и обработка на съобщенията.

Основните примитиви за работа с опашката са:

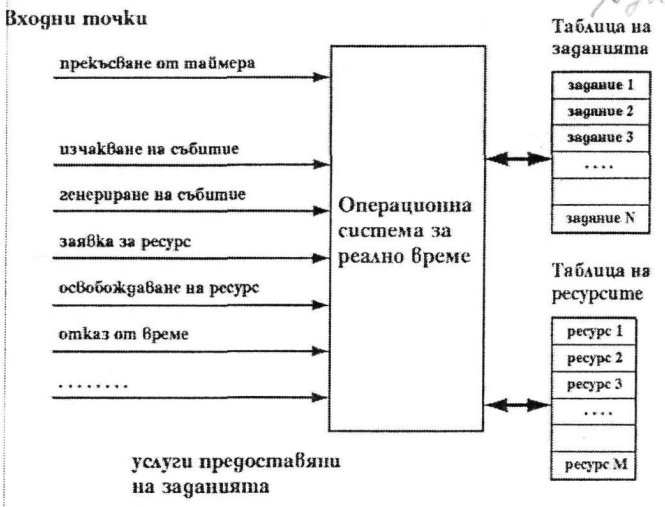
- Въвеждане на елемент данни в опашката;
- Извеждане на елемент данни от опашката.

Освен основните примитиви съществуват и **допълнителни**:

- проверка за празна или непразна опашка;
- копиране на първия елемент данни от опашката, като той остава в нея;
- изхвърляне на първия елемент от данни.

2. Функции и реализация на RTOS

Обобщена блокова схема на RTOS

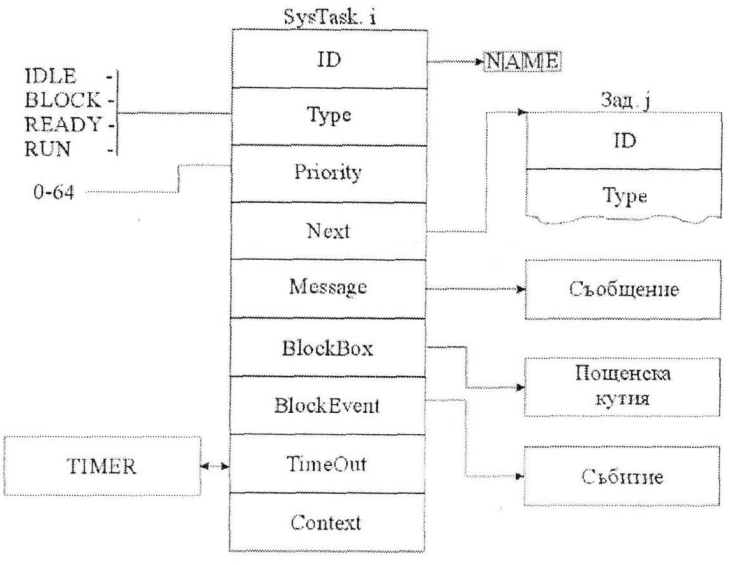


Входни точки

Операционната система за реално време е програма, която управлява разпределението на ресурсите и обслужва заявките от страна на задачите. За тази цел тя трябва да има определени входни точки.

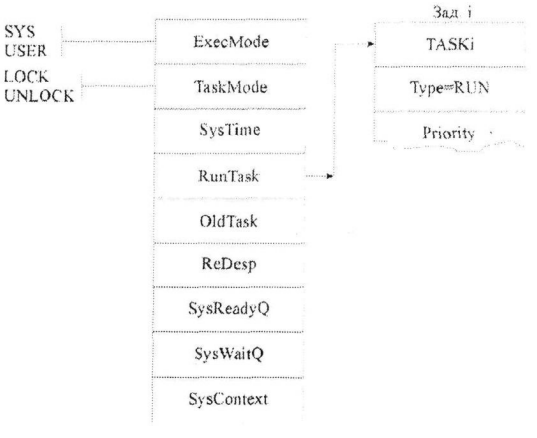
Основна входна точка е обслужването на прекъсването от системния таймер. След всяко прекъсване управлението се предава на тази точка и се извършват дейности по избор на задача, на която да се предостави процесорното време.
Останалите входни точки са заявките за услуги, които предоставя операционната система за реално време - искане на ресурс, генериране на събитие и т.н.
Достъпът до входните точки за заявки може да бъде програмно организиран различно за различните процесори, но най-често става чрез механизма на програмните прекъсвания.

Структурата на дескриптора на задачите



Системен процес

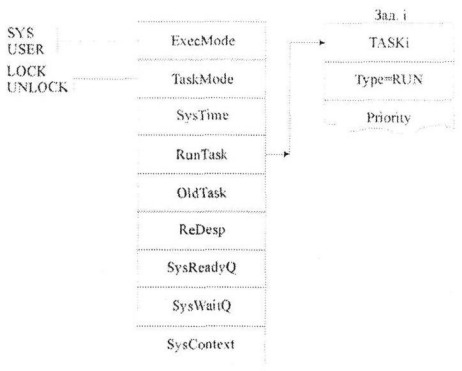
Преди създаването на известен брой приложни задачи в системата се стартира един системен процес, който в последствие поражда приложните задачи.
Той не се преустановява с пораждането на приложните задачи. Той продължава да се изпълнява паралелно с породените от него задачи.
По аналогия с приложните задачи, системния процес притежава също своя системна таблица – дескриптор.



Системният дескриптор (дескриптора на ядрото), за разлика от тези на приложните задачи няма свой собствен идентификатор, поради факта, че е единствен в системата.

Системен процес

Първото поле от системния дескриптор служи за указване на текущия режим на работа. Той може да бъде :
Потребителски (USER) – няма забранени прекъсвания, има предпоставки за диспечеризация на задачите от всички опашки, както и на задачата, която е активна в момента. Тя може да бъде превключена при изтичане на системния квант от време.
Системен (SYS) – в момента се извършват системни манипулации и активния процес е системният.



Полето TaskMode задава режимът на активната задача. Обикновено активната задача е "отключена" (UNLOCK). По този начин механизма на преразпределение на времето между задачите е разрешен. Текущата задача може да бъде заместена от друга при изтичане на системния интервал от време.

Реализация на RTOS

Обикновено операционните системи за реално време се реализират на асемблер за съответния процесор. Все по-често напоследък се срещат и операционни системи за реално време, реализирани на езика С. Той съчетава предимствата на езиците от високо ниво (бързо написване и настройка, кратка и прегледна програма) с предимствата на програмите, написани на асемблер (бързина в изпълнението и малък обем).
Операционните системи за реално време не са големи програми като брой на операторите поради изискването да са бързи и поради ограничения брой услуги, които обикновено предоставят на заданията.

все по-често С

3. Изисквания при избор на RTOS

Изборът на операционна система за реално време има съществено значение за определяне на характеристиките на цялостната система, а именно: нейната функционалност, производителност, надеждност, време за реализация на пазара и цена. Проектирането на системи с висока надеждност е още по-труден и отговорен процес и изисква внимателен избор на операционна система.

➢ **Бързодействие на системата.**
В приложения с висок коефициент на сигурност е изключително важно да се направи оценка на времевите параметри на системата, т.е. да се определи нейното бързодействие. Следва да се изчисли не само времето за свързване на RTOS и за изпълнение на програмния код на отделните нишки, но и допълнителното време, асоциирано към всяка нишка, а именно времето за превключване на контексти, за изпълнение на системни заявки, времена от прекъсвания и за изпълнение на процедурите за прекъсвания.

3. Изисквания при избор на RTOS

➢ **Защита на паметта.**
Толерансът към възникване на грешки, а следователно и надеждността на една система зависят основно от методите за защита на паметта. В момента микропроцесорите разполагат с вградено в чипа устройство за управление на паметта (MMU – memory management unit), което позволява на отделните задачи да работят в хардуерно защитено пространство памет.

➢**Толеранс на грешките и достъпност.**
Дори и в най-добрия софтуер има скрити, латентни грешки.
Колкото по-сложно става дадено приложение, колкото повече функции се имплементират в него, толкова повече нараства възможността за грешки, вградени в самата RTOS. За целта операционната система следва да предоставя механизми за откриване на грешки и възстановяване на системата.
Естествено начините за възстановяване на системата след грешка е силно зависима от самото приложение.

3. Изисквания при избор на RTOS

➢ **Права за достъп до системни ресурси.**
За системи с критична надеждност е от съществено значение процесът с основна важност винаги да разполага с необходимия му ресурс памет. В повечето RTOS, паметта за поддържане на контролния блок на всяка задача или друг обект се разпределя от общ централен ресурс. Когато обаче има грешка или вирус в задачата може като резултат да се стигне до ситуация, в която тази задача създава много голям брой обекти, всеки от които иска обем памет и се изчерпват ресурсите на системата.

➢ **Гарантиран достъп до ресурс "време на процесора".**
При повечето RTOS се използва приоритетно базирания принцип на разпределение на процесорното време. [2,4,7] При тази схема, нишката с най-висок приоритет получава процесорно време за изпълнение. Когато има повече от една нишка с един и същи приоритет, се използва принципът на времеделение на малки порции, които се предоставят на всяка нишка равностойно. При този механизъм няма гаранция за обезпечаване на критичните задачи вътре в даден приоритет.

Дилов

09.4.2012 г.

Взаимодействие между задачите (процесите)

Изпълняваните процеси (задачи) могат да бъдат:

- **Независими процеси** – работят с множества от независими променливи и работата на такъв процес не оказва влияние върху резултата от изпълнението на друг независим процес и обратно.
- **Взаимодействащите процеси** – имат достъп до общи променливи и изпълнението на един процес влияе върху резултата от изпълнението на друг.

Асинхронността на работа на паралелните процеси и различията в скоростта на изпълнението им усложнява взаимодействието между процесите.

Взаимодействие между задачите (процесите)

Въпреки че физическите и логическите ресурси могат да бъдат споделяни, обикновено във даден момент от време те са достъпни **само за един процес**. Такива ресурси се наричат критични.

Ако няколко процеса искат да използват критичен ресурс, те трябва да съгласуват действията си във времето така, че в даден момент ресурсът да бъде използван само от един процес – останалите процеси трябва да чакат, докато се освободи ресурсът. Това е същността на понятието **взаимно изключване на процеси.**

1. Взаимно изключване на процеси

Коректното решаване на проблема на взаимното изключване трябва да отговаря на следните критерии:

1. Само един процес може да използва ресурса в даден момент.
2. Ако няколко процеса едновременно желаят ресурса, той трябва да бъде предоставян последователно на всеки един от тях за краен интервал от време.
3. Ако един процес получи ресурс, той трябва да го освободи за крайно време.

Никакви други предположения относно относителните скорости на изпълнение на процесите не се правят.

Пример

Например два процеса работят с общи данни, при което първият процес увеличава съдържанието на обща променлива: **c:=c+1**, а вторият го намалява: **c:=c-1**. Предполагаме, че тези оператори се транслират в следните последователности от процесорни команди:

```
Процес 1: LDAA  c      Процес 2: LDAB  c   {зареждане на акумулатор A/B}
          INCA                    DECB      {увеличаване/намаляване A/B}
          STAA  c                 STAB  c   {запомняне на акумулатор A/B}
```

Ако тези последователности от команди се изпълняват в непресичащи се интервали от време, ще се получат коректни резултати. Обаче е възможна следната ситуация (нека стойността на общата променлива **c** в началото е равна на 4):

```
t0:   Процес 1 – LDAA  c      {Acc.A=4}
t1:   Процес 1 – INCA         {Acc.A=5}
t2:   Процес 2 – LDAB  c      {Acc.B=4}
t3:   Процес 2 – DECB         {Acc.B=3}
t4:   Процес 1 – STAA  c      {c=5}
t5:   Процес 2 – STAB  c      {c=3}
```

Накрая се получава некоректен резултат **c=3** (вярното е **c=4**).

Решение на проблема на взаимното изключване на процеси се състои в определяне на **критичната секция** във всеки от взаимодействащите си процеси.

Критичната секция е област на процеса, в който той работи с критичния ресурс (общите данни също могат да се разглеждани като ресурс).

За да се реши задачата за взаимното изключване, трябва да са изпълнени следните **условия**:

1. Само един процес може да се намира в критичната си секция в даден момент, т.е. изпълненията на критичните секции взаимно се изключват във времето.
2. Процес може да остане в критичната си секция крайно време.
3. Ако един процес иска да влезе в критичната си секция, трябва да му се даде такава възможност в крайно време. В частност, процес намиращ се извън критична секция, не може да пречи на другите процеси да влязат в критичните си секции.

За решаването на този проблем са предлагани много подходи, един от които е **програмното решение**. Програмното решение се базира на блокировка на паметта.

Използва се конструкцията **parbegin/parend (cobegin/coend),** която има следния смисъл.

Нека е дадена последователността:

$S_0; parbegin; S_1; S_2; ...; S_n; parend; S_{n+1};$

където S_i може да е оператор, извикване на процедура, блок или тяхна комбинация, които за кратко ще наричаме оператор. Операторите заключени между скобите **parbegin/parend** , се изпълняват паралелно и S_{n+1} към се преминава, когато се изпълнят всички оператори в тези скоби.

2. Взаимно изключване на два процеса
Първи вариант – твърда синхронизация

На фигурата е дадено решение, при което процесите представляват **безкраен цикъл**, а влизането в **критична секция** се управлява от оператор **while...do;**, повтарящ се, докато променливата **ред_на** не стане равна на номера на процеса

Процесите стриктно се редуват при влизане в критичната секция и ако единият процес блокира, след като излезе от критичната си секция, то и другия ще блокира след излизане от критичната си секция.

```
program взаимно_изключване;
  var ред_на: integer;
  procedure процес_1;
    begin
      repeat
        begin
          while ред_на ≠ 1 do;   {вход в критична секция}
          критична секция
          ред_на := 2;           {изход от критична секция}
          .....
        end;
      until false;
    end;
  procedure процес_2;
    begin
      repeat
        begin
          while ред_на ≠ 2 do; {вход в критична секция}
          критична секция
          ред_на := 1;           {изход от критична секция}
          .....
        end;
      until false;
    end;
  begin
    ред_на := 1;
    parbegin процес_1; процес_2; parend;
  end.
```

Втори вариант

Тук променливата **ред_на** се заменя с две логически променливи: **ред_на_1** и **ред_на_2**. Те имат стойност **true**, когато съответния процес (първи или втори) **е в критичната си секция**, и **false**, когато не е. Началната им стойност е false.

```
procedure процес_1;
  begin
    repeat
      while ред_на_2 do;
      ред_на_1 := true;
      критична секция
      ред_на_1 := false;
    until false;
  end;
```

```
procedure процес_2;
  begin
    repeat
      while ред_на_1 do;
      ред_на_2 := true;
      критична секция      { вход }
      ред_на_2 := false;
    until false;           { изход }
  end;
```

Недостатък:

Алгоритъмът не осигурява изискването само един процес да е в критична секция. Защо?

