Realizarea unui sistem de timp real cu microcontroler AVR

Aceasta lucrare ilustrează modul în care se poate realiza un sistem de timp real utilizînd microcontrolerul AVR AT90S8515 realizat de firma Atmel.

Lucrarea este structurată în 3 secțiuni:

- descrierea microcontrolerului AT90S8515
- descrierea hardware-lui de dezvoltare STK500
- descrierea unei aplicații softaware: planificatorul de procese şi procesele secvențiale

I. Descrierea microcontrolerului AT90S8515

AT90S8515 este un microcontroler CMOS pe 8 biți, produs de ATMEL, realizat în arhitectura AVR RISC.

Caracteristicile microcontrolerului AT90S8515 sînt:

- Arhitectură RISC:
  - 118 instrucțiuni, cele mai multe fiind executate într-un ciclu;
  - 32 registre de uz general pe 8 biți;
  - până la 8MIPS la o frecvență de 8 MHz.
- Memorie:
  - 8 K Bytes In-System Programmable Flash (Memorie de Program);
  - 512 Bytes memorie SRAM;
  - 512 Bytes In-System Programmable EEPROM;
  - Protecție la citire pentru Flash şi EEPROM.
- Module periferice:
  - 1 Timer/Counter pe 8 biți cu prescaler separat;
  - 1 Timer/Counter pe 16 biți cu prescaler separat şi diverse moduri de lucru;
  - Comparator analogic;
  - Watchdog Timer programabil cu oscilator RC intern;
  - Port serial UART programabil;
  - Interfață serială SPI master/slave.
- 32 de linii I/O programabile.
- Întreruperi externe şi interne.
- Moduri “Low Power Idle” şi “Power Down”.
- Frecvență de ceas până la 8MHz.

Diagrama acestui microcontroler este prezentată în figura 1.

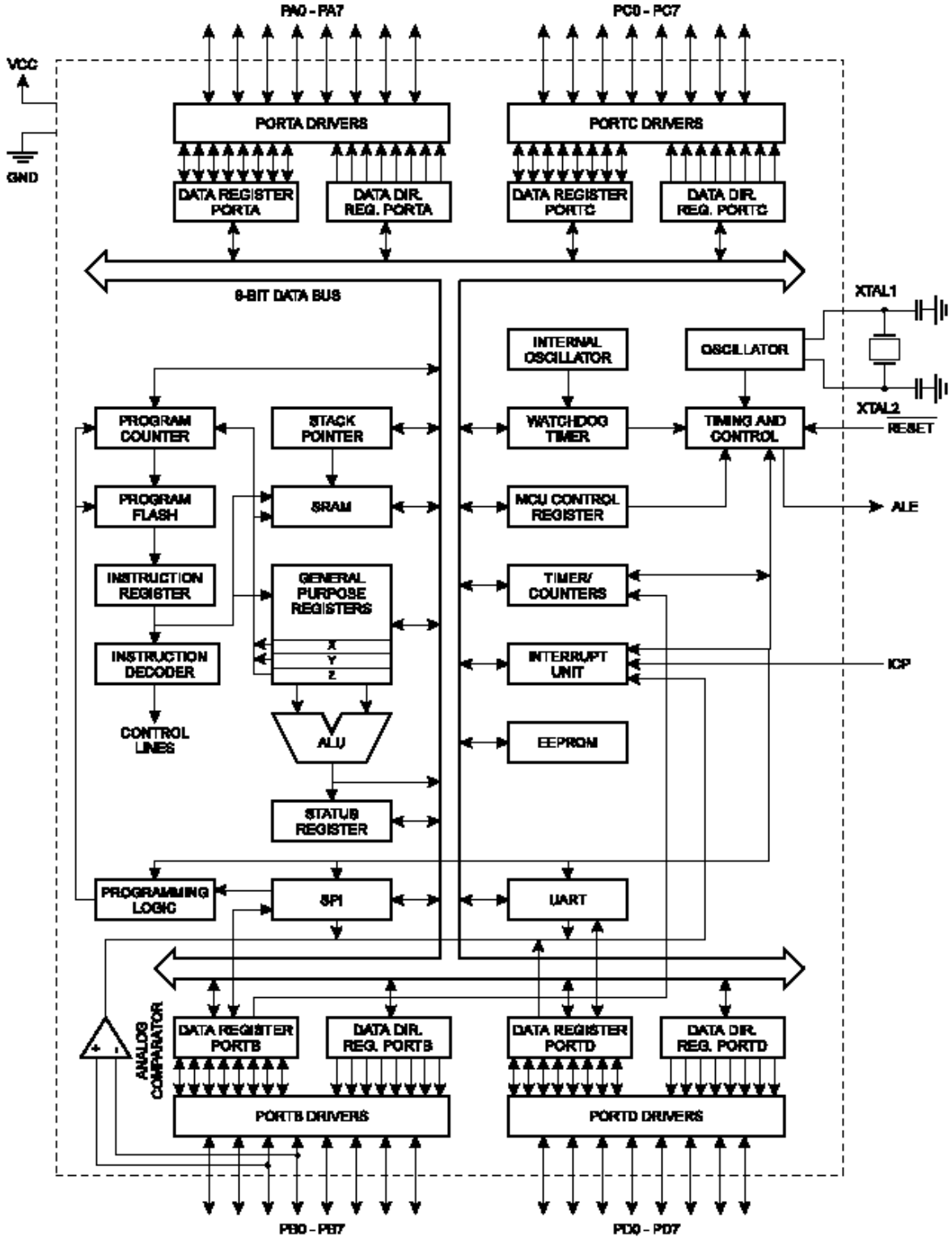


Figura 1: Diagrama bloc a lui AT90S8515

În figura 2 este prezentată arhitectura simplificată a microcontrolerului AT90S8515.

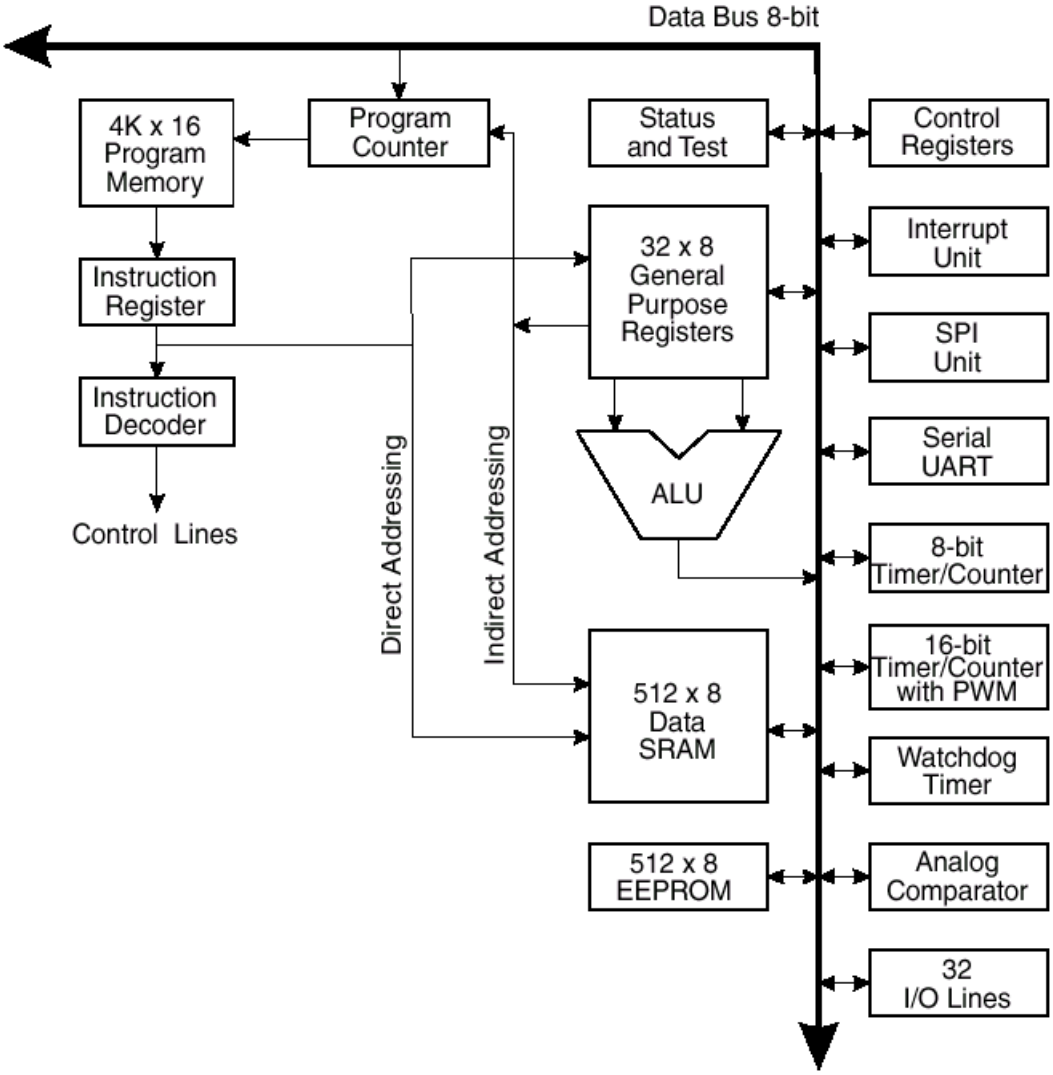


Figura 2: Arhitectura AVR

Arhitectura AVR este o arhitectură Harvard cu bus-uri separate pentru program şi date. În timpul execuţiei unei instrucţiuni, instrucţiunea următoare este adusă din memoria de program (pipe-line cu două nivele). Tehnica pipe-line permite ca instrucţiunile să fie executate la fiecare ciclu de tact. Memoria de program este de tip Flash programabilă intern.

Conceptul de "registru cu acces rapid" aduce 32x8 de registre de lucru cu timp de acces rapid. Aceasta înseamnă o operaţie aritmetică logică este executată într-un singur ciclu de către ALU (Arithmetic Logic Unit). Cei doi operanzi sunt extraşi din registre, se execută operaţia şi rezultatul este stocat în registrul destinaţie, toate acestea într-un singur ciclu. O parte dintre cele 32 registre, mai exact 6, pot fi folosite ca 3 pointeri pe 16 biţi pentru adresare relativă în memoria de date SRAM. Aceste registre sunt X,Y şi Z. ALU permite efectuarea operaţiilor logice şi aritmetice cu un registru, între două registre sau între un registru şi o constantă. În plus registrele de lucru pot fi accesate de prin intermediul celor trei pointeri ca oricare locaţie de memorie. Acest lucru este posibil deoarece registrelor le sunt alocate primele 32 de adrese din spaţiul memoriei de date. Spaţiul locaţiilor de memorie I/O conţine 64 de adrese pentru dispozitivele periferice ale microcontrolerului ca: registre de control, Timere/Countere, convertoare A/D, porturi I/O şi alte funcţii I/O. Locaţiile de memorie I/O

pot fi adresate direct sau indirect prin cei trei pointeri între adresele 0x20-0x5F. Cu instrucțiunile ***rjmp*** şi ***rcall*** se poate adresa direct tot spațiul de memorie de program, internă de 4K. Toate instrucțiunile AVR au un format pe 16 biți. Fiecare adresă de memorie de program conține o instrucțiune pe 16 sau 32 biți. În timpul întreruperilor şi al apelurilor de subrutină adresa de revenire este salvată în stivă. Stiva se află în spațiul de memorie SRAM şi este limitată doar de dimensiunea acestei memorii. Pointerul stivei SP este accesibil în spațiul de memorie I/O. Cele 512 locații de memorie pot fi accesate uşor prin 5 metode diferite.

Memoria de date este ilustrată în figura 3:

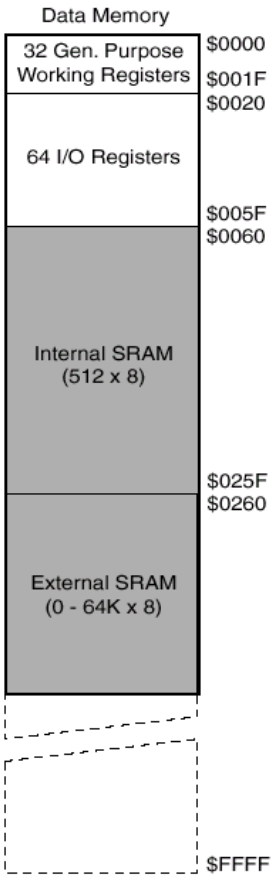


Figura 3: Organizarea memoriei de date

În continuare se vor detalia următoarele caracteristici ale microcontrolerului AT90S8515 precum si dispozitivele periferice:

1. Registrele de lucru de uz general;
2. Memoria de program FLASH;
3. Memoria de date SRAM internă şi externă;
4. Moduri de adresare a memoriei de program şi de date;
5. Timpii de acces ai memoriei, execuția în timp a instrucțiunilor;
6. Memoria I/O;
7. Resetul şi înteruperile;
8. Moduri “Sleep”;
9. Porturile I/O.

1. Registrele de lucru de uz general.

Organizarea acestora este prezentată în figura 4:

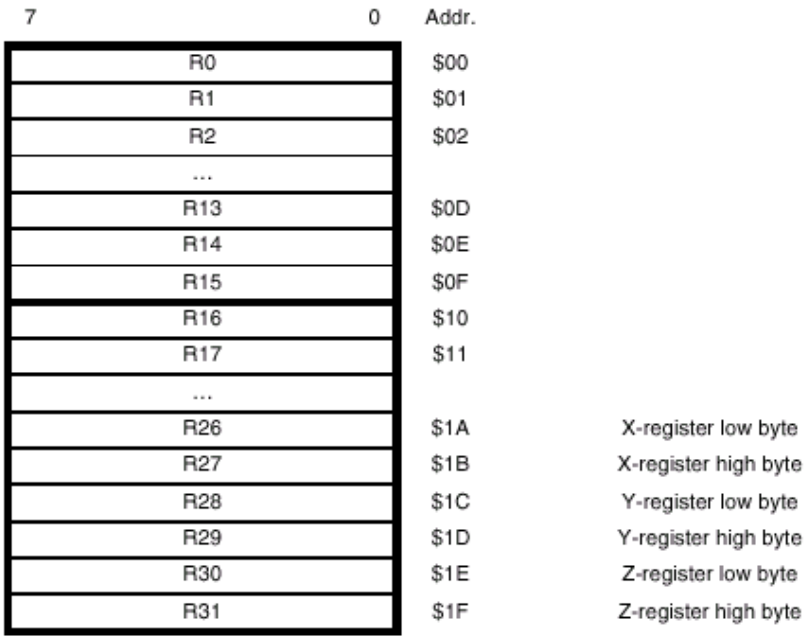


Figura 4: Organizarea registrelor de lucru

Toate instrucțiunile care operează cu registre au acces direct într-un singur ciclu la toate aceste registre. Singurele excepții sunt instrucțiunile care operează cu un registru şi o constantă. Acestea pot folosi doar registrele din a doua jumătate a spațiului registrelor de lucru şi anume R16..R31.

După cum reiese din figura 4 fiecărui registru îi este alocată o adresă din spațiul memoriei de date în primele 32 locații. Deşi nu sunt implementate fizic ca locații de memorie SRAM, această organizare permite o largă flexibilitate în accesarea acestor registre.

Registrele R26..R31 au în plus nişte funcții pe lângă cele de uz general. Aceste registre sunt X, Y, Z şi sunt pointeri pentru adresarea indirectă a locațiilor din spațiul memoriei de date.

2. Memoria de program FLASH

Microcontrolerul AT90S8515 are înglobați 8K Bytes de memorie de program programabilă intern. Deoarece toate instrucțiunile sunt pe 16 sau 32 biți, această memorie este organizată ca 4K x 16 biți. Registrul "Program Counter" are 12 biți, deci se pot adresa direct 4096 de locații. Anduranța memoriei de program (flash) este cel puțin 1000 de cicluri scriere/citire.

3. Memoria de date SRAM internă şi externă

Această memorie are o structură ca în figura 5:

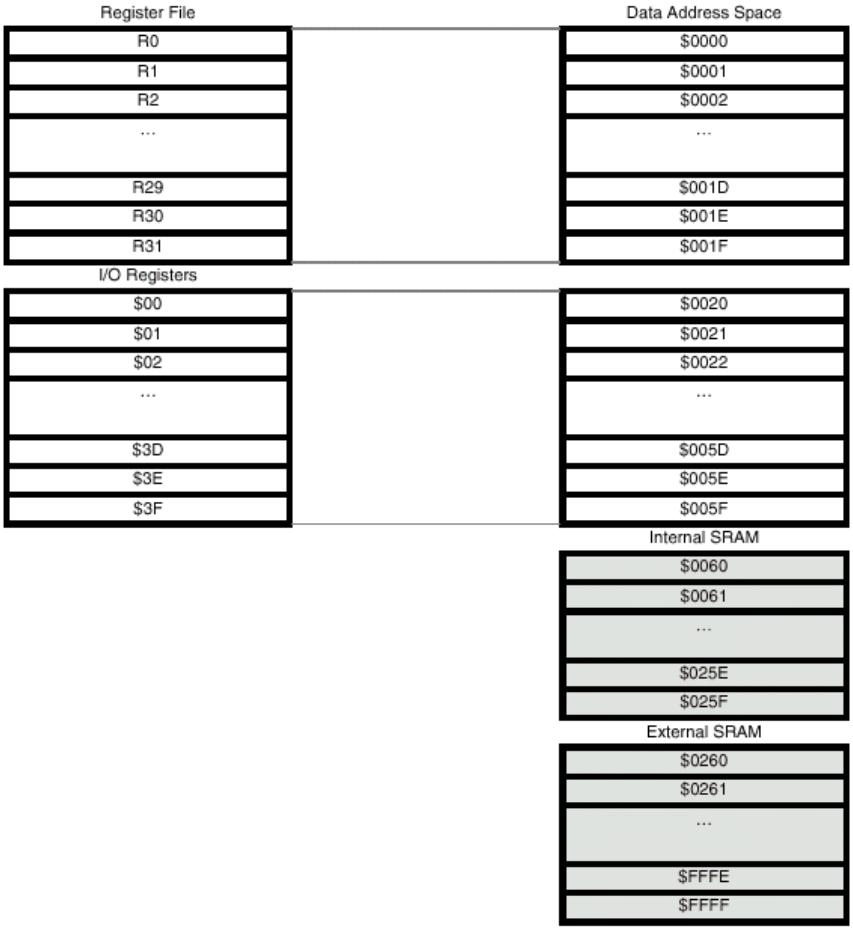


Figura 5: Organizarea memoriei SRAM

Primele 608 locaţii ale memoriei de date cuprind Registrele de lucru (32), Memoria I/O (64) şi memoria internă SRAM (următoarele 512 locaţii). O memorie externă opţională poate fi ataşată şi ocupă spaţiul de adrese cuprins între adresa superioară a memoriei SRAM interne şi 64K-1. Locaţiile din memoria SRAM externă pot fi accesate de aceleaşi instrucţiuni ca cele din memoria internă, singura diferenţă fiind că la accesarea unei locaţii de memorie externă există o întârziere de un ciclu maşină faţă de accesarea unei locaţii de memorie internă. De asemenea, dacă stiva este situată în spaţiul extern SRAM la întreruperile şi apelurilor subrutinelor se vor adăuga încă două cicluri maşină, deoarece adresele de program (2 octeţi) vor fi introduse sau extrase din stivă.

Accesarea memoriei externe SRAM implică setarea bitului SRE din registrul I/O MCUCR şi folosirea pinilor RD şi WR. Pentru acoperirea întregului spaţiu de 64K SRAM pot fi folosite toate modalităţile de adresare directă şi indirectă.

4. Moduri de adresare a memoriei de program şi de date

Modurile de adresare a memoriei de date sunt:

- **Adresare directă**
  - a unui registru de lucru – adresa registrului este inclusă în codul instrucțiunii (pe 5 biți);
  - a două registre de lucru – adresele celor două registre sunt incluse în codul instrucțiunii (pe câte 5 biți);
  - a unei locații de memorie I/O – adresa locației I/O (pe 6 biți) împreună cu adresa registrului destinație (pe 5 biți) sunt incluse în codul instrucțiunii;
  - a unei locații din memoria SRAM – adresa locației SRAM pe 16 biți reprezintă al doilea cuvânt din cele două cuvinte pe 16 biți ale instrucțiunii;
- **Adresare indirectă**
  - a unei locații din memoria SRAM prin pointerii X,Y, Z – adresa locației SRAM pe 16 biți se află într-unul din pointerii X, Y, Z;
  - a unei locații din memoria SRAM cu înlocuire prin pointerii Y, Z – adresa locației SRAM se obține prin adunarea unui număr de 6 biți (aflat în codul instrucțiunii) la unul ditre pointerii Y, Z;
  - a unei locații din memoria SRAM cu predecrementare (X, Y, Z) – adresa locației SRAM conținută în unul dintre pointerii X, Y, Z este decrementată înainte de efectuarea operației RD/WR asupra locației;
  - a unei locații din memoria SRAM cu postincrementare (X, Y, Y) – adresa locației SRAM conținută în unul dintre pointerii X, Y, Z este incrementată după efectuarea operației RD/WR asupra locației;
  - a unei constante din memoria de program folosind instrucțiunea LPM – adresa constantei se află în pointerul Z pe 15 biți, iar din LSB se selectează octetul inferior dacă LSB=0 sau superior dacă LSB=1.

Modurile de adresare a memoriei de program sunt:

- Adresarea indirectă prin instrucțiunile ***ijmp*** şi ***icall*** – execuția programului continuă de la adresa aflată în pointerul Z;
- Adresarea relativă prin instrucțiunile ***rjmp*** şi ***rcall*** – execuția programului continuă de la o adresă obținută astfel: PC+k+1 unde PC este adresa curentă de program, k o constantă pe 12 biți cu semn conținută în codul instrucțiunii (-2048<k<2047).