Трети вариант

За отстраняване на този проблем всеки процес ще установява флага си, преди да влезе в цикъла, т.е. **ред_на_1=true** само индицира, че **процес_1 иска да влезе в критична секция**

```
procedure процес_1;
  begin
    repeat
      ред_на_1 := true;
      while ред_на_2 do;
      критична секция
      ред_на_1 := false;
      .....
    until false;
  end;
```

```
procedure процес_2;
  begin
    repeat
      ред_на_2 := true;
      while ред_на_1 do;
      критична секция
      ред_на_2 := false;
      .....
    until false;
  end;
```

Недостатък:

Възможно е и двата процеса да вдигнат флаговете, преди да влязат в цикъла. Защо?

Четвърти вариант

За да се избегне мъртвата **хватка трябва да се разбият циклите** във вариант 3. Въвежда се периодически нулиране на флага във false и кратковременна задръжка на всеки процес влязъл в цикъл на очакване.

```
procedure процес_1;
  begin
    repeat
      ред_на_1 := true;
      while ред_на_2 do
        begin
          ред_на_1 := false;
          случайна задръжка
          ред_на_1 := true;
        end;
      критична секция
      ред_на_1 := false;
      .....
    until false;
  end;
```

```
procedure процес_2;
  begin
    repeat
      ред_на_2 := true;
      while ред_на_1 do
        begin
          ред_на_2 := false;
          случайна задръжка
          ред_на_2 := true;
        end;
      критична секция
      ред_на_2 := false;
      .....
    until false;
  end;
```

вход

{*изход*}

По този начин се гарантира взаимното изключване и се избягва мъртвата хватка, но възниква друг проблем – безкрайното отлагане.

Пети вариант

Следващото решение се основава на вариант 3 и на известна модификация на вариант 1.

Ако два процеса искат да влязат в критична секция по едно и също време, променливата **процес_номер** ще се установи два пъти, но кой ще влезе ще се реши от последната ѝ стойност. Взаимното изключване е осигурено от условието в **while**. Ако и двата процеса искат да влязат в критична секция, то **ред_на_1=ред_на_2=true**, но **процес_номер** може да има само **една стойност 1 или 2**. От друга страна, условието не се променя докато процес е критичната секция.

```
program взаимно_изключване;
  var ред_на_1, ред_на_2: boolean;
  процес_номер: integer;
    procedure процес_1;
      begin
        repeat
          ред_на_1 := true;                                  { вход в
          процес_номер := 2;                                   критична
          while (ред_на_2 and процес_номер=2) do;              секция }
          критична секция
          ред_на_1 := false;      {изход от критична секция}
          .....
        until false;
      end;
    procedure процес_2;
      begin
        repeat
          ред_на_2 := true;
          процес_номер := 1;
          while (ред_на_1 and процес_номер=1) do;
          критична секция
          ред_на_2 := false;
          .....
        until false;
      end;
  begin
    ред_на_1 := false; ред_на_2 := false;
    parbegin процес_1; процес_2; parend;
  end.
```

процес №, инвариант на procedure process

2. Взаимно изключване на n процеса

При **n** процеса взаимното изключване се основава на **планиращите алгоритми** (bakery algorithm).

Всеки купувач получава талон, на който се записва поредния му номер. Всеки талон има и уникален фабричен номер. Обслужва се купувач с най-малък номер, но ако се случи така, че два купувача (респективно процеса) имат един и същи номер, ще се обслужи първо този с по-малкия фабричен номер. Тъй като фабричните номера са уникални и подредени, алгоритъмът става детерминиран.

Взаимно изключване чрез операциите P и V

Реализирането на критични секции с помощта на блокировка на паметта осигурява взаимно изключване, но е сложно и неефективно, особено при повече от два процеса. За преодоляване на горните трудности Дейкстра въвежда две примитивни операции **P** и **V** , които оперират над цели променливи, т.е. семафори.

Класическата дефиниция на P и V е:

P(s): *while s=0 do skip ;*

s=s-1;

Извиквайки **P(s)**, процесът чака при стойност на семафора (изпълнява празен цикъл), докато друг процес не го освободи чрез **V(s)**. Тогава стойността на **s** се намалява и процесът продължава.

V(s): *s=s+1 ;*

Стойността на семафора **s** се увеличава и процесът продължава.

Освен това трябва да има инициализация на семафора **s** чрез операция. *s:=<цяла константа>*.

P и **V** изпълняват ролята на скоби, които задействат критичните секции. Началната стойност на семафора **s=1**.

3. Синхронизация

Между процесите, кооперирващи се за изпълнение на обща работа, взаимодействието е пряко. На тях им е необходимо не само да се изключват при работа с общи ресурси, но и да си разменят информация, за да синхронизират работата си във времето.

Минималната единица предавана информация е прост времеви (синхронизиращ) сигнал.

Пример за този вид **взаимодействие** е, когато процесите се намират в отношение **производител-потребител**, т.е. единият процес произвежда ресурс, а другия го използва.

Тези действия се извършват асинхронно, затова има опасност производителят да се опита да запише в пълен буфер или потребителя да се опита да чете от празен (или вече прочетен) буфер.

Събития

Основното изискване при синхронизацията на процесите е: на единия процес се дава възможност да чака, докато друг процес не му съобщи за настъпването на определено събитие. Първият процес чака, изпълнявайки операция **wait** (чакане). Вторият процес сигнализира за настъпване на събитието чрез операция **signal** (събитие).

Също така трябва да се предвиди процедура за **начално установяване на опашката** и на **флага за състоянието** на събитието. Полезно е да се включат и други операции, като например за **проверка на текущото състояние на събитие**, за **модификация на флага** и др.

Пример: Производител - Потребител

В примера за **производител – потребител** събитията могат да бъдат **запълнен_буфер** и **прочетен_буфер**. Производителят запълва буфера (глобална променлива) и сигнализира за събитието **запълнен_буфер**, което позволява на потребителя, чакащ събитието да започне разпечатването на буфера.

Аналогично потребителят, след прочитане на буфера, съобщава за събитието с **прочетен_буфер**, което позволява на производителя, очакващ настъпването на това събитие, да запълни отнова буфера.

Включена е и операцията **reset** за нулиране на флага на събитието.

```
program производител_потребител;
 var буфер: T;                                   {произволен тип}
     запълнен_буфер, прочетен_буфер: event;  {тип „събитие"}
 procedure производител;
   var буф: T;
   begin
      repeat
            производство на данни в буф;
            буфер := буф;
            signal(запълнен_буфер);
            wait(прочетен_буфер);
            reset(прочетен_буфер);
      until false;
   end;
procedure потребител;
   var буф: T;
    begin
      repeat
            wait(запълнен_буфер);
            буф := буфер;
            потребление на данни от буф;
            reset(запълнен_буфер);
            signal(прочетен_буфер);
      until false;
   end;
begin
   initialize(запълнен_буфер, false);
   initialize(прочетен_буфер, false);
   parbegin производител; потребител parend;
end.
```

Използване на операциите P и V за синхронизация в предния пример

С помощта на операциите **P** и **V** лесно се решава проблемът **производител – потребител**. Синхронизацията се извършва, като за всяко събитие се използва двоичен семафор, операция **P** – вместо **wait** и операция **V** – вместо **signal**.

```
program производител_потребител;
  var буфер: integer;
    пълен, празен: semaphore;
  procedure производител;
    var буф: integer;
    begin
      производство на данни в буф;
      буфер := буф;
      V(пълен);
      P(празен);
    end;
  procedure потребител;
    var буф: integer;
    begin
      P(пълен);
      буф := буфер;
      потребление на буф;
      V(празен);
    end;
  begin
    initialize(пълен, 0); initialize(празен, 0);
    parbegin
      while true do производител;
      while true do потребител;
    parend;
  end.
```

Класически проблем: Писател - Читател

Данни (например файл или запис) могат да се използватот няколко процеса. Някои процеси искат само да четат общите данни, докато други искат да обновяват данните, т.е. могат да се различат два типа процеси, наречени **писатели** и **читатели**. Ако двама читатели искат да четат общите данни едновременно, няма никакви проблеми. Писател и читател не могат да имат едновременен достъп до данните.

Изисквания:

- Читател да не се държи чакащ, освен ако писател вече не е получил право да разполага с общия обект.
- Готов писател да извършва запис веднага, щом като е възможно, т.е. ако писател е чакащ за достъп, никакъв нов читател не може да стартира.

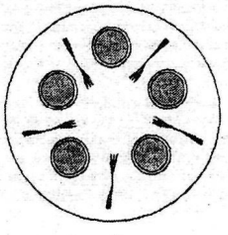
Семафорът **запис** е общ за читатели и писатели. Той осигурява взаимното изключване на писателите, а също така служи на първия/последния читал да влезе/напусне критичната секция.

Семафорът **взаимно_изкл** се използва за осигуряване на работата с променливата **брой_читатели**, която дава броя на четящите процеси.

```
program читатели_и_писатели;
  var брой_читатели: integer;
    взаимно_изкл, запис: semaphore;
  procedure читател;
    begin
      repeat
        P(взаимно_изкл);
        брой_читатели := брой_читатели + 1;
        if брой_читатели=1 then P(запис); {Първият влиза, респ.чака}
        V(взаимно_изкл);          {И други читатели могат да влязат.}
        четене на данни;          {за да четат}
        P(взаимно_изкл);
        брой_читатели := брой_читатели – 1;
        if брой_читатели=0 then V(запис);   {Последният читател}
                                            {излиза}
        V(взаимно_изкл);
      until false;
    end;
  procedure писател;
    begin
      repeat
        P(запис);
        запис на данни;
        V(запис);
      until false;
    end;
  begin
    initialize(взаимно_изкл, 1);
    initialize(запис, 1);
    брой_читатели := 0;
  parbegin писател; читател; parend;
  end.
```

Класически проблем: Обядващи философи

Пет философа прекарват живота си в мислене и ядене. Те сядат около кръгла маса, на която има 5 чинии със спагети и 5 вилици. Когато философът мисли, той не контактува с колегите си. От *време на време огладнява и се опитва да вземе две вилици, които са около чинията си.* Философът може да вдигне само една вилица (без значение лява или дясна) в даден момент . Очевидно, той не може да вземе вилица, която е в ръката на негов съсед. Когато филосов успее да вземе две вилици, той започва да яде, без да ги освобождава. Когато свърши яденето, слага двете вилици от двете страни на чинията и започва да мосле отново.



```
Общите данни са:
    var вилица: array[0..4] of semaphore;
    initialize(всеки семафор, 1);
Структурата на i-тия философ е следната:
    repeat
        мисли;
        P(вилица[i]);
        P(вилица[(i+1) mod 5]);
        яде;
        V(вилица[i]);
        V(вилица[(i+1) mod 5]);
    until false;
```

Едно решение е да се *представи всяка вилица със* семафор. Философът се опитва да вземе вилица чрез P-операция, и я освобождава чрез V-операция. Решението гарантира, че двама съседи няма да ядат едновременно, но трябва да се отхвърли, защото *може да възникне* **мъртва хватка**.

Например, ако петимата философи огладнеят едновременно и всеки вземе своята лява вилица. Всички елементи на вилица са 0. Когато философ се опита да вземе дясната вилица, той ще се блокира завинаги.

Модификация: *Философът след като вдигне лявата вилица, проверява дали е свободна* и дясната. Ако не е, той лявата вилица на масата, чака известно време и пак се опитва да вземе вилиците. При това решение се избягва мъртвата хватка.

Локални компютърни мрежи в системите за управление

Системите за управление с мрежова комуникация (Networked Control System – NCS) са такива системи, при които управляващите и крайните устройства (сензори, измервателни преобразуватели, изпълнителни механизми, регулиращи органи, регулатори, контролери, операторски станции и др.) са свързани в мрежа.

Интерфейс и мрежа

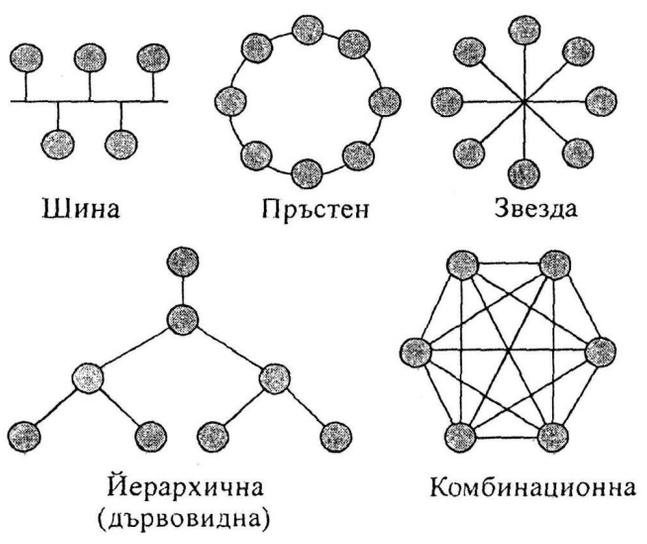
1. **Интерфейсът** е съвкупност от програмни и технически средства за осъществяване на връзка между две или повече устройства за обмен на данни и сигнали. Най-често под интерфейс се разбират средствата за свързване на две устройства и следователно той не е спецификация за мрежи в общоприетия смисъл, когато в тях има повече от две устройства Например серийният интерфейс RS232 специфицира само електрическите сигнали, конектори и кабели, но не и какво ще се предава като данни, кадри и др.

2. **Мрежата** е комуникационна система, свързваща устройства с цел поделяне на ресурси и обмен на данни. Тя позволява едно устройство (станция) да получи достъп до ресурсите на друго, които са свързани в една мрежа. Модел на едно мрежово устройство е например известният OSI-модел, а мрежите подробно са описани в своите спецификации, които понякога са и стандарти (напр. IEEE 802.3, Profibus-DP).

Мрежови топологии и модели

Топологията дефинира структурата на мрежата. Има два вида топология – **топология на физическите връзки** (физическа топология) и **логическа топология**. Физическата топология се отнася за същинската конфигурация на свързващите проводници, мрежовото оборудване и потребителските станции в мрежата. Логическата топология определя по какъв начин потребителите получават достъп до мрежата и възможните пътища за обмен на данни между мрежовите възли. Най-често използваните топологии са: шинна (bus), пръстен/кръг (ring), звезда (star), дървовидна или йерархична (tree, hierarchical) и комбинационна, които са показани на фигурата.

Мрежови топологии и модели



Мрежови топологии и модели

Най-елементарното устройство за физическо свързване на отделни сегменти от мрежата е **повторителят (Repeater).** С повторители се увеличава общата дължина на комуникационната мрежа чрез възстановяване на мощността на сигнала и подобряване на фронтовете на импулсите.

Концентратор (Hub) се нарича многовходов повторител, с помощта на който се осъществява свързване от тип звезда. Прибавянето на концентратори изменя физическата топология на мрежата, но не променя нейната логическа структура, защото действието на тези устройства е на елементарно физическо ниво и не влияе на пътя на съобщенията.