5. Timpii de acces ai memoriei, execuţia în timp a instrucţiunilor

Microcontrolerul este controlat direct de ceasul generat de un cristal extern fără nici o divizare. În figura 6 este ilustrat paralelismul dintre execuția instrucțiunilor şi aducerea acestora din memorie (fetch), paralelism relizat pe baza arhitecturii Harvard şi a conceptului de acces rapid la registre. Această tehnică numită şi pipe-line permite obținerea a unui MIPS/1MHz

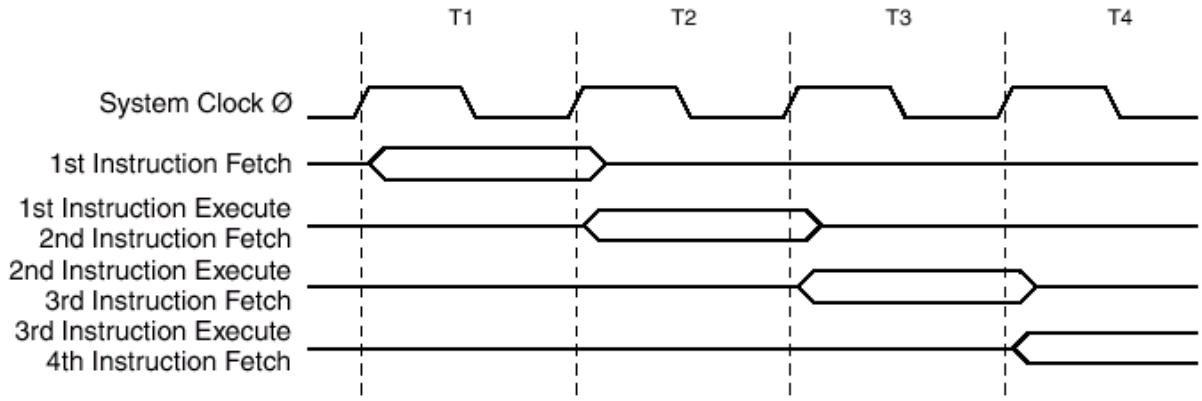


Figura 6: Tehnica pipe-line cu două nivele

În figura 7 este ilustrat modul în care este efectuată o operaţie aritmetică-logică folosind două registre şi salvat rezultatul în registrul destinaţie într-un singur ciclu.

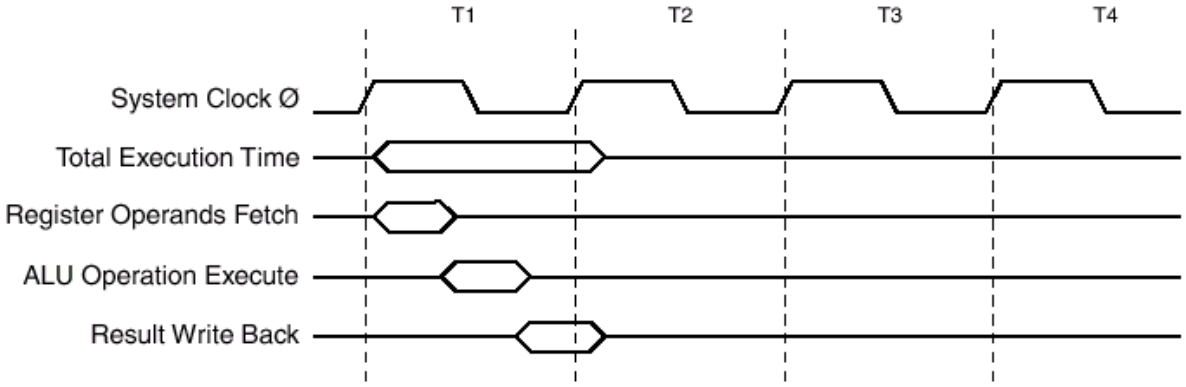


Figura 7: Modul de execuţie a unei operaţii aritmetice-logice

6. Memoria I/O

Porturile I/O şi dispozitivele periferice sunt situate în spaţiul de adrese I/O 0x20-0x5F. Locaţiile din acestă zonă de memorie pot fi accesate de instrucţiunile ***in*** şi ***out*** care transferă date între registrele de lucru şi spaţiul I/O. Primele 16 registre pot fi accesate direct de instrucţiunile ***sbi*** şi ***cbi*** care acţionează la nivel de bit.

Spaţiul I/O este definit ca în tabelul următor, unde în paranteze este reprezentată adresa locaţiei ca memorie SRAM:

Address Hex	Name	Function
$3F ($5F)	SREG	Status Register
$3E ($5E)	SPH	Stack Pointer High
$3D ($5D)	SPL	Stack Pointer Low
$3B ($5B)	GIMSK	General Interrupt Mask register
$3A ($5A)	GIFR	General Interrupt Flag Register
$39 ($59)	TIMSK	Timer/Counter Interrupt Mask register
$38 ($58)	TIFR	Timer/Counter Interrupt Flag register
$35 ($55)	MCUCR	MCU general Control Register
$33 ($53)	TCCR0	Timer/Counter0 Control Register
$32 ($52)	TCNT0	Timer/Counter0 (8-bit)
$2F ($4F)	TCCR1A	Timer/Counter1 Control Register A
$2E ($4E)	TCCR1B	Timer/Counter1 Control Register B
$2D ($4D)	TCNT1H	Timer/Counter1 High Byte
$2C ($4C)	TCNT1L	Timer/Counter1 Low Byte
$2B ($4B)	OCR1AH	Timer/Counter1 Output Compare Register A High Byte
$2A ($4A)	OCR1AL	Timer/Counter1 Output Compare Register A Low Byte
$29 ($49)	OCR1BH	Timer/Counter1 Output Compare Register B High Byte
$28 ($48)	OCR1BL	Timer/Counter1 Output Compare Register B Low Byte
$25 ($45)	ICR1H	T/C 1 Input Capture Register High Byte
$24 ($44)	ICR1L	T/C 1 Input Capture Register Low Byte
$21 ($41)	WDTCR	Watchdog Timer Control Register
$1F ($3E)	EEARH	EEPROM Address Register High Byte (AT90S8515)
$1E ($3E)	EEARL	EEPROM Address Register Low Byte
$1D ($3D)	EEDR	EEPROM Data Register
$1C ($3C)	EECR	EEPROM Control Register
$1B ($3B)	PORTA	Data Register, Port A
$1A ($3A)	DDRA	Data Direction Register, Port A
$19 ($39)	PINA	Input Pins, Port A
$18 ($38)	PORTB	Data Register, Port B
$17 ($37)	DDRB	Data Direction Register, Port B
$16 ($36)	PINB	Input Pins, Port B
$15 ($35)	PORTC	Data Register, Port C
$14 ($34)	DDRC	Data Direction Register, Port C
$13 ($33)	PINC	Input Pins, Port C

Address Hex	Name	Function
$12 ($32)	PORTD	Data Register, Port D
$11 ($31)	DDRD	Data Direction Register, Port D
$10 ($30)	PIND	Input Pins, Port D
$0F ($2F)	SPDR	SPI I/O Data Register
$0E ($2E)	SPSR	SPI Status Register
$0D ($2D)	SPCR	SPI Control Register
$0C ($2C)	UDR	UART I/O Data Register
$0B ($2B)	USR	UART Status Register
$0A ($2A)	UCR	UART Control Register
$09 ($29)	UBRR	UART Baud Rate Register
$08 ($28)	ACSR	Analog Comparator Control and Status Register

Dintre toate aceste registre vor fi detaliate ca exemple în continuare: SREG (Status Register) şi SP (Stack Pointer).

SREG este situat la adresa 0x3F în spațiul I/O sau 0x5F în SRAM.

El este definit astfel:

Bit	7	6	5	4	3	2	1	0	
$3F ($5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

unde:

- I – Global Interrupt Enable este bitul ce maschează toate întreruperile;
- T – Bit Copy Storage este un bit în cere se poate copia orice bit din locațiile SRAM pentru testare;
- H – Half Carry Bit este un bit folosit în operațiile aritmetice;
- S – Sign Bit bitul de semn;
- V – Overflow Bit în complement față de 2.
- N – Negative Flag este un bit care indică rezultatul negativ în urma unei operații aritmetice sau logice;
- Z – Zero Flag indică rezultatul 0 al unei operații aritmetice sau logice;
- C – Carry Flag indică carry în urma unei operații aritmetice sau logice.

SP este situat la adresa 0x3E (0x5E) şi 0x3D (0x5D).

Acest registru este constituit efectiv din 2 locații de 8 biți SPH şi SPL alcătuind astfel un pointer pe 16 biți al vârfului stivei după cum se poate vedea în figura următoare:

Bit	15	14	13	12	11	10	9	8	
$3E ($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

SP se decrementează la fiecare introducere de date în stivă şi se incrementează la fiecare extragere din stivă.

7. Reset-ul şi gestionarea întreruperilor

AT90S8515 are trei surse de Reset:

- Power-on Reset – microcontrolerul este resetat când tensiunea de alimentare este mai mică de un anumit prag (V POT);
- Reset extern – microcontrolerul este resetat de un nivel logic zero prezent pe pinul RESET pe o durată mare de 50nS;
- Watchdog Reset – microcontrolerul este resetat când a expirat perioada modulului Watchdog Timer dacă acest lucru este validat. În urma unui Reset toate locațiile din spațiul I/O sunt setate la valorile lor inițiale şi programul începe de la adresa 0x000.

AT90S8515 dispune de 12 surse de întrerupere fiecare din acestea având asociat un vector de întrerupere în memoria de program. De asemenea pentru fiecare întrerupere există un bit de mascare care trebuie setat împreună cu bitul SREG[I] pentru a valida acea întrerupere. Lista tuturor vectorilor este prezentată în tabelul următor:

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Reset, Power-on Reset and Watchdog Reset
2	$001	INT0	External Interrupt Request 0
3	$002	INT1	External Interrupt Request 1
4	$003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	$004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	$005	TIMER1 COMPB	Timer/Counter1 Compare Match B
7	$006	TIMER1 OVF	Timer/Counter1 Overflow
8	$007	TIMER0, OVF	Timer/Counter0 Overflow
9	$008	SPI, STC	Serial Transfer Complete
10	$009	UART, RX	UART, Rx Complete
11	$00A	UART, UDRE	UART Data Register Empty
12	$00B	UART, TX	UART, Tx Complete
13	$00C	ANA_COMP	Analog Comparator

Din acest tabel reiese şi nivelul de prioritate al întreruperilor. Cu cât adresa vectorului este mai mică cu atât prioritatea este mai mare. Astfel prioritatea cea mai mare o are întreruperea INT0 (în afară de Reset), iar prioritatea cea mai mică o are întreruperea ANA_COMP.

Sistemul de întreruperi este controlat de o serie de registre: General Interrupt Mask Register – GIMSK, General Interrupt Flag Register – GIFR, Timer/Counter Interrupt Mask Register – TIMSK şi Timer/Counter Interrupt Flag Register – TIFR. Când apare o întrerupere bitul Global Interrupt Enable SREG[I] este resetat (zero) şi toate întreruperile sun mascate. Utilizatorul poate seta acest bit pentru a valida eventualele întreruperi ce pot apărea în timpul execuției rutinei de servire a întreruperii curente. La execuția instrucțiunii ***reti*** bitul I va fi setat automat.