За влияние върху логическата структура на комуникационните мрежи се изпол¬зват устройства като мостове, комутатори, маршрутизатори и шлюзове.

Мостът (Bridge) е устройство, с което мрежата се разделя на логически сегменти и съобщенията се предават от единия към другия сегмент през порт на моста само ако това е необходимо, т. е. ако физическият адрес на получателя принадлежи към съответния сегмент. По този начин мостът изолира трафика на даден сегмент от трафика на останалата част и повишава общата пропускателна способност на мрежата.

Мрежови топологии и модели

Комутаторът (Switch) е комуникационен мултипроцесор. За разлика от моста, който използва един процесор, комутаторът разполага с по един специализиран процесор за всеки свой порт, който обработва постъпващите кадри независимо от процесорите на останалите портове. Комутаторите може да се считат за мостове от ново поколение с паралелна обработка на информацията.

Маршрутизаторът (Router) извършва изолиране на трафика по-ефективно от моста, използвайки съставни числови адреси на отделните възли. Всички възли с еднакво поле за номер на мрежата в своя адрес образуват подмрежа. Маршрутизаторите се използват и за свързване в единна мрежа на подмрежи от различни мрежови технологии – например Ethernet и X.25.

Шлюзове (Gateways) се използват за обединение на мрежи с различни типове системно и приложно програмно осигуряване. При това обединение като страничен ефект се получава и естествено изолиране и локализиране на трафика в различните мрежи от двете страни на шлюза.

Мрежов модел - Open System Interconnect (OSI)

При мрежовите комуникации се извършват множество обработки, контрол, кодиране, действия над единиците с пренасяна информация. Затова е необходим някакъв формализиран начин за описание (моделиране) на различните етапи (слоеве) и фази, през които преминава комуникационният процес. Тези описания се означават като **мрежови модели**. Като международен стандарт за абстрактно моделиране широко се използва **моделът OSI.**

OSI моделът описва базова архитектура, в която разделя мрежовите комуникации с комутация на пакети на **седем слоя**. Всеки слой обхваща различни мрежови действия, компоненти или протоколи. Това е модел, създаден в рамките на Международната организация за стандарти ISO (International Standards Organization) в Женева и е съставен на базата на американския IEEE стандарт.

Мрежов модел - Open System Interconnect (OSI)

Моделът OSI е създаден през 1978 г., а през 1984 г. е допълнен. В този модел пренасянето на съобщения и данни се извършва в поредица от независими един от друг слоеве

7. Приложен слой
6. Представителен слой
5. Сесиен слой
4. Транспортен слой
3. Мрежов слой
2. Канален слой
1. Физически слой

Мрежов модел - Open System Interconnect (OSI)

Приложният слой (7) е единственият мрежов слой, към който потребителят има пряк достъп. В този слой се намират програмните средства (приложения) и данни, с които се работи и които евентуално ще обменят информация с други станции. Към така подготвените съобщения се добавя заглавие (Header) на приложния слой, което представлява информационна структура, идентифицираща подаващия и приемащия компютър.

Представителният слой (6) е място, където съобщенията, съставени в приложния слой, се превеждат в удобна за пренасяне и разбиране от приемащата страна форма. Ако е необходимо, в този слой се извършва компресиране и кодиране на данните. Към така обработените данни отново се добавя заглавие с информация за използваните методи за компресия и кодиране, а също и за използвания формат на записване на данните.

Сесийният слой (5) осигурява свързването между мрежовите станции (управление, синхронизиране пренасянето на данни, защита от прекъсване на преноса и т.н.). В този слой също се взема решение за метода на свързване - полудуплекс (станциите се редуват в приемането и предаването) или пълен дуплекс (станциите едновременно приемат и предават). Информацията за съответния избор е записана в заглавието на сесийния слой.

Мрежов модел - Open System Interconnect (OSI)

Транспортният слой (4) се грижи за защитата на пренасяните данни. В този слой данните се разпределят на по-малки структури – пакети; тук се изготвят контролни суми, които позволяват откриването и коригирането на допуснати при пренасянето на данни грешки. В този слой се изготвят архивни копия за пренасяните данни, които по-късно могат да се използват за повторно изпращане, в случай че предаването на данни е неуспешно. Контролните суми и информацията за мястото на пакета в съобщението са оформени като заглавие и се добавят към пакета.

Мрежовият слой (3) има за задача да се грижи за избиране на оптимален път (канал), по който съобщението (или част от него) ще предава към определената станция-получател. Съответните междинни адреси също са прикрепени към данните под формата на заглавия.

Каналният слой (2) е свързващ и контролира пренасянето на пакетите. В този слой обикновено се съхранява и копие на пакетите до момента, в който приемащата станция потвърди безпроблемното пренасяне на съобщението. В случай на грешка при преноса на пакетите този слой осигурява повторното им изпращане и контролира евентуалното дублиране на приети пакети. Каналният слой изпраща кадрите с данни от мрежовия слой към физическия. При получаващата страна той пакетира последователните битове, идващи от физическия слой, в кадри (frames) с данни. Кадърът с данни е подредена логическа структура, в която се поставят данните и представляват пакети, обогатени с необходимите адреси и данни за синхронизация на съвместно работещите станции. Информацията за извършваните операции се съдържа в заглавието на каналния слой.

Физическият слой (1) осигурява кодирането на байтовете и битовете от кадрите във вид на електрически сигнали и пренасянето им по мрежовата среда. Този слой предава неструктуриран поток от битове по физическата среда. Физическият слой дефинира начините на свързване на кабелите с изходните терминали на мрежовите устройства. Този слой отговаря за предаването на битове от една станция към друга. Физическият слой дефинира кодирането на данните и синхронизирането на битовете, гарантирайки, че когато се изпрати единица, този бит ще бъде получен като единица, а не като нула, и обратно.

Методи за достъп до физическата среда

Методите за достъп до средата (Medium Access Control – MAC) описват начините, чрез които даден възел в мрежата запитва, получава и контролира достъпа си до физическата линия. Най-често се използват следните методи:

- Метод с откриване на колизиите (CSMA/CD);
- Метод с избягване на колизиите (CSMA/CA);
- Метод с предаване на маркер (Token Passing);
- Метод по заявка с приоритет (Demand Priority);
- Метод с разпределено време (STDMA).

Основната задача на контрола за достъп до средата е да се установи или евентуално предотврати "сблъсък" на пакети, предавани едновременно от два или повече възела в мрежата. Всеки един от посочените методи решава този проблем по различен начин.

Понятието **"Колизии"** (конфликт) се използва за означаване на събитие, при което две станции предават едновременно. Наслагването на сигналите води до загуба на информация.

Метод с откриване на колизиите (CSMA/CD);

Методът CSMA/CD е съкращение от **Carrier-Sense Multiple Access with Collision Detection** – множествен достъп с откриване на носещата и разпознаване на колизиите. "Откриване на носещата" означава, че всяка станция в мрежата непрекъснато "прослушва" средата за присъствието на сигнал, преди да се опита да предаде своите данни. "Множествен достъп" означава, че са налице много станции, опитващи се да предават, т.е. те се състезават за правото на предаване. "Разпознаване на колизиите" означава, че при откриване на такива, станциите спират да предават и всяка изчаква случаен интервал от време, преди да направи нов опит за предаване. Методът CSMA/CD работи най-добре в среда, където се предават относително малко на брой кадри, но със значителен обем/дължина. Този метод по-скоро контролира колизиите, а не ги предотвратява.

CSMA/CD методът се използва в мрежите от тип Ethernet.

Метод с избягване на колизиите (CSMA/CA);

Методът CSMA/CA е съкращение от Carrier-Sense Multiple Access with Collision Avoidance – множествен достъп с откриване на носещата и избягване на колизиите. Методът е подобен на CSMA/CD, но вместо само да се установи наличието на колизия, се прави опит тя да бъде избегната, като всяка станция предава сигнал за "намерение" за предаване преди самото предаване на данните по канала. Въпреки че методът позволява да се избегнат колизиите, това става за сметка на допълнително натоварване на трафика с разпространяването (broadcast) на допълнителни сигнали. Методът CSMA/CA е по-бавен от CSMS/CD – използва се в мрежите Apple.

Метод по заявка с приоритет (Demand Priority);

Методът за достъп по заявка с приоритет (Demand Priority Method) се появява с амбицията да замести CSMA/CD и да се превърне в алтернатива на Fast Ethernet. Характеризира се със следните особености:

– използва интелигентни концентратори (smart hub) за контрол на достъпа до мрежата, които фактически са арбитри за достъпа до средата; станциите изискват разрешение от концентратора за предаване на данните си, като те се предават чрез концентратора, а не се разпространяват до всички станции;

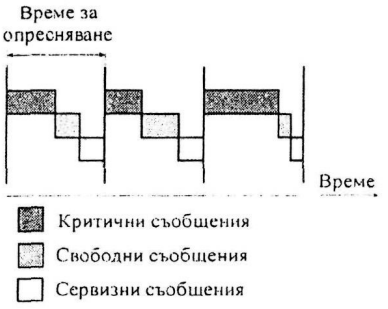
– станциите могат да предават и приемат едновременно с използване на кабел UTP Category 3 с 4 усукани двойки, всяка със скорост 25 MBit/s (стандартният 10BASET Ethernet използва 2 усукани двойки);

– възможно е приоритетно предаване на трафик, който е критичен по време за кадри от тип Ethernet или Token Ring.

За разлика от обикновения Ethernet, в който предаването е ориентирано към всички станции, при метода Demand Priority се използва насочено предаване. Интелигентният концентратор насочва трафика от подателя към определен получател вместо към всички станции.

Метод с разпределено време (STDMA)

При метода с разделено време CTDMA достъпът до средата се контролира от алгоритъм за времеделене – Concurrent Time Domain Multiply Access (CTDMA). Той регулира възможността на устройствата да предават данни във всеки мрежов интервал, наречен време за опресняване на мрежата (Network Update Time - NUT). В мрежите ControlNet например най-краткият период на опресняване е 2ms. Информацията, която е критично необходима, се изпраща през резервираната част от времевия период .



Информацията, която не е критично необходима за работата на системата (като конфигурационни данни), се предава през свободната част от интервала. Съществува и трети времеви период, в който се предават съобщения за синхронизиране работата на мрежата.

Комуникационни механизми

Master-устройства

С термина master-устройства в индустриалните мрежи се означават тези възли, които имат управляващи или информационно-контролни функции. Примери за такъв тип устройства са програмируемите контролери (PLC), операторски станции, терминали, устройства за човеко-машинен интерфейс, регистриращи и архивиращи системи и др.

Slave-устройства

С термина slave-устройства в индустриалните мрежи се означават тези възли, които представляват източници на първична информация (сензори, измервателни системи) или други крайни устройства като регулиращи органи, изпълнителни механизми, модули за отдалечен вход/изход и др.

Multi-Master

Механизмът за комуникация от тип Multi-Master се използва, когато е необхо¬димо няколко master-устройства да получат достъп до данните, предоставяни от едно и също slave-устройство. При това обаче само едно master-устройство има правото да управлява физическите изходи и конфигурацията на slave-устройството. Механизмът Multi-Master се използва в индустриалните мрежи DeviceNet, Profibus-DP и др.

Комуникационни механизми

Peer-To-Peer

Механизмът от тип peer-to-peer се използва за комуникация между устройст¬вата в мрежата на равноправна основа. Той се прилага за обмен на "опознавателна" информация между slave и master-устройства. Достъпен в индустриалните мрежи Profibus-FMS.

Производител / Консуматор (Producer / Consumer)

В някои индустриални мрежи се използва механизъм, при който данни / информация се предават периодично с метод broadcast и всеки възел може да получи тази част, която му е необходима, и да я използва съобразно своите нужди без закъснение. Този механизъм е известен като производител / консуматор. Индустриалните мрежи ControlNet, DeviceNet, FieldBus са базирани на производител / консуматор технологията, при която всички възли имат едновременен достъп до едни и същи данни от един и същи източник.

Моделът производител / консуматор притежава и редица други преимущества, най-важното от които е пълноценно използване на мрежовата среда и увеличаване на пропускателната възможност.

17

Комуникационни механизми, специфични за индустриалните мрежи

Запитване (Polling)

При този режим master-устройство изпраща запитване индивидуално до всяко подчинено устройство, за да изиска доставянето на входно/изходни данни. От своя страна подчиненото устройство отговаря на всяко такова запитване, дори и ако то не е валидно. Този механизъм е достатъчно прецизен, но недостатъчно ефикасен от гледна точка на използване на мрежовата среда, тъй ка¬то останалите устройства нямат достъп до обменяните данни, дори и да са им необ¬ходими.

Групово стробиране (Strobing, broadcast)

При този механизъм master-устройството изпраща общо запитване до всички подчинени устройства за да изиска доставянето на нови входно/изходни данни. От подчинените устройства отговарят само тези, които "се разпознаят" в broadcast съобщението. Например, в механизма BitStrobe на спецификацията DeviseNet това съобщение представлява поредица от битове, като поредният номер на всеки бит отговаря на всеки възел в мрежата и с единица са обозначени тези устройства, от които се очаква отговор.

18

Комуникационни механизми, специфични за индустриалните мрежи

Циклични съобщения (Cyclic)

Използват се най-често от подчинените уст-ройства. Те се конфигурират така, че да предават към master-устройствата си данни за хода на процеса през определен интервал. В системите за управление това е удобен режим за съблюдаване на постоянен период на дискретизация за пряко цифрово управление.

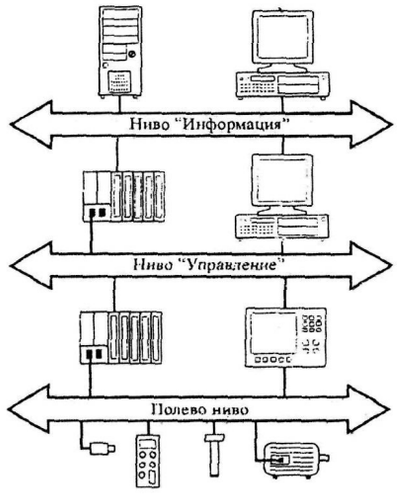
Промяна на състоянието (Change Of State - COS)

Подчинените устройства изпращат входно/изходни данни към master-устройствата си само когато се смени състояни¬ето им или това на обекта за управление. В останалото време управляващият алго¬ритъм счита "старите" данни за валидни/текущи и ги използва за целите на управле¬ние на процеса. Този механизъм намалява до минимум трафика по мрежата и пре¬възхожда останалите по скорост няколко пъти.

19

Йерархична структура на системите за управление с мрежова комуникация

От функционална гледна точка съвременните системи за управление с мрежова комуникация могат да бъдат разпределени на три нива



20

Йерархична структура на системите за управление с мрежова комуникация

Ниво 1: Полеви мрежи - мрежи за комуникация на Програмируеми Логи-чески Контролери (Programmable Logic Controllers – PLC) с интелигентни крайни устройства - сензори, измервателни устройства, изпълнителни меха-низми, панели от бутони за ръчно управление на процесите. От този тип са мрежови спе¬цификации като: DeviceNet, CAN, AS-Interface, Profibus-DP, Interbus-S, Modbus, Berges и др.