Latența oricărei întreruperi este de minim patru cicluri maşină. În acest timp adresa PC este salvată în stivă şi SP este decrementat cu doi. La revenirea din rutina de întrerupere lucrurile se întâmplă exect invers: PC este extras din stivă şi SP este incrementat cu doi. Registrul SREG nu este introdus în stivă, de aceesa acest registru trebuie salvat.

8. Moduri “Sleep”

Există două moduri:

- Idle Mode – execuţia instrucţiunii ***sleep*** opreşte execuţia programului, dar dispozitivele periferice (Timere, sistemul de întreruperi, etc) funcţionează. Acest lucru face ca microcontrolerul să poată fi scos din modul “Idle” de întreruperi externe şi interne, ca de exemplu trecerea prin 0 a unui timer.
- Power Down Mode – în acest mod oscilatorul extern şi dispozitivele periferice sunt blocate şi microcontrolerul poate fi scos din acest mod doar de o întrerupere externă, de un Reset extern sau de Watchdog Reset.

La apariţia evenimentului care repune microcontrolerul în funcţiune, acesta reia execuţia programului cu rutina de întrerupere sau cu rutina de Reset dacă microcontrolerul a primit vreun Reset.

În timpul în care microcontrolerul se află în unul dintre cele două moduri, conţinutul registrelor de lucru, registrelor I/O şi al locaţiilor SRAM rămâne nemodificat.

9. Porturile I/O

AT90S8515 dipune de 32 pini I/O organizaţi în patru porturi: PortA, PortB, PortC, PortD. Toate aceste porturi sunt bidirecţionale. Fiecărui port îi sunt asociate câte trei registre:

- Data Register – PORTx;
- Data Direction Register – DDRx registrul din care se setează direcţia portului individual pentru fiecare pin ;
- Port x Input Pins – PINx registrul din care se citeşte portul respectiv.

Toate porturile au rezistoare interne legate la tensiunea de alimentare “pull-ups” (pentru a nu lăsa pinii setaţi ca intrări în gol) şi au bufere de ieşire care pot da un curent de 20mA. Funcţiile generale I/O ale porturilor sunt multiplexate cu funcţii speciale asociate dispozitivelor periferice, accesului memoriei externe, întreruperilor, etc.

Dispozitivele periferice

În continuare vor fi descrise pe scurt dispozitivele periferice ale microcontrolerului AT90S8515.

Acestea sunt:

1. Modulele Timer/Conters;
2. Watchdog Timer;
3. Memoria de date EEPROM;
4. Serial Peripheral Interface (SPI);
5. Universal Asynchronus Receiver Transmiter (UART);
6. Analog Comparator;
7. Interfaţa cu memoria externă.

1. Modulele Timer/Conters

AT90S8515 dispune de două numărătoare/timere de uz general, unul de 8 biţi şi unul de 16 biţi. Fiecăruia i se poate selecta o valoare individuală pentru prescaler din acelaşi prescaler de 10 biţi. Ambele module pot fi folosite ca timere cu un clock generat intern sau ca numărătoare cu un clock extern de la un pin I/O. Cele 4 valori pentru prescaler sunt CK/8, CK/64, CK/256 şi CK/1024 unde CK este clock-ul dat de oscilatorul extern.

Ambele timere pot genera întreruperi ale căror biţi de validare se află în registrul Timer/Counter Interrupt Mask Register (TIMSK) şi ale căror flaguri de întrerupere se află in registrul Timer/Counter Interrupt Flag Register (TIFR).

Timer/Counter0 este un numărător pe 8 biţi. Registrul de control al acestuia este Timer/Counter0 Control Register (TCCR0) din care se setează valoarea prescalerului, ceasul intern sau extern şi starea pornit-oprit a modulului. La trecerea din 0xFF în 0x00 flagul overflow este setat şi este generată întreruperea TMR0_OVR dacă aceasta este validată.

Ceasul extern este sincronizat cu oscilatorul microcontrolerului. Pentru a asigura o numărare corectă a impulsurilor externe, durata dintre două tranziţii ale clock-ului extern trebuie să fie mai mare ca perioada oscilatorului microcontrolerului.

Timer/Counter1 este un numărător pe 16 biţi. Registrele de control ale acestuia sunt Timer/Counter1 Control Registers (TCCR1A and TCCR1B). Diferitele flaguri de stare (overflow, captura unui eveniment, etc) se află în registrul Timer/Counter Interrupt Flag Register (TIFR). De asemenea ceasul extern trebuie să îndeplinească condiţia ca perioada dintre două tranziţii să fie mai mare decât perioada oscilatorului microcontrolerului pentru a asigura o funcţionare corectă a modulului.

Modulul Timer/Counter1 suportă funcţia de comparare folosind registrele Output Compare Register 1 A şi B (OCR1A şi OCR1B) pe care le compară cu registrul TCNT1. Funcţiile de comparare includ resetarea numărătorului la egalitatea cu registrul OCR1A sau alte acţiuni pe pinii de ieşire la egalitatea cu ambele registre A şi B. Modulul poate fi utilizat şi ca modulator de impulsuri în durată pe 8, 9 sau 10 biţi. De asemnea funcţia Input Capture poate salva conţinutul TCNT1 în registrul Input Capture Register (ICR1) la apariţia unui eveniment extern pe pinul de captură ICP. Setările evenimentului de captură sunt definite de registrul Timer/Counter1 Control Registers B (TCCR1B). În plus modulul Analog Comparator poate genera evenimentul de captură.

2. Watchdog Timer

Timerul Watchdog are un clock separat intern la o frecvenţă de 1 MHz, valoare tipică la o tensiune de alimentare de 5V. Din setarea valorii prescalerului intrevalul de reset poate fi ajustat la una dintre cele opt valori posibile. Instrucţiunea ***wdr*** resetează numărătorul. Dacă în perioada setată nu se foloseşte instrucţiunea ***wdr*** atunci microcontrolerul va fi resetat. Aceasta este o protecţie pentru prevenirea blocării programului (într-o buclă infinită ieşire de exemplu).

3. Memoria de date EEPROM

AT90S8515 conţine 512 Bytes de memorie EEPROM care sunt organizaţi ca un spaţiu separat de date în care se poate scrie sau citi câte un octet. Are o anduranţă de 100 000 de

cicluri scriere/ştergere. Timpul de scriere este în intervalul 2.5-4ms depinzând de tensiunea de alimentare. Când o locaţie EEPROM este scrisă, execuţia programului este întreruptă pentru două cicluri maşină înainte ca instrucţiunea următoare să fie executată. La citirea din EEPROM se întâmplă acelaşi lucru numai că execuţia este oprită timp de 4 cicluri.

Registrele EEARH and EEARL specifică adresa în spaţiul de 512 locaţii ale memoriei EEPROM. Registrul EEDR conţine datele care trebuiesc scrise sau citite din locaţia a cărei adresă este dată de perechea de registre EEAR.

Registrul Control Register – EECR conţine biţii de control ai modulului EEPROM. Pentru prevenirea scrierilor nedorite în locaţiile din memoria EEPROM trebuie luate anumite măsuri de protectie (de exemplu resetarea microcontrolerului pe perioadele în care tensiunea de alimentare scade sub un anumit prag).

4. Serial Peripheral Interface (SPI)

Acest modul permite un transfer de mare viteză între microcontroler şi dispozitive periferice externe sau între mai multe microcontrolere. SPI are următoarele caracteristici:

- Transfer sincron de date Full Duplex pe 3 fire;
- Operare mastaer sau slave;
- Transferul poate începe cu LSB sau MSB;
- Patru rate de transfer programabile;
- Generează întrerupere la terminarea transmisiei;
- Protecţie la coliziunea de scriere în registrul de transmisie;
- Poate scoate microcontrolerul din Idle Mode.

Înterconexiunea între două dispozitive Master-Slave prin SPI este ilustrată în figura 8:

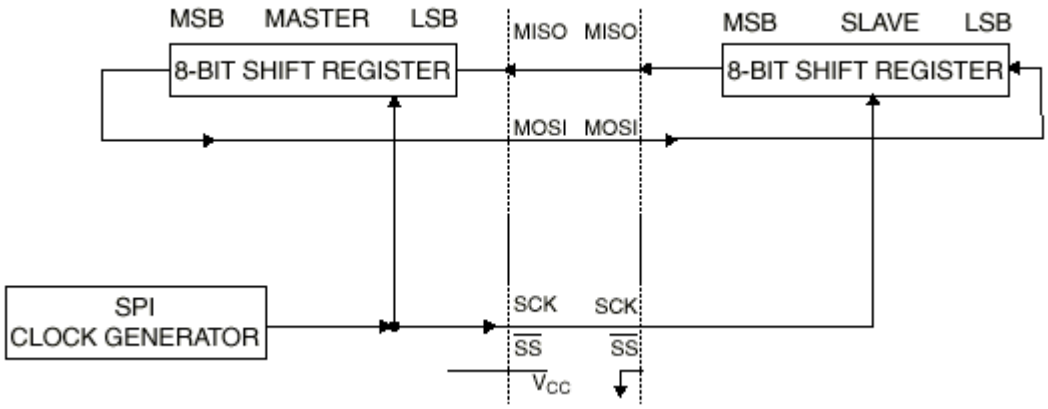


Figura 8: Conexiunea master-slave

Pinul PB7 (SCK) este ieşire în modul Master şi intrare în mod Slave. Scrierea în registrul SPI Data Register al microcontrolerului Master porneşte generatorul de tact, datele încep să fie transmise pe pinul PB5 (MOSI) şi recepţionate pe acelaşi pin PB5 al microcontrolerului Slave. Acelaşi lucru se întâmplă şi în sens invers de la slave către master pe pinii PB6 (MISO). Aceasta înseamnă că pe timpul unei perioade de Clock sunt interschimbaţi 2 biţi între Master şi Slave. Pentru comunicarea între mai multe dispozitive este prevăzut un semnal de control Slave Select PB4(SS) care trebuie ţinut în zero pentru a accesa un dispozitiv individual. După transmiterea ultimului bit SPI generează o întrerupere dacă aceasta este validată.

Registrele asociate cu modulul SPI sunt SPI Control Register – SPCR şi SPI Status Register – SPSR.

5. Universal Asynchronus Receiver Transmiter (UART)

Principalele caracteristici ale acestui modul sunt:

- Generatorul pentru stabilirea ratei de transmisie pemite a gamă largă de viteze de transmisie;
- Se pot transfera 8 sau 9 biți;
- Protecții:
  - Overrun – umplerea buferului de recepție;
  - Framing error – recepționarea unui bit de stop 0;
  - Start bit fals – eliminarea tranzițiilor false care ar putea să pornească recepția unui octet.
- Trei întreruperi asociate, separate TX Complete, TX Data Register Empty, RX Complete.

TRANSMISA

Transmisia de date este inițiată prin scrierea datelor care trebuie transmise în registrul UART I/O Data Register. Transferul din acest registru în registrul de transmisie apare când:

1. Un nou octet a fost scris în registrul UDR după ce bitul de stop al octetului anterior a fost transmis. În această situație registrul de transmisie este încărcat imediat;
2. Un nou octet a fost scris în registrul UDR înainte de transmiterea bitului de stop al octetului anterior. Registrul de transmisie este încărcat după ce a fost transmis bitul de stop al octetului curent transmis.