Полевите мрежи са в основата на йерархията и се характеризират със съоб¬щения с минимална дължина и високонадеждна връзка между отделните възли. Тези мрежи се използват при управлението на непрекъснати производствени процеси, в дис¬кретното производство, енергетиката и т.н. за управление на контролери, сензори и изпълнителни механизми от най-ниското ниво в йерархията на системите за промишлена автоматизация. Това са устройства, между които пре¬даването на информация за измервания и управление е много интен¬зивно. Основно полевите мрежи се отличават с простота, голяма скорост на предаване на данните, голяма гъвкавост и ниска цена на използваната апаратура.

Йерархична структура на системите за управление с мрежова комуникация

Ниво 1: Полеви мрежи

Предимствата на полевите мрежи, сравнени с 4-20mA сигналните и други кон¬венционални структури, са:

- по-висока сигурност и точност на данните;
- многовариантен достъп;
- дистанционно конфигуриране и диагностика;
- намаляване на кабелните системи.

По-голямата точност се дължи на цифровата връзка, тъй като микропроцесо¬рите в едно предаващо устройство или един контролер могат да комуникират директно, вместо да преминават през АЦП и ЦАП превръщания, които може да са голям брой. Проблемите със сигурността на данните се преодоляват чрез включване на контролна информация към предаваните данни. Многовариантният достъп означава, че даден ин-телигентен сензор не е ограничен само до отчитането на един параметър, но също информира и за други характеристики на процеса.

Йерархична структура на системите за управление с мрежова комуникация

Ниво 2: Ниво "Управление"

Изгражда се от програмируеми логически контролери (PLC). На това ниво се извършва управлението на процесите в реално време и си взаимодействат PLC от различен тип, операторски станции, персонални компютри, устройства за програмиране и диагностика и др. Мрежа от това ниво е предназначена за обмен на технологични програми, координираща и управляваща информация за контролерите, свързани към отделните агрегати или групи от агрегати. При този тип мрежи изискванията към обема на предаваната информация са значителни, както и са повишени изискванията по отношение на шумоустойчивост и детерминираност (предсказуемост) за работа в реално време. Някои стандартни протоколи са неподходящи за подобен род приложения, което налага използването на специализирани фирмени протоколи.

Второ ниво от йерархията в индустриалното управление е пресечна точка за изискванията към другите две нива. Тук освен бързината на обмен на данни, толкова нужна за полевото ниво, се появява и необходимостта от висока достоверност и по-голям обем на предаваната информация.

Йерархична структура на системите за управление с мрежова комуникация

Ниво 3: Ниво "Информация"

За управление и обмен на информация на ни¬во предприятие. Мрежите от това ниво се характеризират с висока пропускателна способност и са предназначени за обмен на големи масиви от информация. Системите от ниво "информация" имат за цел да решават глобалните задачи в управлението. Основните проблеми на това ниво са свързани с обема на предаваната информация и скоростта, с която се извършва това. Също така значителни са и изискванията за достоверност и защита от неоторизиран достъп на предаваната информация. За разлика от долните две нива от йерархията, където разнообразието е по-голямо, тук най-широко приложение в съвременните системи е намерила спецификацията, известна като Ethernet. Версията Fast Ethernet достига 100 Mbit/s. Ethernet окончателно се налага при локалните мрежи след съревнование с други технологии като например Token Ring. Преминаването от Ethernet към Fast Ethernet (а в бъдеще и към Gigabit Ethernet) е с минимални разходи на време и средства.

Локална мрежа за индустриално приложение

Изискванията към локалната мрежа за индустриално приложение, както и за свързване на компютърните системи от ниските етажи на големите системи за събиране и преобразуване на информация, се различават съществено.

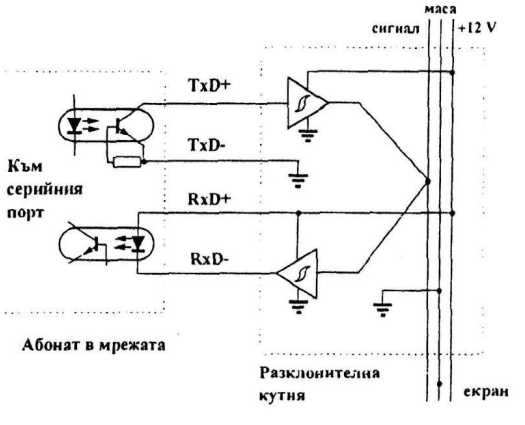
- Тук не е необходима висока скорост на предаване на данните, тъй като интензивността на информационните потоци с данни не е голяма, защото програми и файлове не се предават по такива мрежи. Освен това значително по-маломощните процесори на тези компютърни системи трудно ще подържат протоколите на скоростните мрежи.
- Галваничното разделяне на всяка станция е абсолютно задължително по вече коментирани причини.
- Нивото на електрическите и електромагнитните смущения в индустриални условия е много по-високо и се налага да се вземат изключителни мерки за устойчивост - както апаратни, така и програмни.

Желателно е да се минимизира цената на апаратното осигуряване на локалната мрежа, понеже броят на компютърните системите от ниските етажи е значително по-голям.

25

Локална мрежа за индустриално приложение

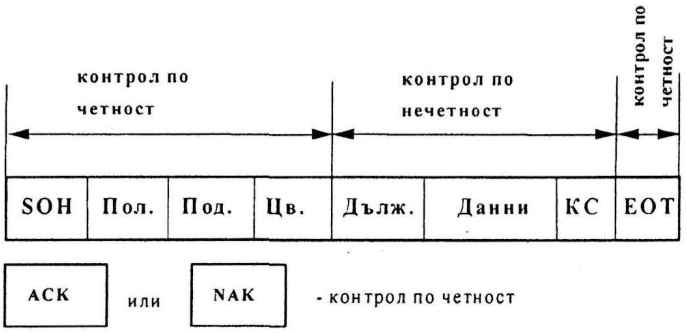
Мерките за шумоустойчивост са взети с екраниране на линията, прилагането в апаратния приемник на тригер на Шмид с широк хистерезис и с натоварване на линията в двата края с резистори.



26

Протокол

Поради силната зашуменост на средата се налага данните да се блокуват на сравнително къ-си блокове. Пакетът данни на индустриалната локална мрежа има следната структура:



27

Протокол

1. **Водещ символ**, 1 байт. Указва началото на предаването на блок данни. Може да бъде два типа - начало на предаването на съобщение (Start Of Tranmission SOH) и начало на предаването на блок данни (Start Of Text STX). Първият символ стои в началото на съобщението. Вторият е в началото на втория и следващите блокове на многоблоково съобщение.
2. **Получател**, 1 байт. Указва адреса на станцията получател на съобщението. Размерът му ограничава броя на станциите в мрежата до 256.
3. **Подател**, 1 байт. Число указващо адреса на подателя на съобщението.
4. **Цвят на блока**, 1 байт. Има стойности нула или FF16, които се редуват алтернативно при предаване на многоблоково съобщение. В литературата това се нарича оцветяване на блока в "синьо" или "оранжево". Смисълът му ще бъде коментиран по-надолу.
5. **Дължина на блока**, 1 байт. Указва броя на байтовете в информационната част. Обикновено е до 128.

28

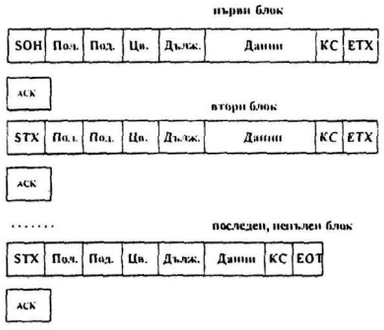
Протокол

6. **Информационна част**, от 1 до 255 байта. Това са данните.
7. **Контролна сума**, 1 байт. Представлява младшия байт на сумата на всички предадени досега байтове и служи за проверка на верността на приемането на блока.
8. **Завършващ символ**, 1 байт. Указва края на предаването на блок данни. Може да бъде два типа ? край на предаването на съобщение (End Of Transmision EOT) и край на предаването на блок данни (End Of Text ETX). Първият символ стои в края на последния блок от съобщението. Вторият е в края на първия и следващите блокове на многоблоковото съобщение без последния блок.

За да се отличава служебната информация от информационната част тя се предава с контрол по четност, а байтовете на информационната част - с контрол по нечетност.

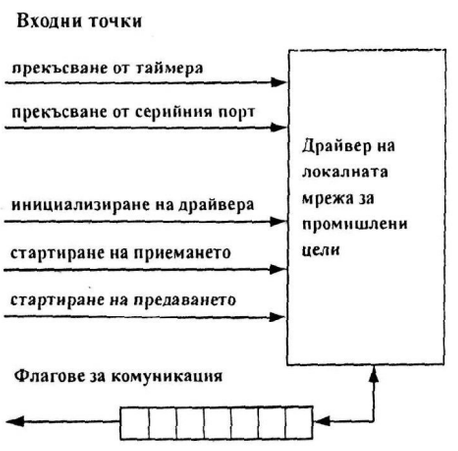
Протокол

Предаването на съобщение се извършва многоблоково, блок след блок, ако то е голямо и не се събира в един блок, както е показано на фигурата. Разделянето на кратки блокове се налага за да не се повтаря цялото дълго съобщение в случай на възникване на грешка, а само сгрешения блок. Освен това при дълго съобщение вероятността да възникне грешка е по-голяма. Всички действия по мрежата имат своите защитни времена. Ако предаващата станция не получи квитанция до определено време повтаря предаването на текущия блок. Това налага и алтернативното оцветяване на блоковете.



Структура на драйвера

Драйверът на локалната мрежа за индустриални цели е програма, реализираща всички функ-ции по изпращане и приемане на съобщения по мрежата, защитните времена, повторенията при неу-спешно предаване и др. Като алгоритъм програмата не е сложна. Сложното тук е, че драйверът тря-бва да работи само в прекъсване и да не отнема много време от процесора. Схематично структурата на драйвера е показана на фигурата.



31

Структура на драйвера

Най-общо казано програмата се състои от инициализираща част, приемник и предавател. Освен соб-ствената си работна зона в оперативната памет драйверът подържа и дума на състоянието, посредст-вом която чрез установяване и изчистване на флагове комуникира с приложното програмно осигуря-ване. Драйверът има следните входни точки:

1. **Прекъсване от системния таймер.** Управлението се предава тук при всяко прекъсване от системния таймер. Това е "чувството за време" на драйвера. Тази входна точка само модифи-цира броячите на защитните времена и не задържа за дълго процесора.
2. **Прекъсване от серийния канал.** Това е прекъсването от серийния порт, който подава заявка при постъпване на символ от мрежата и при завършване на изпращането на символ. Драйверът обработва всеки постъпил символ в зависимост от режима в който работи. Следи дали линията е заета, дали не пристига съобщение за този абонат, а в режим на предаване - дали полученият символ е същият който е предал.

32

Структура на драйвера

3. **Инициализация на мрежата.** Тази входна точка се извиква от програмата за начално установяване на компютърната система. Тук се извършва инициализация на серийния порт, зада-ване на векторите за прекъсване и присвояване на адреса на станцията. Приемникът и предавателят се установяват в подтиснато състояние. След това управлението се връща на ви-кащата програма.

4. **Стартиране на приемането.** Тази входна точка се активира от приложния слой на програмното осигуряване. Той указва адреса на входния буфер и максималната му дължина. Драйверът прави необходимите промени в думата на състоянието (установява необходимите флагове) с което привежда приемника в активно състояние и връща управлението на приложната програма без да задържа продължително процесора. От тук нататък приемникът започва да следи линията и да приема съобщения за тази станция. След приемане на съобщение при-емникът маркира това с установяване на съответните флагове в думата на състоянието и преминава в подтиснато състояние. Приложната програма е длъжна да проверява периодично флаговете в думата на състоянието и след приемане на съобщение да го изтегли за обработка и да активира отново приемника от тази входна точка.

33

Структура на драйвера

5. Стартиране на предаването. Тази входна точка също се активира от приложния слой на програмното осигуряване само при условие, че предавателят е в подтиснато състояние, т.е. е завършил предаването на предишното състояние. Приложната програма указва адреса на изходния буфер, дължината на съобщението и позивната на получателя. Драйверът прави необходимите промени в думата на състоянието (установява необходимите флагове) с което привежда предавателя в състояние на готовност и връща управлението на приложната програма без да задържа продължително процесора. От тук нататък предавателят проверява дали линията е заета и след освобождаването ѝ започва да предава съобщението. След завършване на предаването предавателят маркира това с установяване на съответните флагове в думата на състоянието и преминава в подтиснато състояние.

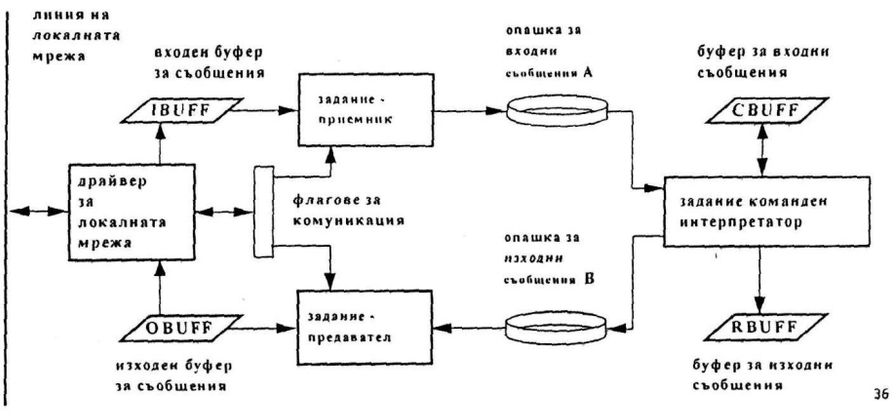
34

Организиране на приемането на потока от съобщения

В работата на една система за събиране и обработка на информация участват много (десетина и повече) задания под управлението на операционната система за реално време. Тук ще се спрем само на едно типово решение за организирането на потока от съобщения по локална мрежа - приемането на съобщението, обработката му и предаването на евентуалния отговор. По принцип всяка отделна част (компютърна система) от една разпределена слабосвързана система за събиране и преобразуване на информация трябва да може да работи самостоятелно. Обикновено съобщенията по локалната мрежа са кратки команди за смяна на режима или за изискване на натрупана информация. Съобщението-отговор, което се изпраща също не е с голяма дължина. Потокът от съобщения трябва да се приема и обработва заедно с подържане на нормалната работа на цялата система. Възможно е в даден момент да се получат няколко съобщения едно след друго с темп по-голям от способностите на системата да ги обработи. Това налага и буферирането на съобщенията.

Организиране на приемането на потока от съобщения

За приемане, буфериране, обработка и предаване на съобщения по локалната мрежа се гри-жат три от заданията под управлението на операционната система за реално време: задание приемник, задание интерпретатор на съобщенията и задание предавател. За комуникация между тях се използват опашките A и B, с обем достатъчен да поеме съобщенията докато се обработва едно от тях.