Dacă registrul de transmisie este gol, datele sunt transferate din UDR. În acelaşi moment flagul UDRE (UART Data Register Empty) este setat şi modulul UART este gata să primească următorul octet. În timp de un ciclu de tact se transmite bitul de start urmat apoi de LSB. Când s-a transmis ultimul bit, bitul de stop registrul de transmisie se va încărca cu date dacă acestea au fost scrise în prealabil în registrul UDR în timpul transmisei. Dacă până la transmitera bitului de stop registrul UDR nu a fost scris, flagul TX Complete (TXC) va fi setat. Bitul TXEN validează modulul de transmisie când este setat (unu) şi pinul PD1 va fi configurat automat ca ieşire, indiferent de valoarea setată în registrul DDRD (Data Direction Register D). Când bitul TXEN este resetat (zero) pinul PD1 poate fi folosit ca un pin general I/O.

RECEPȚIA

Recepția semnalului de la pinul RXD se face cu detecție majoritară la o rată de 16 ori mai mare decât viteza de recepție. În timp ce linia este în repaus (1) detecția unui nivel logic 0 va fi interpretată ca front negativ al bitului de start şi secvența de detecție a acestui bit va fi inițiată. După această tranziție negativă se sondeaza nivelul semnalului de intrare la eşantioanele 8,9,10. Dacă două sau mai multe dintre aceste eşantioane vor fi detectate ca 1,

bitul de start nu va fi considerat valid (va fi considerat o tranziție falsă) şi receptorul va căuta o nouă traziție negativă pentru a detecta un bit de start real. Dacă se detectează o tranziție validă se va face detecția majoritară pentru fiecare bit recepționat. Aceasta înseamnă că valoarea găsită la două sau mai multe dintre eşantioanele 8, 9 sau 10 va fi luată ca valoarea detectată pentru bitul respectiv. Toți biții astfel detectați vor fi introduşi în registrul de recepție. Aceste lucruri sunt ilustrate în figura 9.

Figura 9: Detectarea semanalului recepționat

Analog, la detectarea bitului de stop majoritatea eşantioanelor trebuie să fie 1 pentru ca acest bit să fie validat. Dacă nu se întâmplă acest lucru va fi o eroare semnalată prin setarea flagului Framing Error (FE) din registrul UART Status Register (USR). Înainte de citirea registrului UDR trebuie verificat bitul FE pentru detecția erorilor.

Indiferent dacă s-a detectat un bit de stop valid sau nu, la sfârşitul recepției unui octet datele sunt transferate în UDR şi bitul RX Complete (RXC) din registrul USR va fi setat. UDR reprezintă de fapt două registre, unul pentru recepție şi unul pentru transmisie. Când UDR este citit atunci este accesat registrul de recepție, iar când se scrie în UDR este accesat registrul de transmisie.

Dacă la terminarea recepției unui octet, registrul UDR nu a fost în prealabil citit flagul OverRun (OR) este setat, semnalizând că ultimul octet recepționat nu a putut fi transferat în UDR şi a fost pierdut. Bitul OR este reactualizat după citirea din registrul UDR şi trebuie verificat pentru a detecta erorile ce pot apărea la necitirea la timp a registrului UDR în cazurile în care volumul de prelucrări ale microcontrolerului este mare sau când viteza de recepție a datelor este mare.

Când bitul RXEN (RX Enable) din registrul UCR este resetat (0) modulul de recepție este invalidat. Aceasta înseamnă ca pinul PD0 poate fi folosit ca un pin general I/O. Când RXEN este setat (1) pinul PD0 va fi configurat automat ca intrare, indiferent de valoarea setată în registrul DDRD (Data Direction Register D). Când bitul CHR9 din registrul UCR este setat se foloseşte un format de 9 biți plus un bit de start şi unul de stop. Cel de-al nouălea bit ce trebuie transmis trebuie setat la valoarea dorită înainte de inițierea transmisiei prin scrierea în registrul UDR. Al nouălea bit recepționat se află în bitul RXB8 din registrul UCR.

Registrele asociate modulului UART sunt UART I/O Data Register – UDR şi UART Control Register – UCR.

Generatorul de tact pentru stabilirea vitezei de transfer este un divizor de frecvență care generează un semnal de tact cu frecvența dedusă din următoarea formulă:

$$Baud=f_{CK}/16(UBRR+1)$$

unde:

- Baud – este viteza de transfer;
- f_{CK} – este frecvența oscilatorului microcontrolerului;
- UBRR – este conținutul registrului UART Baud Rate Register (0-255).

Se pot obține o gamă largă de viteze de transfer plecând de la o frecvență dată f_{CK}.

6. Comparatorul Analogic (Analog Comparator)- AC

Modulul AC compară nivelele de tensiune de pe intrarea pozitivă PB2 (AIN0) şi intrarea negativă pinul PB3 (AIN1). Când nivelul tensiunii de la AIN0 este mai mare decât AIN1 bitul Analog Comparator Output (ACO) este setat (1). ACO poate să declanşeze funcția Timer/Counter1 Input Capture. În plus comparatorul poate declanşa o întrerupere separată, asociată exclusiv cu acest modul. Registrul de control al modulului comparator este Timer/Counter1 Input Capture.

7. Interfaţa cu memoria externă

Interfața cu memoria SRAM constă în:

- Port A – reprezintă bus-ul de date multiplexat cu bus-ul de adrese de nivel inferior;
- Port C – reprezintă bus-ul de adrese de nivel superior;
- Pinul ALE – Address Latch Enable;
- Pinul RD/WR – reprezintă strobe RD/WR.

Memoria externă este validată prin setarea bitului SRE din registrul MCUCR (MCU Control Register), acest lucru implicând suprascrierea setărilor din registru de direcție DDRA. Când bitul SRE este resetat (0) memoria externă nu este folosită, Portul A este folosit în concordanță cu setările din registrul DDRA şi spațiul de adrese peste spațiul alocat memoriei interne SRAM nu poate fi folosit.

Pe frontul negativ al pinului ALE pe Portul A este disponibilă o adresă validă. În timpul transferului de date ALE este 0. RD şi WR sunt active doar când se accesează memoria SRAM externă. În figura 10 se arată cum se poate conecta o memorie SRAM externă la microcontroler folosind 8 latch-uri care sunt transparente când G este în 1. În mod normal accesul SRAM extern se face într-o schemă de trei cicluri după cum se poate vedea în figura 10. Când este nevoie de un ciclu în plus pentru acces se setează bitul SRW din registrul MCUCR. Noua schemă este prezentată în figura 11.

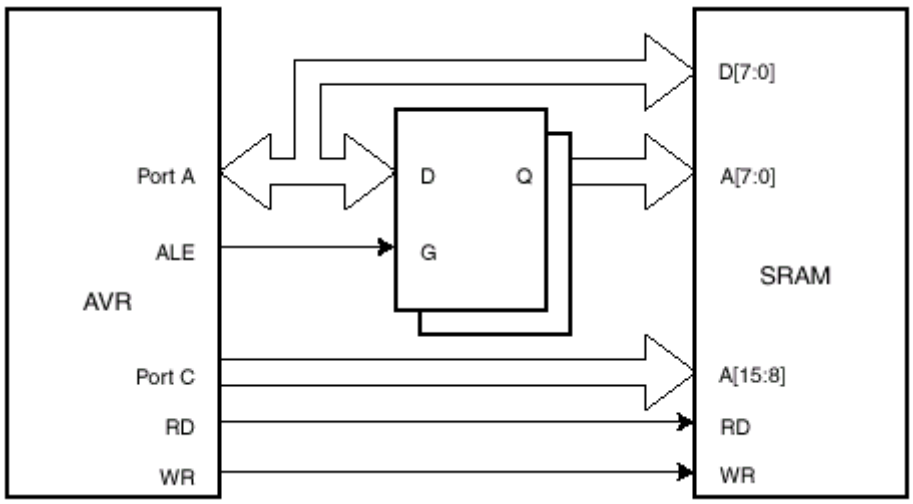


Figura 9: Conectarea unei memorii externe SRAM

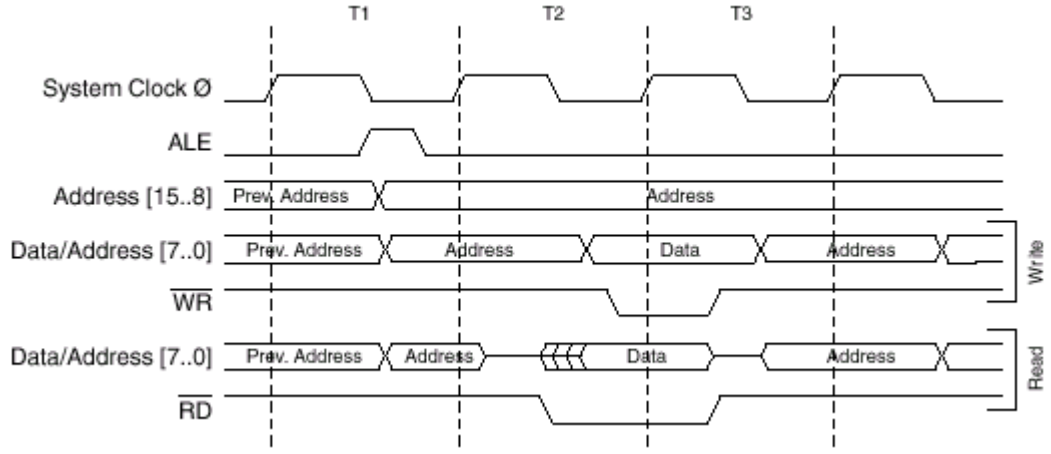


Figura 10: Accesarea memoriei externe SRAM fără ciclu de aşteptare

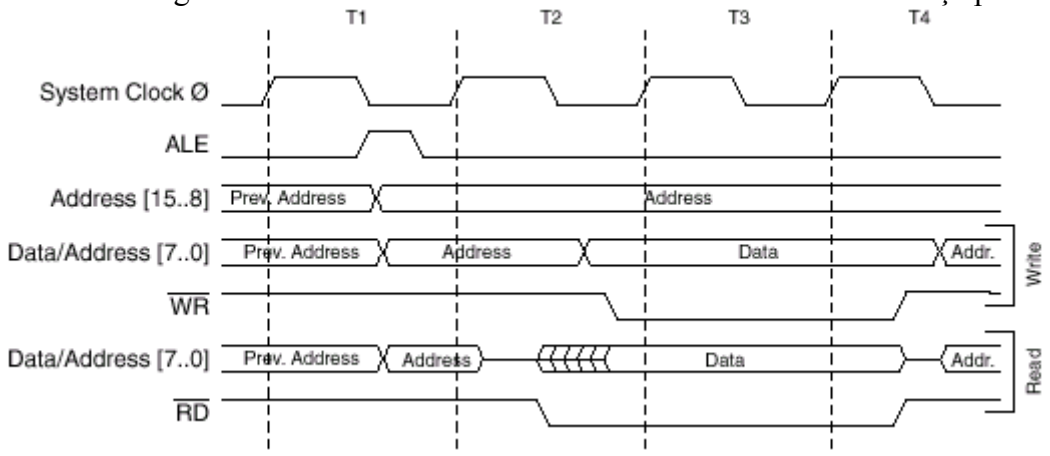


Figura 11: Accesarea memoriei externe SRAM cu ciclu de aşteptare

II. Descrierea hardware-lui de dezvoltare STK500

STK500 este un sistem complet de dezvoltare a aplicațiilor cu microcontrolere AVR Flash de la compania ATMEL. Este conceput pentru a furniza utilizatorului un mod rapid de a începe să dezvolte programe pentru microcontrolerele AVR şi un mod uşor de a realiza şi testa prototipurile noilor aplicații. Caracteristicile principale ale acestei platforme de dezvoltare sunt:

- este compatibil cu programul AVR Studio furnizat de aceeaşi firmă, care este un mediu de dezvoltare şi simulare a programelor pentru AVR;
- are interfață serială pentru controlul şi programarea kitului cu ajutorul PC;
- are socluri pentru toate dispozitivele AVR (8, 20, 28, 40 pini);
- programare serială şi paralelă;
- Programare în sistem pentru dispozitivele AVR;
- 8 butoane (taste) pentru uz general;
- 8 LED-uri pentru uz general;
- toate porturile I/O sunt accesibile prin conectori externi;
- are un port serial RS232 adițional pentru uz general;
- conține pe placă o memorie Flash de 2 MBiți.