Организиране на приемането на потока от съобщения

Неразделна част от комуникацията чрез локалната мрежа е **драйверът за локалната мрежа**, който комуникира със заданията посредством думата на състоянието с шестнадесет флага. Програмата за начална инициализация на компютърната система е длъжна да стартира точката за инициализация на драйвера за мрежата още преди стартиране на операционната система за реално време.

Заданието приемник след стартирането си активира приемника на драйвера и след това само проверява дошло ли е съобщение. Ако не е пристигнало съобщение се обръща към операционната система за реално време и се отказва от повече процесорно време с което предоставя своето време на другите задания. В случай на пристигане на съобщение във входния буфер IBUFF (индицира се с установяване на флаг в думата на състоянието) това задание го прехвърля заедно с позивната на подателя в опашката за входни съобщения А и отново активира приемника.

37

Организиране на приемането на потока от съобщения

Заданието интерпретатор на съобщенията проверява празна ли е опашката за входни съобщения А. Ако да - обръща се към операционната система за реално време и се отказва от повече процесорно време с което предоставя своето време на другите задания. В случай на непразна опашка изтегля съобщението в буфера за входящи съобщения CBUFF и го обработва.

Съдържанието на съобщението зависи от конкретната програма и може да съдържа команда за изпълнение или команда за даване на информация. В зависимост от съдържанието му може да се формира съобщение - отговор в буфера RBUFF. След това отговорът, заедно с позивната на подателя, се изпраща в опашката за изходни съобщения В.

38

Организиране на приемането на потока от съобщения

Заданието предавател проверява празна ли е опашката за изходни съобщения B. Ако да - обръща се към операционната система за реално време и се отказва от повече процесорно време с което предоставя своето време на другите задания. В случай на непразна опашка се прави проверка дали предавателят на драйвера на локалната мрежа не е активен. Ако предавателят на драйвера за локалната мрежа е активен се изчаква да премине в подтиснато състояние, т.е. да довърши предаването на предишното съобщение. След това се изтегля съобщението в изходния буфер OBUFF и се активира предавателя на драйвера на локалната мрежа, който веднага щом се освободи линията започва да предава съобщението. През това време заданието предавател чака постъпването на следващото съобщение в опашката за изходящи съобщения B.

Тази примерна организация на приемането, обработката и изпращането на съобщения по ло-калната мрежа под управлението на операционна система за реално време е една илюстрация на це-лия материал по тази тема - засягат се всички структури от данни за комуникация между заданията и се организира комуникация между компютърни системи.

39

Ядро на операционна система за реално време

Времево базирана система /ВБС/

➢Тя се основава на превключване на отделните задачи през зададен период от време, посредством таймерната система на микроконтролера.

➢ Броя на включените в системата задачи за изпълнение се определя от наличната оперативна памет RAM. Всяка задача трябва да завършва своето изпълнение за време по-малко от зададения период на превключване.

➢Така тук също се избягва нуждата от запазване състоянието на регистрите и Таблицата на задачите съдържа само адресите за преходите към всяка една от тях.

За по-добро разясняване действието на системата ще се разгледа конкретна реализация.

I. Основни характеристики на ПБС

1. Тя използва таймерната система на микроконтролера за разделяне времето на интервали, които се разпределят между отделните задачи.

2. Синхронизацията при изпълнението на задачите се управлява от прекъсванията на таймера. Тези прекъсвания се генерират по различен начин в зависимост от таймера, който се използва.

3. В реализацията се подържат два вида таймери:

- програмируем таймер. При него се задава броя на импулсите, след което се генерира прекъсване;
- таймер по препълване. При него прекъсване се генерира след препълване съдържанието на брояча;

Кой таймер ще се използва зависи от микроконтролера. Микроконтролерите на фирмата Motorola от сериите HC05 и HC11 имат таймери и от двата вида. Кой от тях ще се използва зависи от настойката на програмното ядро.

I. Основни характеристики на ПБС

4. Таймерните системи на микроконтролерите на посочента фирма притежават 16 битов брояч, който неприкъснато сумира импулсите /free-running counter/ за синхронизация E /Enable/. Възможно е и предварително деление на тази импулсна поредица на фиксирани стойности.
5. Когато се генерира прекъсване от един от двата таймера, се стартира програма за обслужване на прекъсването, чийто начален адрес се намира на различни места в паметта, в зависимост от използвания таймер.
6. Програмата обслужваща прекъсването е кратка за да осигури повече време за задачите.
7. Броячът на интервала на времето /Time Slice Counter TSC/ определя периода за превключване на задачите, като брои броя на прекъсванията от таймерната система.
8. Броячът на интервала на времето TSC е достъпен и може да се използва като таймер от отделните задачи. Например за формиране на закъснения и др.

I. Основни характеристики на ПБС

10. Броят на задачите няма ограничения. Той се определя само от броя на битовете в брояча на задачите TC. В реализацията броя на задачите е 8, тъй като TC е с големина един байт.
11. Задачите, които са по-дълги от един период на времето, трябва да бъдат разбити на подзадачи. Например при задача за програмиране в EEPROM, където е необходимо задържане между отделните процедури, е възможно тя да се раздели на следните подзадачи:

- изтриване на байт;
- записване на байт;
- проверка на записа.

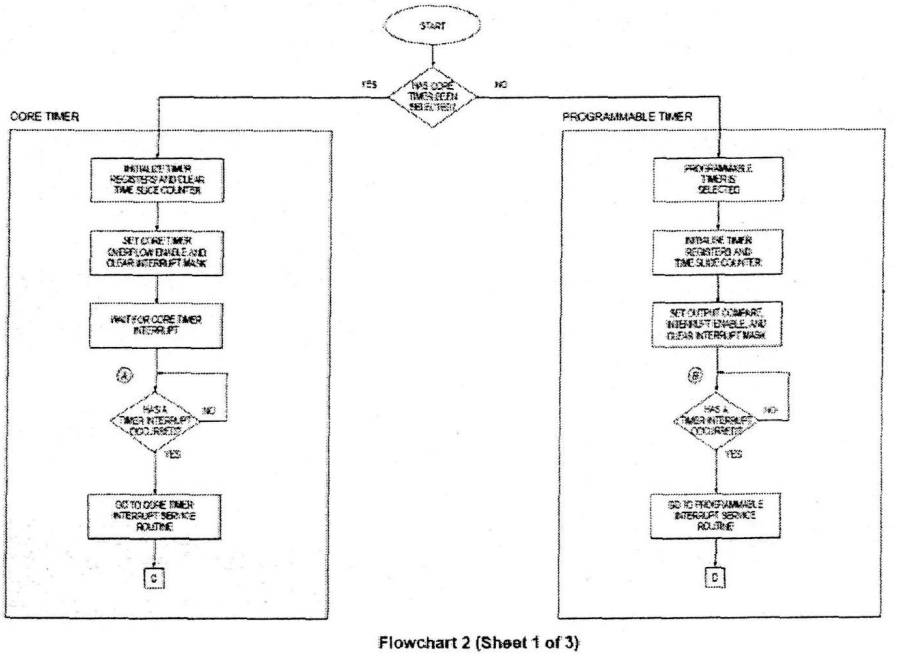
12.Флагове, вътрешни за задачата, се използват да управляват, коя подзадача трябва да се стартира, при предаване изпълнението на задачата.
13. Ядрото подържа локални и глобални променливи.

II. Генериране на прекъсване от таймера

При използване на програмируем таймер прекъсване се генерира, когато стойността на основния брояч /**free running timer**/ се изравни с тази записана в регистъра на таймера. Тази предварително записана в регистъра стойност определя периода и може лесно да се модифицира. Така системата може лесно да се адаптира за приложения с различен интервал за превключване на задачите.

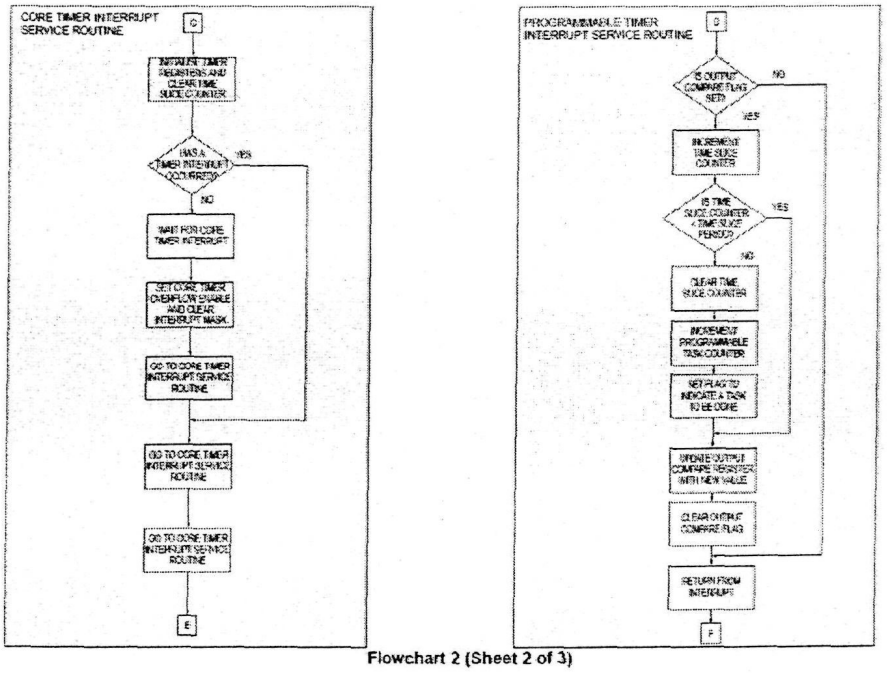
При използване на таймер по препълване, прекъсване се генерира всеки път, когато регистъра преминава от $FF към $00 (при 8-битов брояч на таймера). За разлика от програмируемия таймер, тук стойността не може да бъде променяна.

III. Алгоритъм на работа



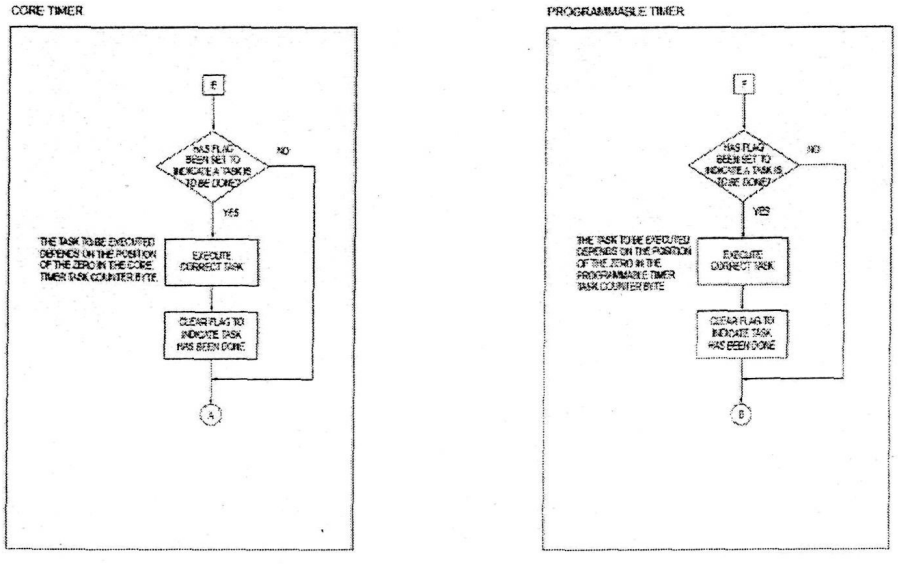
Flowchart 2 (Sheet 1 of 3)

III. Алгоритъм на работа



Flowchart 2 (Sheet 2 of 3)

III. Алгоритъм на работа



Flowchart 2 (Sheet 3 of 3)

III. Алгоритъм на работа

Интервалът от време се установява при стартиране на програмата и позволява лесна настройка. Броячът на интервала от време TSC се увеличава всеки път прекъсване, докато достигне стойността на периода на превключване на задачите /Time Slice Period TSP/. Когато това стане, броячът на задачите TC се увеличава и се изпълнява съответната задача. Броячът на интервала от време TSC се нулира и започва да отброява новия период.

Всяка задача трябва да се изпълнява за време по-малко от периода на превключване на задачите TSP. В противен случай задачата трябва да се разбие на подзадачи.

III. Алгоритъм на работа

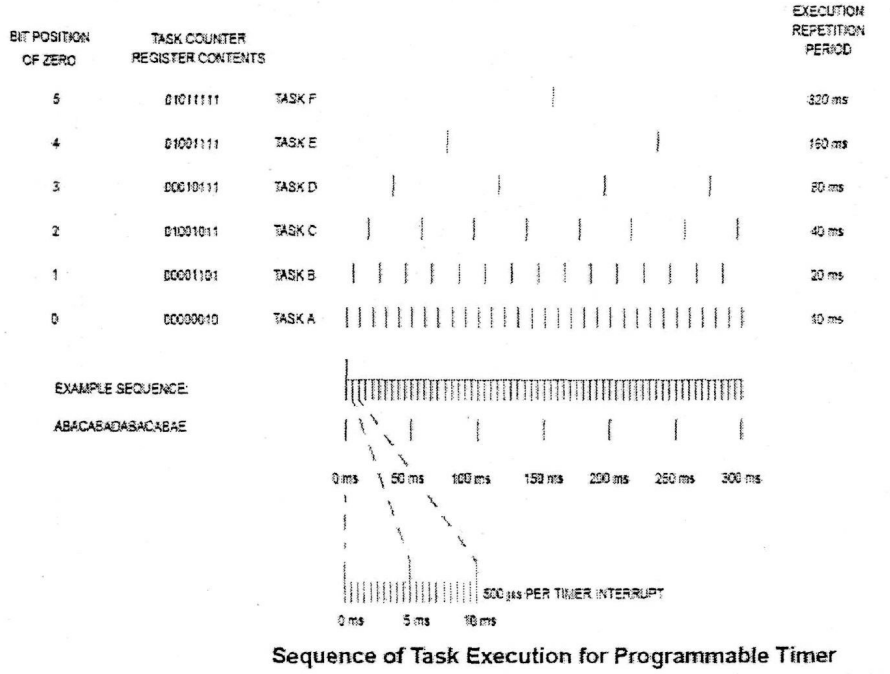
Ядрото предоставя на всяка задача флаг, осигуряващ й различен времеви приоритет (честота на изпълнение). Така задачите ще се изпълняват на периоди кратни на степента на двойката – 2^0, 2^1, 2^2 и т.н. Най-често изпълняваната задача (с най-висок приоритет) ще се изпълнява през интервал от 2^0 периода. Следващата – през интервал 2^1 периода и т.н. Задачите са означени с буквите A, B, C, D и т.н., като задача A е с най-висок приоритет. Задача F ще се изпълнява през интервал 2^5 периода.