Descrierea din punct de vedere hardware este detaliată în continuare:

1. LED-urile

STK500 include 8 LED-uri galbene şi opt taste push-button de uz general separate de restul plăcii prin conectori.

Conectorii pot fi legați prin cablurile panglică cu 10 fire la porturile I/O ale microcontrolerului. În figura 12 se poate vedea circuitul de control al LED-urilor. Microcontrolerul AVR poate furniza un curent suficient pentru LED, dar a fost aleasă această soluție pentru furnizarea unui curent constant prin LED în toată gama de tensiuni de la 1.8V la 6V.

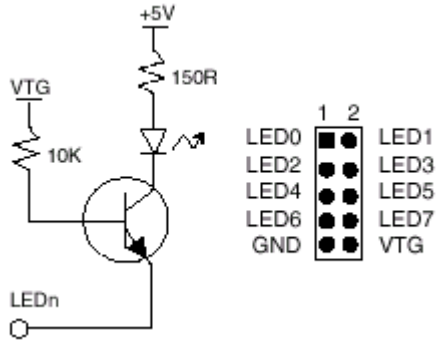


Figura 12: Circuitul de control al LED-urilor şi conectorul acestora

Dacă tensiunea de VTG (V Target) lipseşte, ledul este stins.

2. TASTELE

De asemenea tastele sunt separate de restul plăcii, dar pot fi conectate prin conectorii corespunzători. În figura 13 se poate vedea circuitul corespunzător unei taste.

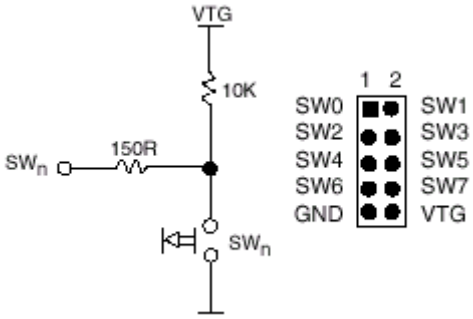


Figura 13: Circuitul tastelor şi conectorul acestora

Apăsarea unei taste implică trecerea semnalului SWx în zero, iar eliberarea ei implică trecerea SWx în VTG.

Se pot folosi rezistoarele interne al microcontrolerului nemaifiind nevoie de rezistoare exeterne. După cum se poate vedea din ultimile două figuri, conectorii furnizează şi tensiunea VTG şi potențialul de masă GND pe lângă semnalele corespunzătoare tastelor.

3. CONECTORII PORTURILOR I/O

Pentru un port în general dispunerea pinilor este prezentată în figura 14:

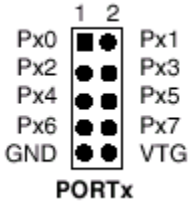


Figura 14: Conectorul unui port I/O

Conectorul portului E are unele semnale în plus față de Portul E după cum se poate vedea în figura 15 :

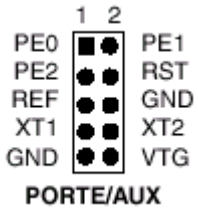


Figura 15: Conectorul portului E

Notațiile utilizate sînt:

- REF este tensiunea de referintă analogică. Acest semnal este legat la pinul AREF la dispozitivele care au referința analogică separată pe un pin;
- XT1 - pinul XTAL1 este semnalul de clock pentru toate soclurile şi poate fi folosit cu un semnal de ceas extern;
- XT2 - pinul XTAL2 se foloseşte pentru conectarea cristalului extern cu cuartz împreună cu pinul XTAL1.

4. Interfaţa serială RS232

STK500 include două porturi seriale RS232. unul este folosit pentru comunicarea cu AVR Studio. Celălalt poate fi folosit pentru comunicarea dintre microcontrolerul de pe placă şi un port serial PC. Pentru a se putea întâmpla acest lucru, pinii asociați modulului UART al microcontrolerului trebuie să fie conectați la acel port prinntr-un convertor după cum se poate vedea în figura 16.

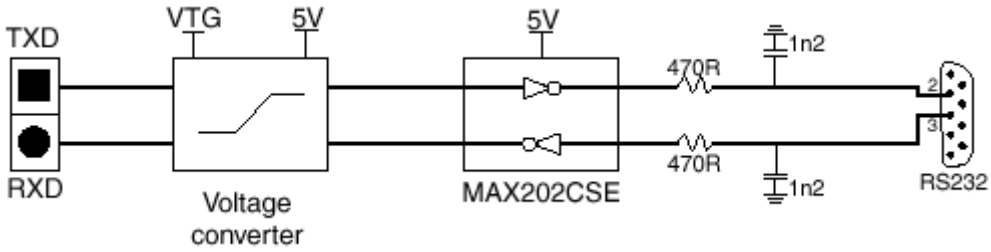


Figura 16: Conectarea portului RS232 la modulul UART

5. Descrierea memoriei Flash

O memorie de 2 Mbiți AT45D021 este inclusă în STK500 pentru stocarea anumitor date. Aceasta este o memorie Flash de mare densitate, cu interfață serială SPI care se poate lega la pinii asociați modulului SPI al microcontrolerului prin conectorul cu 4 pini DATAFLASH, folosindu-se cabluri cu două fire incluse în STK500. Conectarea se face ca în figura 17:

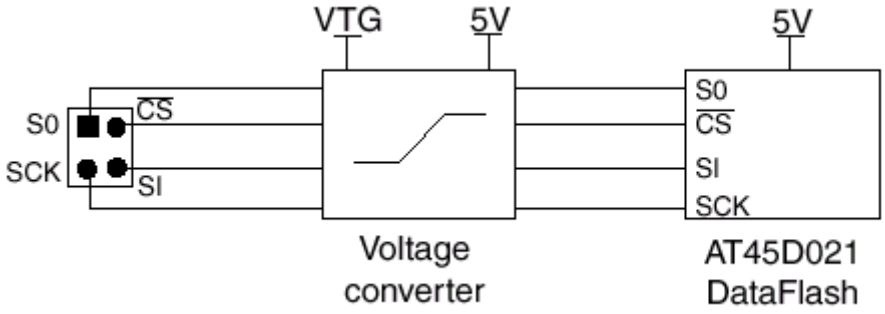


Figura 17: Conectarea memoriei Flash la microcontroler

6. Selecţia soclului

Modulul de programare constă în 8 socluri aflate în centrul plăcii STK500 în care se pot introduce microcontrolerele AVR pentru a fi programate sau pentru a fi folosite în aplicații. Numai un singur microcontroler se poate afla la un moment dat pe placă. Memoria Flash AVR poate suporta garantat 1000 de programări. Odată introdus microcontrolerul în soclu acesta poate fi programat din AVR Studio în două moduri:

1. AVR In-System Programing ce funcționează la tensiunea normală de alimentare a microcontrolerului şi foloseşte modulul SPI pentru a transfera codul în memoria Flash sau în cea EEPROM;
2. Programare la tensiune înaltă când o tensiune de 12 V este aplicată pinului Reset. Există două moduri de programare: serială şi paralelă.

7. Configurarea hardware a plăcii

Un microcontroler master şi 8 jumperi controlează setările plăcii.

Mărimile ce se pot configura din jumperi sunt următoarele:

V Target – reprezintă tensiunea de alimentare a microcontrolerului din soclu. Aceasta poate sa fie generată de STK500 sau poate fi preluată de la o sursă externă. Dacă jumperul VTARGET este montat (on), se conectează tensiunea de alimentare de pe placă ce poate fi controlată din AVR Studio între 0 şi 6V.

Microcontrolerul master de pe STK500 controlează această tensiune prin PWM. Dacă jumperul VTARGET este scos (off), tensiunea de alimentare a microcontrolerului trebuie furnizată din exterior pe pinul VTG al conectorilor porturilor I/O. Existența tensiunii de alimentare VTG este semnalată de LED-ul verde. Cele de mai sus sunt ilustrate în figura 18:

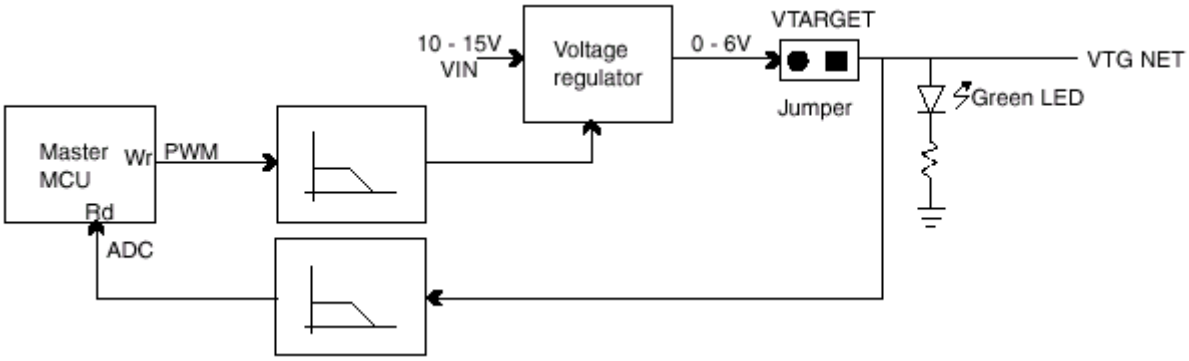


Figura 1.18: Circuitul VTG al STK500

Analog Reference Voltage, AREF – STK500 poate genera o tensiune de referință modulului A/D sau modulului ADC al microcontrolerului, dacă jumperul AREF este montat. Şi această tensiune poate fi controlată din AVR Studio în intervalul 0-6V utilizând tot PWM, dar nu poate să depăşească VTG. Când jumperul ARF este deconectat, tensiunea de referință trebuie furnizată din exterior pe conectorul portului PORTE/AUX, după cum se poate vedea în figura 19.

Tensiunea de referință generată intern este disponibilă la conectorul portului E şi poate fi utilizată în exteriorul plăcii.

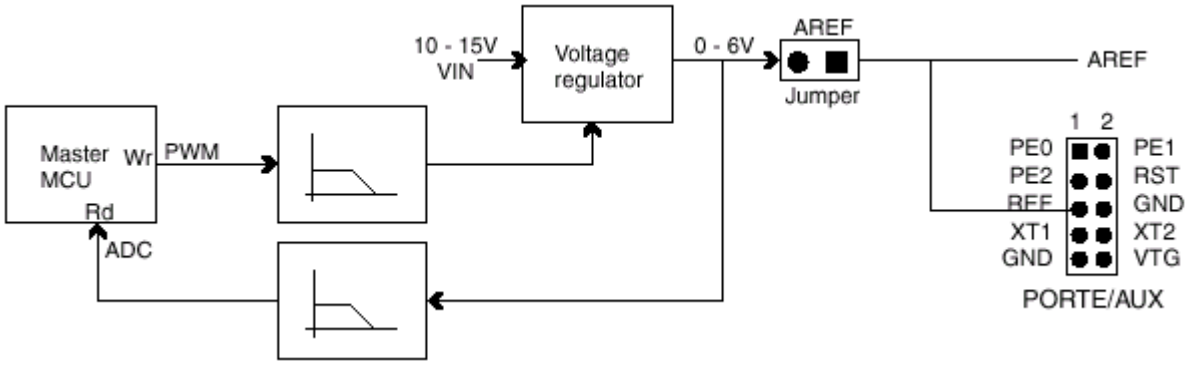


Figura 19: Circuitul ARF al STK500

Reset – jumperul controlează semnalul de reset în sistemul STK500. Când se programează în mod ISP, microcontrolerul master programează AVR fără să interfereze cu aplicația dacă jumperul RESET este montat. Dacă nu, semnalul de Reset de pe STK500 este controlat extern pe conectorul portului PortE/AUX ca în figura 20.

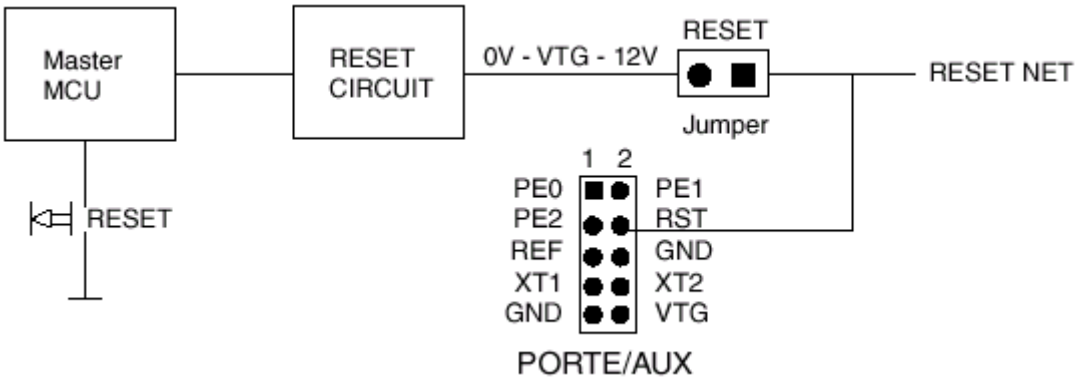


Figura 20: Circuitul de Reset la STK500

În modul de programare cu tensiune mare pe semnalul de Reset STK500 aplică o tensiune de 12V. Deci un circuit extern de Reset care nu ține cont de această tensiune trebuie deconectat în timpul programării.

Pentru programare circuitul extern trebuie să permită controlarea semnalului de Reset de către master.

Circuitul de generare a ceasului

STK500 permite câteva opțiuni pentru setarea clock-ului. Jumperii XTAL1 şi OSCSEL controlează aceste setări. Ultimul determină care semnal ajunge la pinul XTAL1 al microcontrolerului. Când jumperul XTAL1 este montat, clock-ul generat de STK500 este folosit pentru microcontroler iar când nu e montat se poate utiliza un clock extern. Aceste lucruri se pot vedea în figura 21.

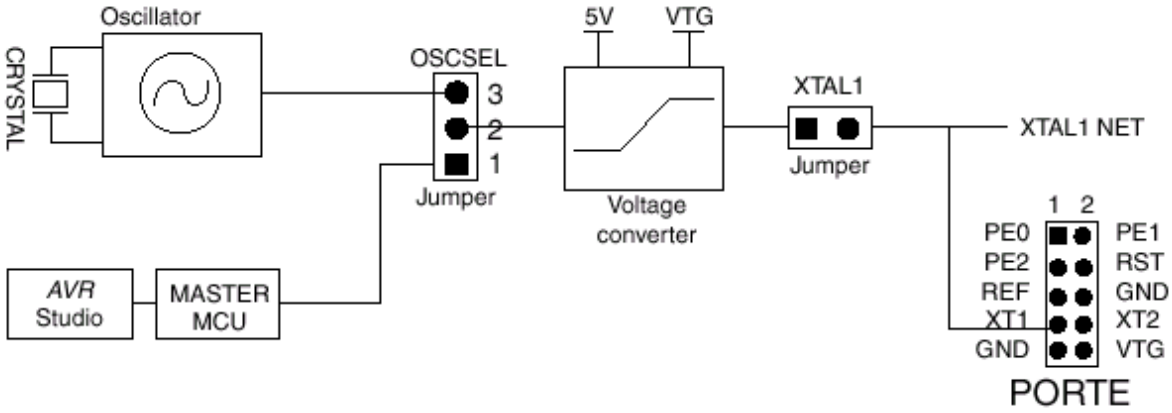


Figura 21: Circuitul de generare a ceasului la STK500

Pinul XT1 al conectorului portului E poate fi folosit pentru furnizarea clock-ului extern sau ca sursă clock pentru un circuit exetrn. Generatorul de semnal poate fi un oscilator cu cristal sau se foloseşte clock-ul generat de microcontrolerul master. Frecvența celui din urmă poate fi variată cu ajutorul AVR Studio în domeniul 0-3.68 MHz. Oscilatorul poate folosi cristale cu frecvența cuprinsă în intervalul 2-20MHz.

8. Conectorii de extensie

STK500 are 2 conectori câte unul pe fiecare parte a modulului de programare. Toate porturile I/O AVR, semnalele de programare şi de control sunt legate la aceşti conectori. Acest lucru permite realizarea uşoară a prototipurilor pentru aplicaţiile noi dezvoltate. Configuraţia pinilor pentru cei doi conectori este ilustrată în figura 22 unde semnificaţiile semnalelor sunt următoarele:

- AUXI1, AUXI0, AUXO1 şi AUXO0 sunt rezervate pentru aplicaţii viitoare;
- DATA[7:0] şi CT[7:1] sunt folosite pentru programare paralelă de tensiune înaltă;
- BSEL2 (Byte Select 2) este folosit pentru programarea ATmega161 şi 163;
    - Semnalele SI, SO, SCK şi CS sunt folosite de memoria de date Flash;
    - NC sunt neconectate;
    - Restul semnalelor reprezintă pinii porturilor I/O.

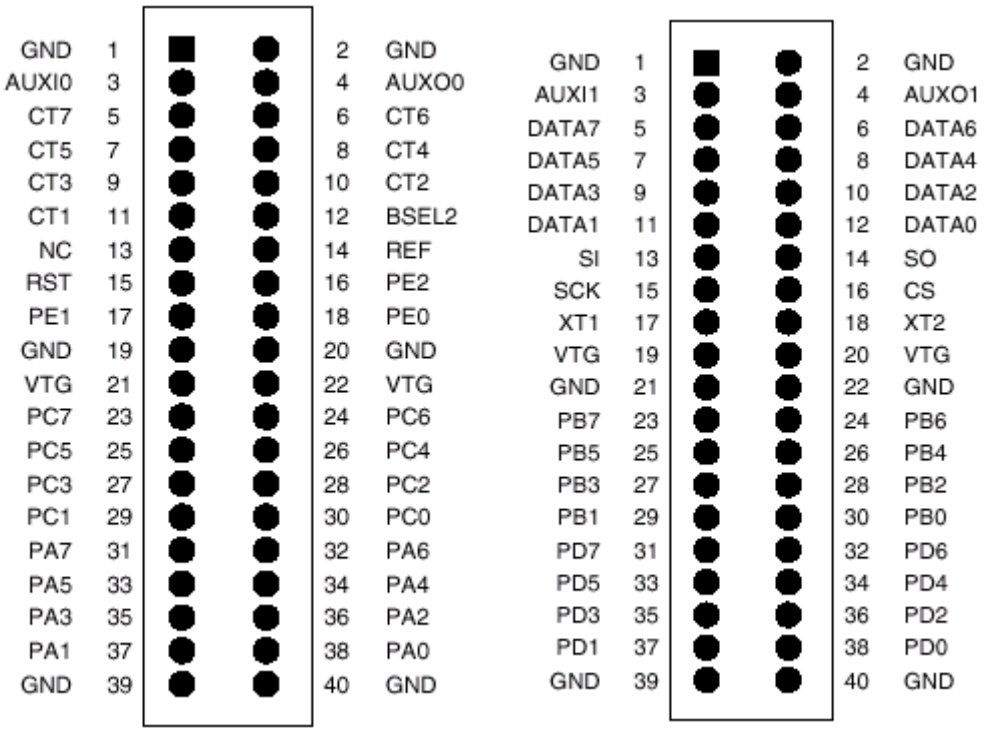


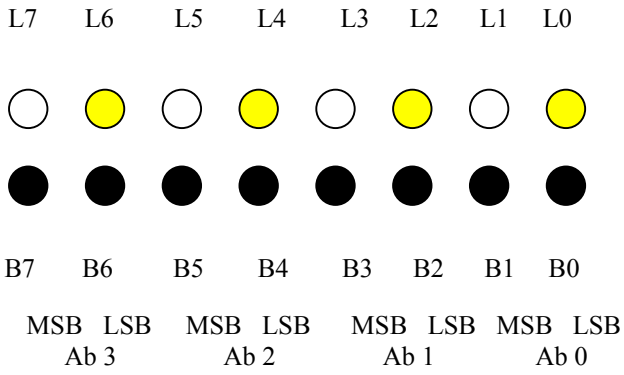
Figura 22: Conectorii de expansiune la STK500

Programele vor fi dezvoltate cu ajutorul compilatorului C – CAVR. Încarcarea codului în sistemul de dezvoltare STK 500 va fi efectuată în mod automat folosindu-se facilitatea de programare “in-sistem” a memoriei flash interne microcontrolerului.

III. Descrierea unei aplicaţii software: planificatorul de procese şi procesele secvenţiale

Aplicatia propusă simulează o centrala telefonică digitală cu 4 abonati.