Всеки бит от брояча на задачите TC съответства на дадена задача. Всеки път с увеличаване на брояча на задачите TC програмната система проверява байта на TC за наличие на нули, започвайки от най-младшия бит. Когато бъде открит нулев бит проверката се прекратява и се стартира задачата съответстваща на този бит. Забележете, че нито една задача не се изпълнява, когато всички битове на TC са 1 - $FF. Тогава е възможно изпълнението на фонова задача или на празен цикъл.

III. Алгоритъм на работа

Може да има толкова задачи, колкото са битовете на **TC**. Самият TC може да се състои от много байтове. Възможно е да се използват само някои от времевите нива /**time slots**/ , т.е. да липсват някои задачи. Неизползваните периоди могат да осигурят повече време за фоновата задача.

IV. Пример



Sequence of Task Execution for Programmable Timer

IV. Пример

В примера от фигурата се формира период **TSP** за изпълнение на задачите от **5ms**. Така най-често изпълняваната **Задача А**, съответстваща на бит нула на ТС ще се изпълнява на всеки **10ms**.

Периодът от 5ms се получава, като се отброят 10 прекъсвания, като всяко прекъсване е през интервал от 500 μs. Прекъсване от 500 μs се получава като системният такт Е (в конкретния случай той е 2 μs) се умножи по 250. Регистърът на таймера след всяко прекъсване се увеличава с 250 десетично. Периодът на изпълнение на всяка задача е

5×2^n , ms,

където n е позиционния номер на буквата в азбуката, обозначаваща дадената задача. За Задача С – $5x2^3 = 40$ ms.

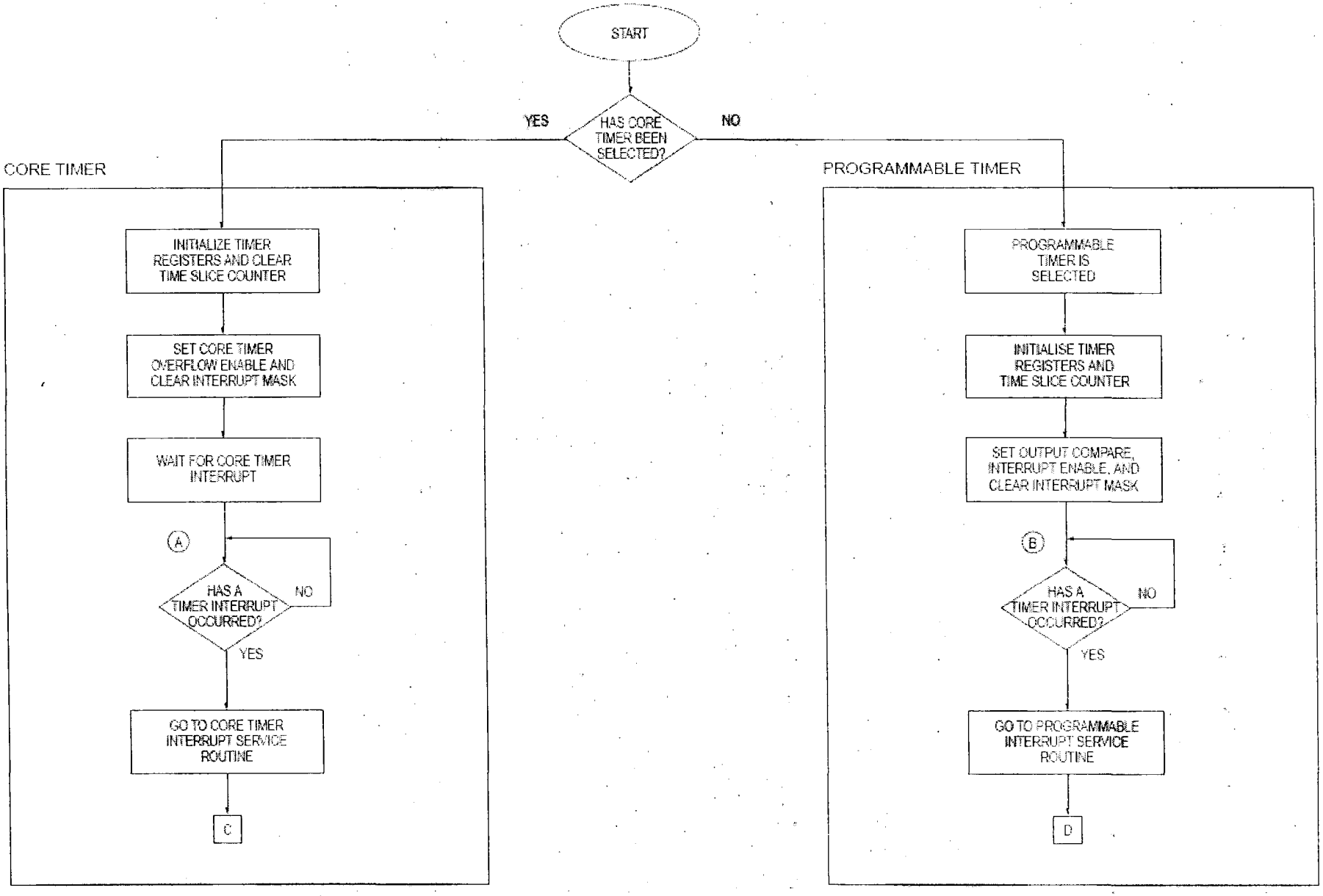
IV. Програмна реализация

Асемблерската програма е за микроконтролера MC68HC11. Всяка задача е обособена като отделна подпрограма, която задължително завършва с инструкцията **RTS**. На нивата, на които няма задачи, е необходимо също да има инструкция **RTS**.

Показаната реализация има само осем времеви (приоритетни) нива. Те могат да бъдат увеличени, чрез увеличаването броя на байтовете на брояча на задачите **ТС**. В реализацията са включени прости задачи, които само илюстрират начина на включване към ядрото.

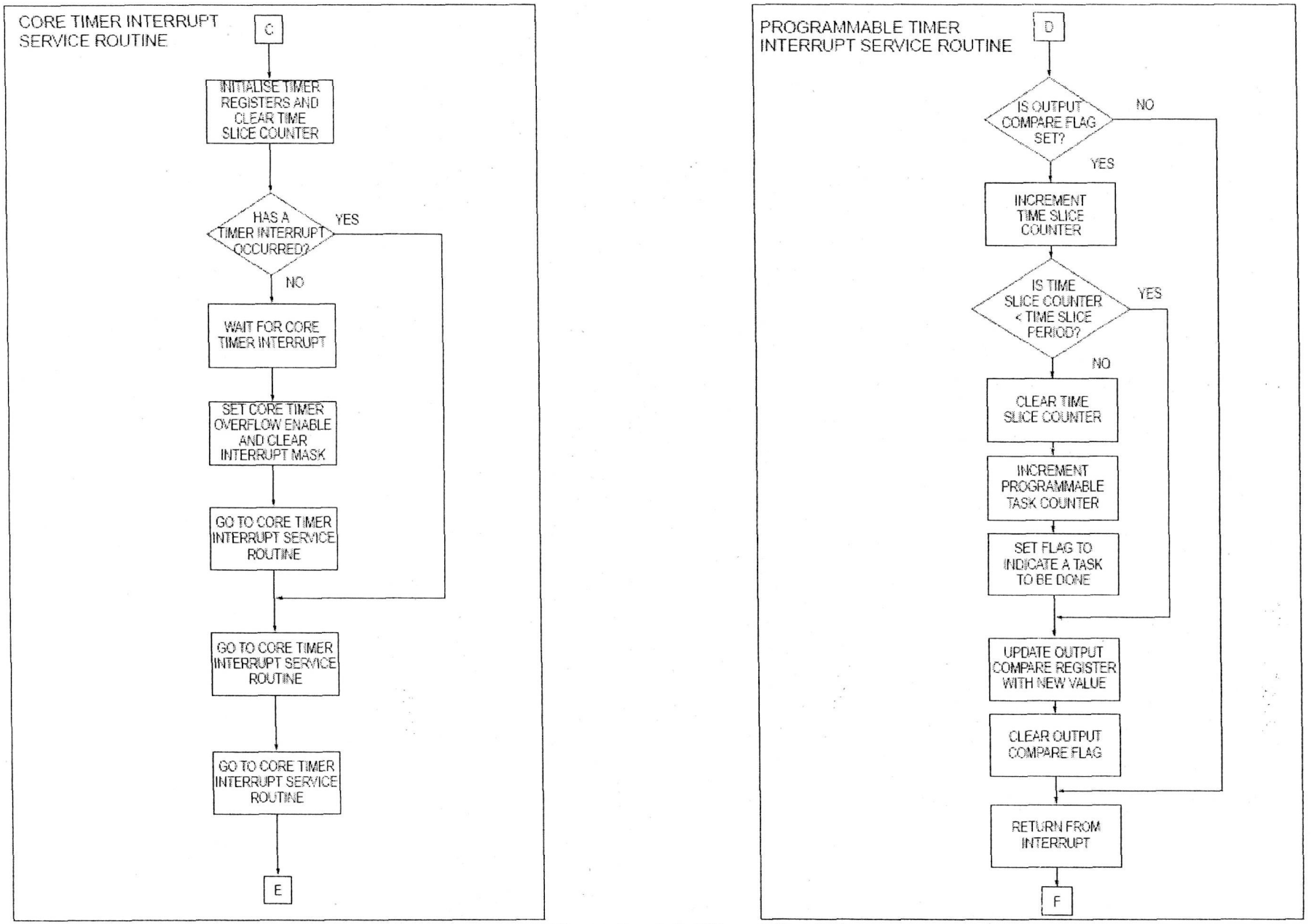
Задължително условие преди включването на дадена задача към ядрото е да се анализира времето за нейното изпълнение. И в на-лошия случай, то трябва да е по-малко от периода на превключване на задачите.

III. Алгоритъм на работа



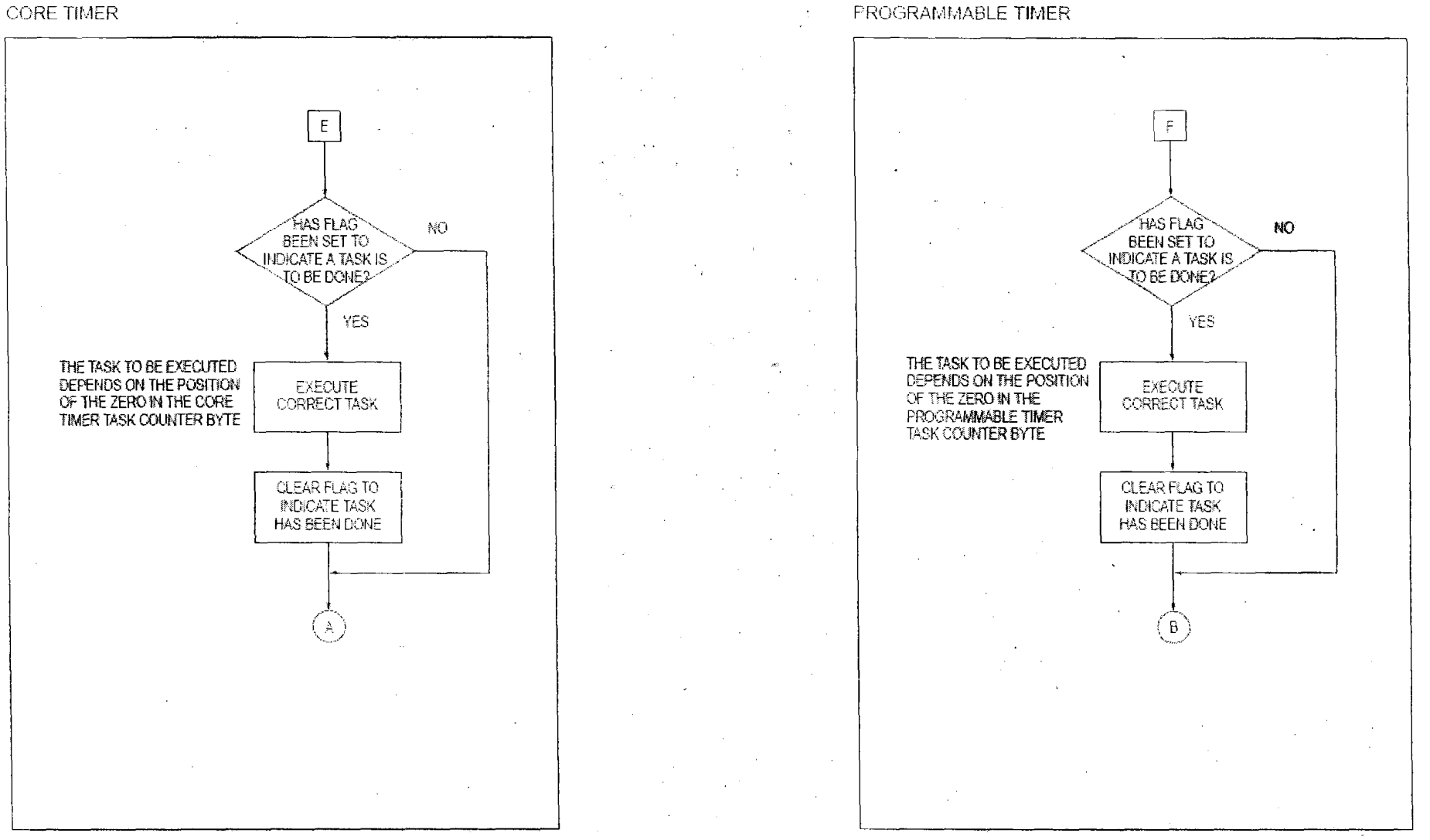
Flowchart 2 (Sheet 1 of 3)

III. Алгоритъм на работа



Flowchart 2 (Sheet 2 of 3)

III. Алгоритъм на работа



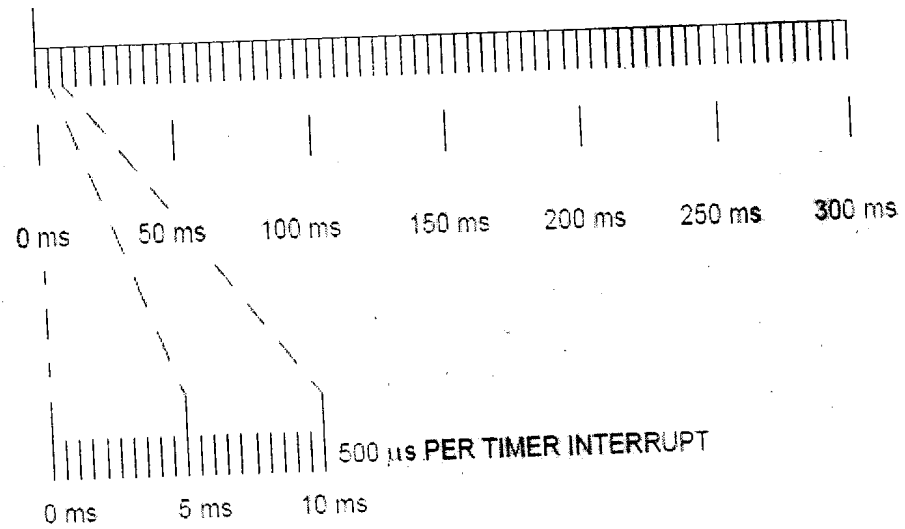
Flowchart 2 (Sheet 3 of 3)

IV. Пример

BIT POSITION OF ZERO	TASK COUNTER REGISTER CONTENTS		EXECUTION REPETITION PERIOD
5	01011111	TASK F	320 ms
4	01001111	TASK E	160 ms
3	00010111	TASK D	80 ms
2	01001011	TASK C	40 ms
1	00001101	TASK B	20 ms
0	00000010	TASK A	10 ms

EXAMPLE SEQUENCE:

ABACABADABACABAE



Sequence of Task Execution for Programmable Timer

Последователни интерфейси за локални мрежи

1

Интерфейс RS485

RS485 е стандарт на EIA (Electronics Industry Alliance) за мултиточкова комуникация. RS485 позволява множество устройства (до 32) да осъществяват полудуплекс комуникация чрез единична двойка проводници плюс заземяващ проводник на разтояние до 1-2 километра. Както дължината на мрежата, така и броят на точките (устройствата, участващи в мрежата) могат лесно да бъдат увеличени чрез различни повторители на сигнали, предлагани на пазара.

Когато една мрежова среда се нуждае от пренос на малки блокове от информация на дълги разстояния, RS485 е най-често избрания интерфейс. Комуникационните точки може да представляват Персонални Компютри, микроконтролери, или всякакви други устройства с възможности за асинхронна серийна комуникация.

В сравнение с Етернет и други мрежови интерфейси, RS485 изискванията за хардуер, както и изискванията на протокола са по-прости и евтини. RS485 стандартът е достатъчно гъвкав за да може да осигури широк избор на драйвери, приемници и други компоненти, в зависимост от дължината на кабела, размера на данните, броя на комуникационните точки и ограниченията за разход на енергия.

2

Интерфейс RS485

RS485 стандартът изисква данните да се предават диференциално по два усукани проводника, наречени усукана двойка. Свойствата на диференциалните сигнали осигуряват защита срещу шумове (смущения) и възможности за пренос на дълги разтояния.

RS485 мрежата може да бъде конфигурирана по два начина - като двупроводна или четирипроводна. В двупроводната мрежа предавателят и приемникът на всяко от устройствата са свързани към усукана двойка. Четирипроводната мрежа има един мастър порт с предавател, свързан с всички подчинени устройства чрез една усукана двойка, а предавателите на подчинените устройства са свързани с мастър приемника чрез втора усукана двойка.

При всяка от конфигурациите устройствата са адресируеми, позволявайки с всяко от тях да бъде комуникирано самостоятелно. В един и същ момент само едно от устройствата може да възбужда комуникация по линията, така че драйверите на всички останали участници в мрежата трябва да бъдат във високоимпедансно състояние, когато не комуникират. Някои RS485 устройства поддържат тази функция автоматично. В противен случай софтуерът на RS485 устройството трябва да управлява драйвера си чрез използването на контролна линия.

3

Интерфейс RS485

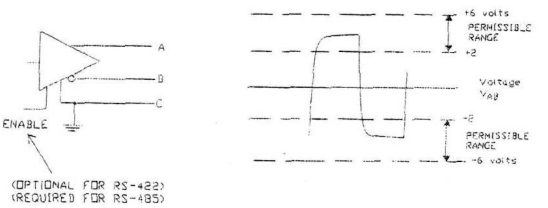
И двете RS485 мрежови конфигурации, двупроводна и четирипроводна, имат своите предимства и недостатъци. Двупроводната има предимството на по-ниски разходи за опроводяване, а освен това позволява на всички свързани устройства да комуникират помежду си. От друга страна недостатък на двупроводната конфигурация е ограничението до полудуплекс, както и изискването за съобразяване със закъснението при преминаване от едно състояние в друго.

Четирипроводната мрежа позволява пълендуплекс операции, но е ограничена до комуникации от тип мастър устройство - подчинено устройство (т.е. мастър устройствата изискват информация от единични подчинени устройства). Подчинените устройства не могат да комуникират помежду си.

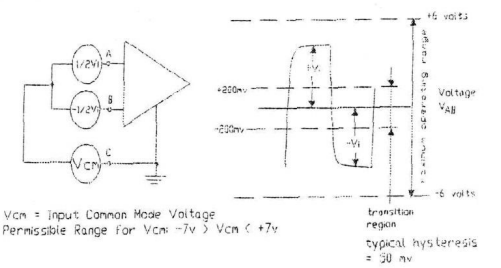
Терминирането на RS485 линията е крайно необходимо, особено при висока скорост на трансфер на данни и дълго опроводяване. Основните причини за необходимостта от коректно терминиране са отраженията на сигнала в краищата на линията, както и изискванията за минимален товар на предавателя. Тъй като всяко устройство комуникира двупосочно (версия на единична усукана двойка), не сме в състояние да определим къде е предавателят и къде приемникът - това се променя постоянно, според това кое устройство предава в момента. Така че и двата края на линията трябва да бъдат терминирани с 100 Ом-ови терминатори.

4

Интерфейс RS485



BALANCED DIFFERENTIAL OUTPUT
LINE DRIVER



Vcm = Input Common Mode Voltage
Permissible Range for Vcm -7v > Vcm < +7v

BALANCED DIFFERENTIAL INPUT
LINE RECEIVER

Интерфейс RS485

Въпреки че диференциалните сигнали (каквито пренася RS485 мрежата) не изискват комуникиращ сигнал «земя», проводникът съединяващ заземяванията играе важна роля.

На разтояние от стотици метри е възможно да има значителна разлика между нивата на напрежение на «земята». RS485 мрежите могат да поддържат коректна комуникация при разлика от -7V до +12V. Но ако разликата в напрежителните нива на «земята» е по-голяма, това ще доведе до загуба на информация, както често и до повреда на порта.

Задачата на проводника, свързващ различните заземявания, е да осигурява еднаква напрежителна стойност на «земята» във всички комуникационни точки.

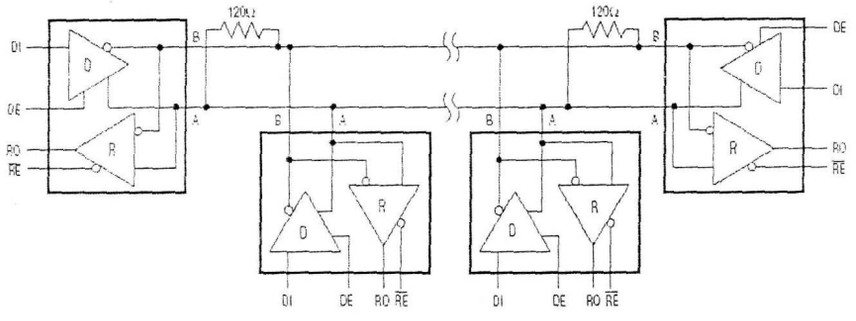
Опроводяване на RS485 мрежа

RS485 спецификацията означава проводниците за данни като „A" и „B", но много производители ги означават като „+" и „-„. В такъв случай проводник „A" трябва да се свърже с линия „+", а проводник „B" с линия „-„. Обръщането на полярността няма да повреди RS485 устройството, но то няма да може да комуникира. Останалото е лесно - винаги трябва да се свързват A с A и B с B.

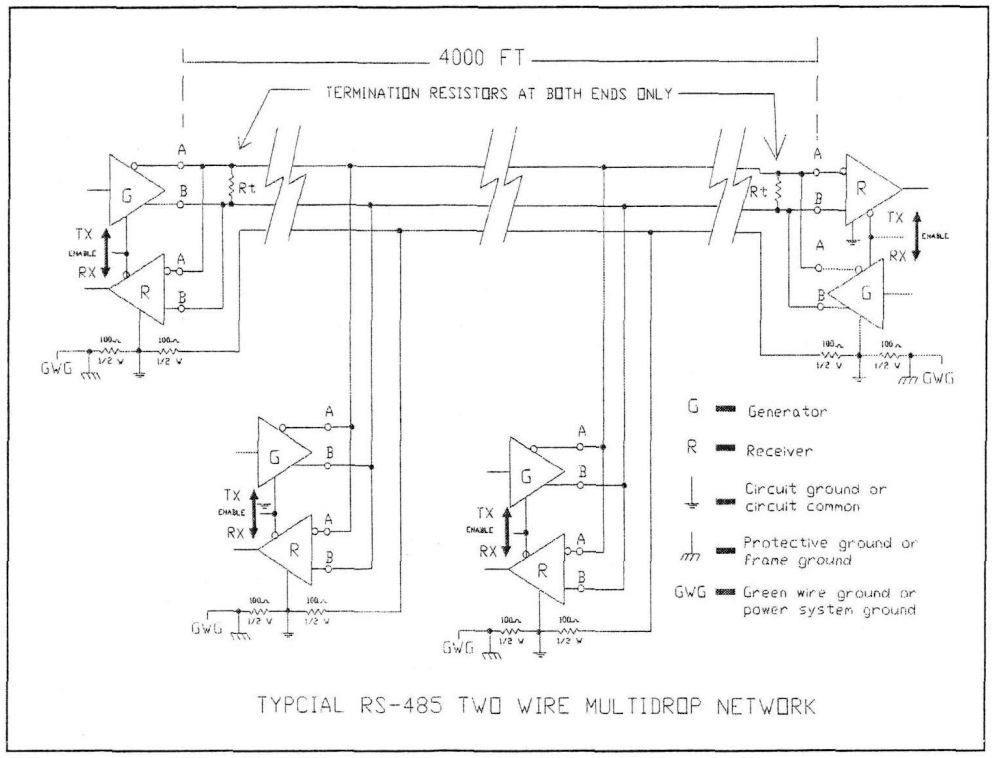
В случаи когато се налага включване на повече от 32 устройства, съществуват две основни възможности. По-простата е да се използват предаватели с 4 пъти входния импеданс (48к), позволяващи включването на 128 устройства едновременно. Такива предаватели днес са общодостъпни. В такъв случай трябва да се преоразмери терминиращата система на мрежата.

Другият вариант е да се използва RS485 повторител. Тези устройства имат два порта за усукана двойка проводници. Когато постъпят данни на едната усукана двойка, повторителят ги прехвърля на другия си порт и обратното. По този начин можем да свържем допълнително 31 или 127 устройства.

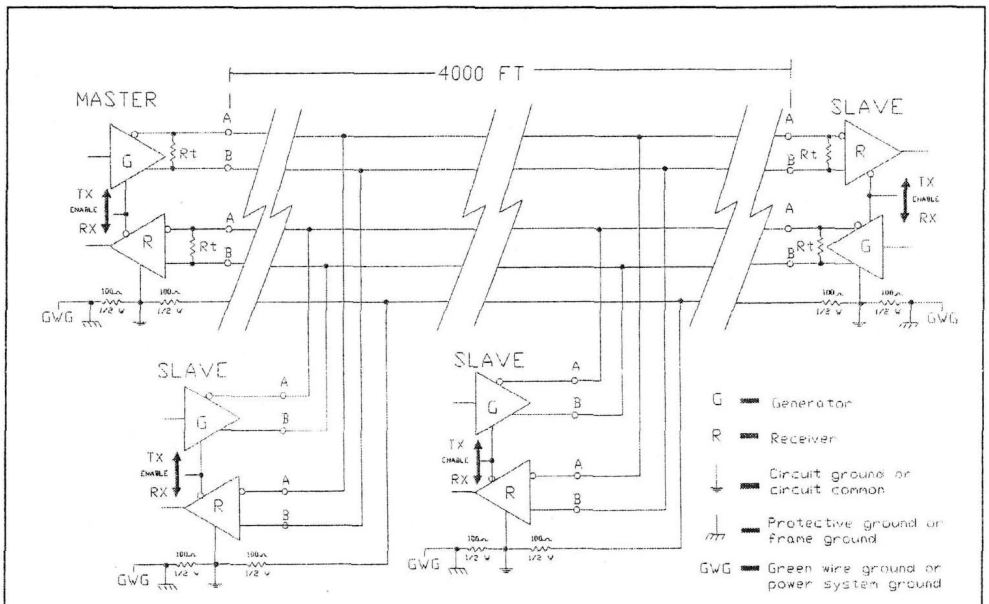
Свързване на RS485 в мрежа



Свързване на MAX485 в мрежа



TYPCIAL RS-485 TWO WIRE MULTIDROP NETWORK



TYPICAL RS-485 FOUR WIRE MULTDROP NETWORK

Интерфейс USB

Конвенционалните серийни портове са използвани още от най-ранните Персонални Компютри, като досега са претърпели известни промени, но не съществени. И макар да са функционални, те имат сериозни ограничения от страна на капацитет, софтуерна поддръжка и производителност.

В средата на 90-те години консорциум от някои от гигантите в производството на компютърна и комуникационна техника, включващ Compaq, IBM, Intel, Microsoft и NEC разработват нов интерфейсен стандарт за включване на сравнително нискоскоростни устройства към Персоналния Компютър. Интерфейсът е наречен Universal Serial Bus или USB.

Въпреки че поддръжката на USB в Персоналните Компютри датира отпреди доста години, едва наскоро започна да получава полагащото му се широко одобрение и възприемане. Сега всички нови компютри са оборудвани с USB поддръжка, а USB портове също така могат да бъдат добавяни и на много от старите компютърни системи.

11

Интерфейс USB

До 127 устройства могат да бъдат свързани чрез последователно включване или чрез употреба на USB хъб, който от своя страна има USB гнезда и се свързва към Персонален Компютър или друго устройство. Седем периферни устройства могат да бъдат свързани към всеки USB хъб. Това би могло да включва допълнителен USB хъб, който от своя страна също има капацитет за едновременно свързване на седем периферни устройства и т.н.

Заедно със сигнала, USB пренася и 5 V захранване, така че малки устройства, като мобилни скенер устройства или озвучителни тела, не е необходимо да притежават собствен захранващ кабел. Всъщност USB кабелът е четирипроводен. Състои се от усукана двойка за предаване на данните, заземен проводник и проводник осигуряващ +5 V захранване.

Заедно със сигнала, USB пренася и 5 V захранване, така че малки устройства, като мобилни скенер устройства или озвучителни тела, не е необходимо да притежават собствен захранващ кабел. Всъщност USB кабелът е четирипроводен. Състои се от усукана двойка за предаване на данните, заземен проводник и проводник осигуряващ +5 V захранване.

12

Интерфейс USB

USB1.1 преодоля скоростните ограничения на UART-базираните серийни портове, поддържайки 12 Mbps (разпределени между всички устройства, свързани в USB мрежата), сравнявайки се с мрежови технологии като Eternet и Token Ring и осигурявайки по-голяма от необходимата лентова ширина за периферните устройства, за които е предназначен. Например ширината на лентата му е в състояние да поддържа устройства като външни CD ROM и касетъчни архивиращи устройства.

И все пак, USB не е предназначен да замени мрежите. За да се запази цената му ниска, обхватът му е лимитиран до 5 метра между различните устройства. Скоростта на комуникация може да бъде ограничена до 1.5 Mbps за нискоскоростните устройства като клавиатура и мишка, с цел заделяне на капацитет за устройствата, които наистина имат нужда от висока скорост.

Като доказателство на възможностите на USB, от Април 2000 година беше обявена нова, USB 2.0 версия на стандарта, достигаща скорост на предаване на данни от внушителните 480 Mbps, позволяваща поддръжката на високоскоростен трансфер на информация.

13

Интерфейс USB

Като доказателство на възможностите на USB, от Април 2000 година беше обявена нова, USB 2.0 версия на стандарта, достигаща скорост на предаване на данни от внушителните 480 Mbps, позволяваща поддръжката на високоскоростен трансфер на информация.

По което и време да бъде включено периферно устройство, хоста открива напрежителната разлика в USB мрежата и извършва запитване към устройството за вида му, производител, функционалност и необходима скорост за обмен на информация. По този начин към устройството се прикрепя уникален идентификационен адрес (ID address). Дори ако две идентични устройства бъдат свързани едновременно, всяко от тях ще получи уникален адрес и ще може да бъде разпознавано от компютъра. Щом номерацията бъде завършена, подходящият за устройството драйвер се зарежда от операционната система и операторът бива помолен да предостави инсталационен диск, ако е необходимо.

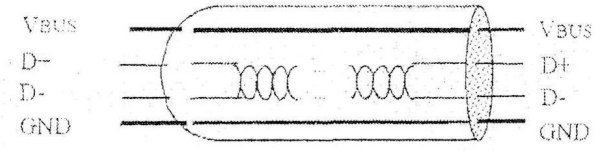
Всякакъв сблъсък между устройствата се разрешава от хоста или резидентен на хоста софтуер. Няма необходимост от конфигуриране на IRQ-та на прекъсване, адреси или DMA канали. Когато устройства се откачат от USB мрежата, хост компютърът открива откачането, подава сигнали на съответните приложения и разтоварва драйверите им.

14

Интерфейс USB

USB шината разпространява 0.5 A (500 mA) захранване чрез всеки свой порт. По този начин нискотокови устройства, които нормално се нуждаят от отделен AC адаптер, сега могат да бъдат захранвани направо чрез USB кабела -USB позволява на Персоналния Компютър да разпознава необходимото за устройството захранване и да го осигурява.

Хъбовете могат да получават захранване от USB шината или от свой собствен AC адаптер. USB хъбовете, комутиращи USB портове, осигуряват електрическа изолация за всеки порт, така че в случай на дефектирало свързано устройство, получило късо съединение, да бъдат предпазени останалите свързани устройства.



15

CAN интерфейс

Обменът на информация при CAN мрежите се осъществява чрез предаване на цифрови данни в сериен вид, което позволява връзката между модулите да се осъществява по два проводника. Предаването на съобщения по шината е от т.нар. радиоразпръсквателен тип (Broadcast Type Bus), при който всички модули могат да работят като предаватели, генерираните съобщения са без адрес и достигат до всички модули, работещи като приемници.

CAN е абревиатура на Controller Area Network. Скоростта на предаване на данни по шината е между 10kbit/s и 1Mbit /s в зависимост от стандарта който се използва и разстоянието, което трябва да се покрие. Предаваните съобщения са къси - максималният обем на полезната информация е 94 bit.

Сред спецификите на мрежата CAN е висока достоверност при предаването на информацията. Сигурността на работа в комбинация със сравнително ниската й цена е предпоставка за все по-широкото използване на CAN мрежите в различни отрасли на промишлеността. Друго предимство на CAN мрежите е възможността за разширяване на системата без никакви промени в програмното осигуряване.

16

CAN интерфейс

CAN може теоретично да свърже до 2032 устройства (ако на всеки възел се присвои отделен идентификатор). Поради практическото ограничение от хардуера обаче, могат да се свържат до 110 възела (с 82C250) в една мрежа. Скоростта на предаване е доста добра - 1Mbps, което позволява контрол в реално време. Допълнителните възможности като откриване и ограничаване на грешки го правят предпочитан в шумово критична среда.

Актуалната в момента спецификация е тази на Bosch версия 2.0, която е разделена на две части:

1. Стандартен CAN (версия 2.0A) - използва 11 битов идентификатор;
2. Разширен CAN (версия 2.0B) - използва 29 битов идентификатор.

CAN е multimaster мрежа.

17

CAN интерфейс

Съобщенията с данни, изпратени от който и да е CAN възел, **не съдържат адресите на предаващия и приемащия възел**. Вместо това съдържанието на **съобщението е маркирано с идентификатор**, който е уникален за тази мрежа. Всички останали възли от мрежата получават съобщението и прилагат тест за приемане на съобщението, за да определят дали съдържанието му ги "интересува". Ако тестът е положителен, съобщението се приема, а ако е отрица¬телен - то се отхвърля.

Уникалният идентификатор определя и приоритета на съобщението. Идентификаторът с най-малък номер (по стойност) има най-голям приоритет. Това позволява арбитриране, когато два или повече възела искат достъп до магистралата едновременно.

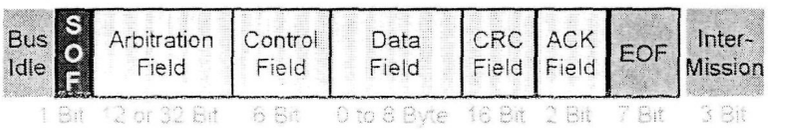
Гарантирано е, че съобщението с най-голям приоритет ще има достъп до магистралата така, както би било и ако е единственото съобщение, което се предава. Съобщението с по-малък приоритет автоматично ще бъде препредадено в следващия магистрален цикъл или в по-следващия, ако има още съобщения с по-голям приоритет.

18

CAN интерфейс

Съществуващите CAN протоколи са над 30. В основата на всеки протокол, независимо от различията между тях, е стандартният протокол CAN - Standard CAN (CAN2.0A) и разширеният протокол CAN - Extended CAN (CAN2.0B). При стандартния CAN протокол цялата обменяна по една CAN мрежа информация се състои по принцип от четири вида пакети - за **данни (Data Frame)**, за **искане на данни (Remote Frame), пакет за грешки (Error Frame)** и пакет за **претоварване (Overload Frame).**

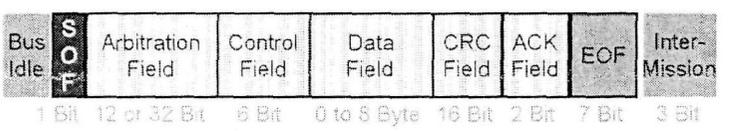
Информационната рамка (**Data Frame**) има вида:



Първите два пакета се разделят от останалите чрез поле за съобщения (Intermission Field) или наличие на свободна шина (Bus-Idle). При останалите два пакета не е необходимо подобно разделяне. Пакетът за данни е носителят на информацията, т.е. всички останали пакети са спомагателни.

19

Data Frame



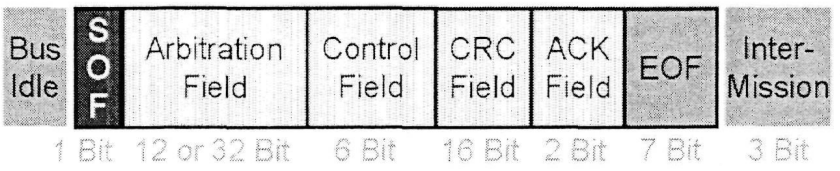
Основният пакет е съставен от седем пакета с различно предназначение и продължителност:

- **SOF** - *Начало на рамка. Бит за твърда синхронизация на мрежата с ниско ниво*
- **Arbitration Field** - поле за арбитраж на съобщения. Това поле е 12 битово (при CAN2.0A) или 32 битово (при CAN2.0B). При CAN2.0A освен идентификатора (11бита) е въведен и допълнителен бит наречен RTR. Той показва каква информация носи пакета Data 4 field. Ако RTR е с ниско ниво, то се предават данни, а ако е с високо ниво, то имаме Remote Frame. При CAN2.0B идентификатора е 29 битов и 3 контролни бита SRR,IDE, RTR.
- **Control Field** – поле което оказва колко са информационните байтове в съобщението което се предава.
- **Data Field** – информационно поле.
- CRC Field – поле за проверка на евентуални грешки при предаването. Извършва се сложна обработка на данните. Данните се разглеждат като полином, който се обработва от предавателя и се изпраща заедно с 1 контролен бит. Приемникът прави същата обработка на приетите от него данни и сравнява резултата с контролният бит. По този начин се откриват грешки при предаване.
- **ACK Field** – поле за потвърждаване на получено съобщение.
- **EOF** – край на предаването.

20

Remote Frame.

Предназначението на пакета за искане на данни е да даде възможност на всеки модул не само пасивно да чака съобщения, но и да накара съответния предавател да ги получи. Формата е същият като при Data Frame с променено ниво на RTR бита и без Data field.



21

Error Frame

Състои се от Error Flag, който е 6 битов с една и съща стойност (0), и така нареченият Error Delimiter, който е 8 битов. Последният се въвежда за да осигури известно време на останалите участници в мрежата да изпратят своите Error Flag's, след като приемат първият сигнал за грешка. При CAN има въведени пет 5 механизма за откриване на грешки – три от тях на ниво съобщение и два на битово ниво. На ниво съобщение това са CRC, Frame Checks, cknowledgment Error Checks. На битово ниво това са Bit Monitoring, Bit Stuffing. Използват се различни механизми и подходи за да се гарантира достоверността на обменяната информация.

Overload Frame

Генерира се в случай че приемника не е в състояние да приеме следващото съобщение, тъй като не е обработил предходното. Друг случай за генериране на препълване е когато приемника не си получи коректен сигнал за край на предаването (EOF).

22

Арбитриране на съобщения

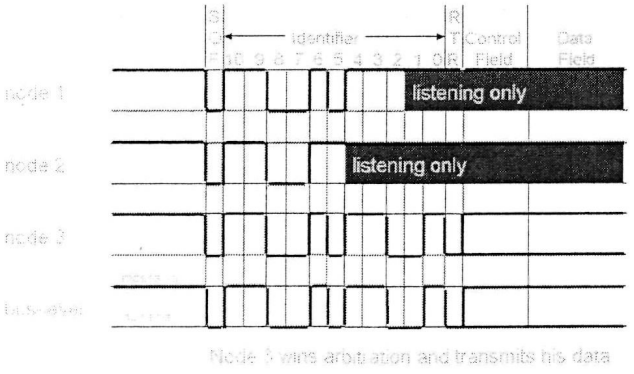
Принципът на работа на мрежите CAN се базира на обстоятелството, че при наличие на свободна шина (в момента по нея не се предава информация) всеки модул може да започне да предава съобщения. Тъй като всяко съобщение носи в себе си приоритет, в случай че няколко модула предават едновременно, достъп до шината получава съобщението с най-голям приоритет.

Аналогично, когато до даден модул едновременно достигнат повече от едно съобщение, приемането се извършва в зависимост от приоритета им. Всеки от модулите, включени в мрежата CAN, би могъл да изиска получаването на съобщения, като изпраща запитвания. На всяко правилно прието съобщение се изпраща потвърждение (acknowledgment).

В случаите, при които дадено съобщение не се приеме поради грешка, се изпраща отрицателен отговор. CAN мрежата разпознава появата на постоянни грешки и автоматично изключва повредения модул.

23

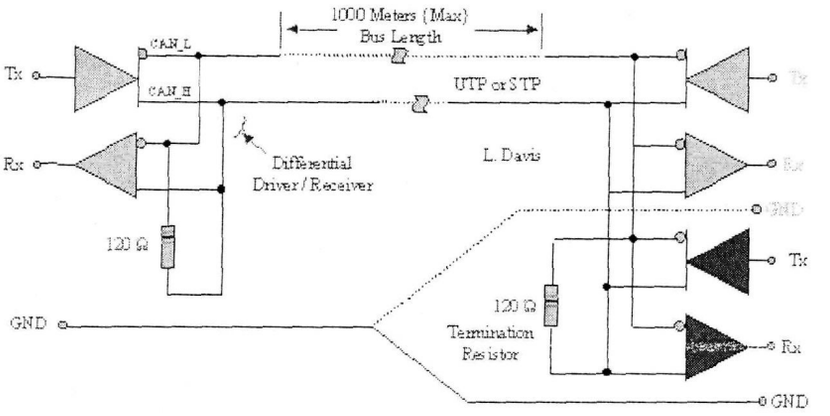
Като пример за определяне приоритета на предавателя в мрежата може да се разгледа схемата в която са включени три модула:



До бит 5 всички модули изпращат към мрежата и прочитат от нея едни и същи данни. При бит 5 от Identifier блока, модули 1 и 3 изпращат заявка за заемане на линията. През това време модул 2 изпраща високо ниво а прочита ниско. При това модул 2 преминава в режим на изчакване и "слуша линията за връзка". При бит 2 от Identifier блока, модул 1 губи арбитража с модул 3 заради по-високият приоритет, който има той.

24

CAN Bus Electrical Interface Circuit



25

CAN Bus I/O Characteristics

CANbus Signal Type	Digital Interface
Output Voltage (High)	V_{OH} +4 volts min, +5.5 volts max
Output Voltage (Low)	V_{OL} +0 volts min, +1.5 volts max
Output Voltage	+16 volts (Absolute Max)
Output Current	100mA
Impedance	124 ohm termination between +/- terminals
Circuit Type	Differential
Bit Times	1uS @ 1Mb/s; 2uS @ .5Mb/s 4uS @ .25Mb/s
Encoding Format	Non-Return-to-Zero (NRZ)
Transmit/Receive Frequency	1Mb/s @ 40 meters
Topology	Point-to-Point
Medium	Shielded Twisted Pair (STP) @ 9 pin D-Sub
Access Control	Carrier Sense, Multiple Access with Collision Detect (CSMA/CD). None destructive bit wise arbitration

Физическият слой не е част от Bosch CAN стандарта. Използват се ISO стандарти. CAN магистралата се състои от два проводника - CAN_High и CAN_Low. Тези два проводника работят в дефиренциален режим за да се увеличи шумоустойчивостта.

26

Round Connector Pin out

Many of the additional connector pin outs are used with CANopen and include: 10-pin header [5 x 2 multipole], RJ10 [Modular Connector Jack], RJ45 [Modular Connector Jack], 5-pin mini [circular], 5-pin micro [circular], Open Style, 7/8/9-pin round connectors.

9 Pin (male) D-Sub CANbus PinOut

Pin #	Signal Names	Signal Description
1	Reserved	Upgrade Path
2	CAN_L	Dominant Low
3	CAN_GND	Ground
4	Reserved	Upgrade Path
5	CAN_SHLD	Shield, Optional
6	GND	Ground, Optional
7	CAN_H	Dominant High
8	Reserved	Upgrade Path
9	CAN_V+	Power, Optional