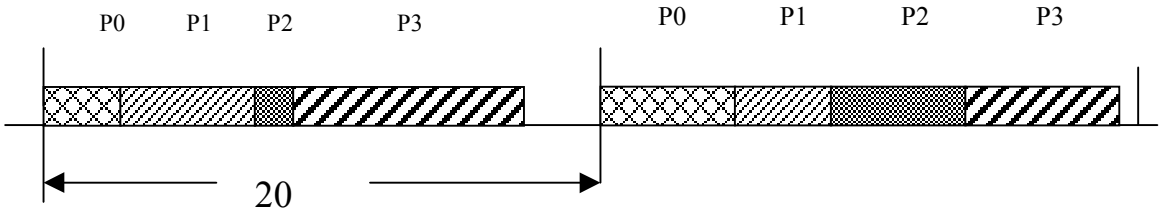
Se consideră procesele de tratare a apelurilor ca in diagrama de mai jos.
Fiecărui proces (abonat) i se aloca 2 butoane si 1 Led ca in figura următoare:



Butoanele sînt utilizate pentru a simula initierea unui apel (ridicarea receptorului) (LSB) sau pentru a forma cifrele abonatului chemat.

Led-urile (doar cele marcate) sînt folosite pentru a semnaliza starea de ocupat (pentru un abonat chemator) sau pentru sonerie (pentru un abonat chemat).

Planificarea proceselor se efectueaza cu diviziune uniforma in timp cu autosuspendare. Fiecare proces executa prelucrarile asociate starii sale curente dupa care se autosuspenda si cedeaza procesorul urmatorului proces. Durata unei cuante de timp este de 20 ms. In acest interval de timp se executa toate procesele existente.

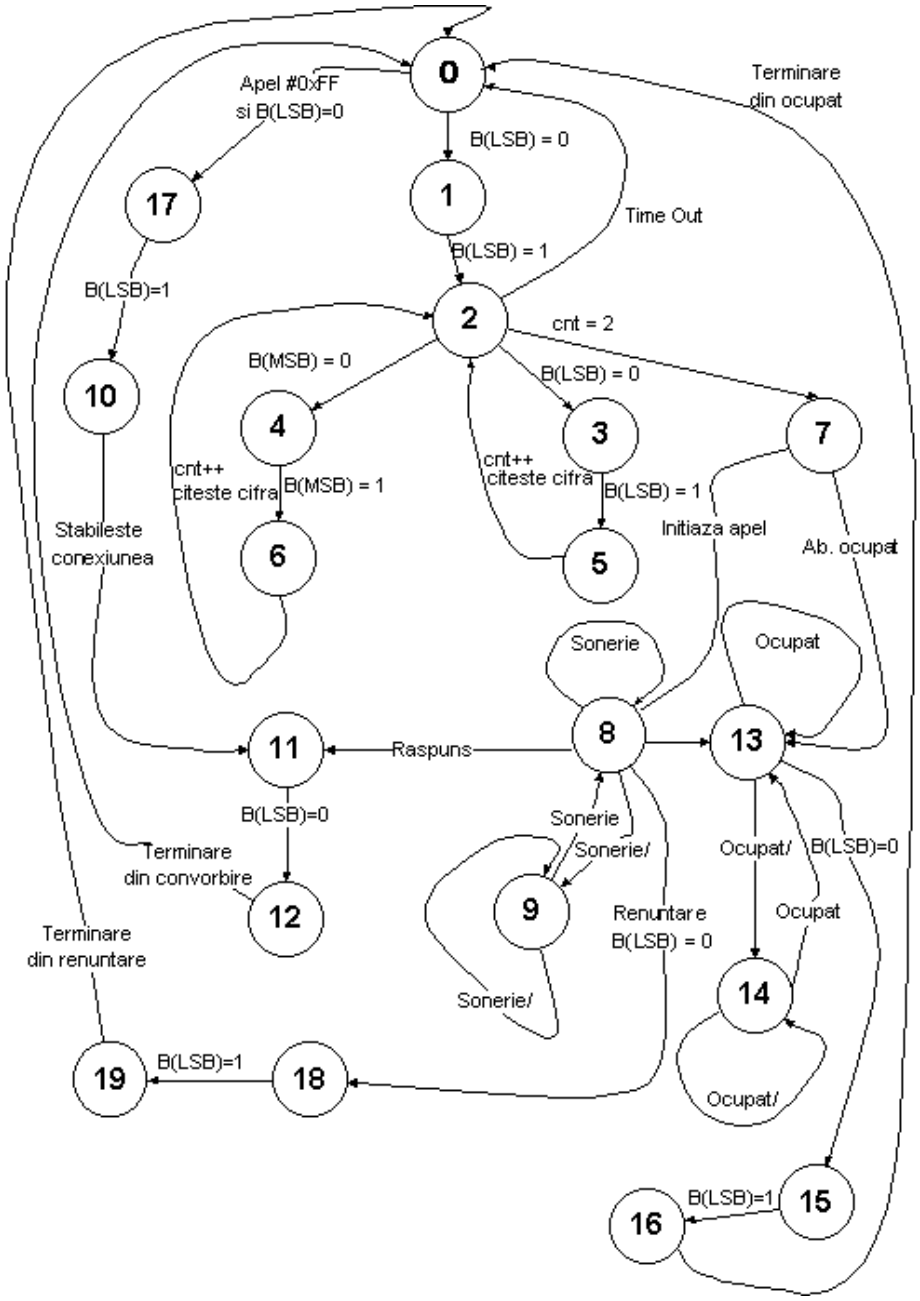


Implementarea planificatorului utilizează o tabela de stari (starile tuturor proceselor). Fiecare proces este initializat într-o stare anume (uzual în starea 0). Se defineste o tabela de pointeri la funcţie care conţine toate adresele funcţiilor care implementează prelucrarile din fiecare stare.

Un ciclu de tip *for* dupa numarul de procese va executa prelucrarea curentă a fiecărui proces.

De asemenea s-au definit timeri pentru fiecare proces pentru a se iesi fortat la expirarea unui interval de timp (daca evenimentul de intrare asteptat nu a aparut).

Listingul progarmului este ilustrat în continuare.



Notații :

B(LSB) – butonul mai putin semnificativ, B(MSB) – butonul mai semnificativ
Initiaza apel – apel[ab_chemat]=nr_ab_chemator+10
Abonat ocupat – apel==0xFF
Raspuns – apel[crt_pid]#0xFF si apel[srt_pid]#20
Terminare – apel[crt_pid]=0xFF
Sonerie – Led aprins/stins pentru abonatul chemat
Ocupat - Led aprins/stins pentru abonatul chemator

Programul principal (test_plan.c)

```
/*****************************************
This program was produced by the
CodeWizardAVR V1.23.6a Standard
Automatic Program Generator
© Copyright 1998-2002 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro , hpinfotech@xnet.ro

Project :
Version :
Date    : 9/19/2002
Author  : Sorin Zoican
Company : Fac Electronica Telecomunicatii
Comments: Test palnificator de proces e- autosuspendare


Chip type           : AT90S8515
Clock frequency     : 4.000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*****************************************/
#include <90s8515.h>

#include "defs.h"
#include "st.h"
#include "fc.h"

// Declare your global variables here

unsigned char crt_pid;//procesul curent
unsigned int timer[NRPROC];//tablou de timeri asociati proceselor
unsigned char CRT_ST[NRPROC];//starea curenta a proceselor

char contor[NRPROC];
char contor1[NRPROC];
char cnt[NRPROC];
char cifra[NRPROC][2];
char nr[NRPROC];
char apel[NRPROC];


//pointeri la functiile de stare ale proceselor
void (*p_func[])(void) = {
                  st_0,//0x00
                  st_1,//0x01
                  st_2,//0x02
                  st_3,//0x03
                  st_4,//0x04
                  st_5, //0x05
                  st_6,//0x06
                  st_7,//0x07
                  st_8,//0x08
                  st_9,//0x09
                  st_10,//0x0a
                  st_11,//0x0b
```

```
                    st_12,//0x0c
                    st_13,//0x0d
                    st_14,//0x0e
                    st_15,//0x0f
                    st_16,//0x10
                    st_17,//0x11
                    st_18,//0x12
                    st_19 //0x13
                    };


void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out
Func7=Out
// State0=0 State1=0 State2=0 State3=0 State4=0 State5=0 State6=0 State7=0
PORTB=0x00;
DDRB=0xFF;

// Port C initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 3.906 kHz
TCCR0=0x05;
TCNT0=0x4E;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
```

```
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;

// Global enable interrupts
#asm("sei")

init_plan();
PORTB=0xff;

while (1)
      {
      // Place your code here
      };
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x4E;

// Place your code here

for(crt_pid=0; crt_pid < NRPROC; crt_pid++)
{
// decrementeaza timerii
if(timer[crt_pid]) timer[crt_pid]--;
//executa procesul crt_pid
(*p_func[CRT_ST[crt_pid]])();
}

}
```

Implementarea starilor proceselor (st.c)

```
#include "defs.h";

void st_0 (void)
{
//cerere abonat Bi = 0
```

```
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01;
if (tmp==0) Install(ST_1);
// abonat chemat
if (apel[crt_pid]!=0xff & tmp==0) Install(ST_17);
}

void st_1 (void)
{
//cerere abonat Bi = 1
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01;
if (tmp==1)       {
            tmp=~(1<<(2*crt_pid));
            PORTB=PORTB&tmp; // Li - 0
            SetTimer(WAIT_CIFRA);
// semnalizare apel in desfasurare
            apel[crt_pid]=20;
            Install(ST_2);
            }
}

void st_2 (void)
{
//asteapta formarea cifrelor numarului abonatului chemat
char tmp;
// numar complet format
if (TimeOut())
            {
            tmp=(1<<(2*crt_pid));
            PORTB=PORTB|tmp; // Li - 1
            Install(ST_0);
                }
if (cnt[crt_pid]==2)    {
            cnt[crt_pid]=0;
            Install(ST_7);
                  }
tmp = (PIND>>(2*crt_pid))&0x3;
// apasat Bi (LSB)
if (tmp==2) Install(ST_3);
// apasat B2i (MSB)
if (tmp==1) Install(ST_4);
}

void st_3 (void)
{
//asteapta Bi = 1
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01;
if (tmp==1) Install(ST_5);
}

void st_4 (void)
{
//asteapta B2i = 1
char tmp;
tmp = (PIND>>(2*crt_pid))&0x02;
if (tmp==2) Install(ST_6);
}
```

```
void st_5 (void)
{
// Bi - puls 0->1
char tmp;
tmp=cnt[crt_pid];
cifra[crt_pid][tmp]=0;
cnt[crt_pid]++;
Install(ST_2);
}

void st_6 (void)
{
// B2i - puls 0->1
char tmp;
tmp=cnt[crt_pid];
cifra[crt_pid][tmp]=1;
cnt[crt_pid]++;
Install(ST_2);
}

void st_7 (void)
{
// calculeza numarul abonatului chemat
// cerere catre abonatul chemat
nr[crt_pid]=2*cifra[crt_pid][0]+cifra[crt_pid][1];
if ( apel[nr[crt_pid]]==0xff )// abonat liber
      {
      apel[nr[crt_pid]]=crt_pid+10;
      Install(ST_8);
      }
else                              // abonat ocupat
      {
      Install(ST_13);
      }
if (nr[crt_pid]==crt_pid) Install(ST_13); // abonat chemat identic cu
abonat cemator
}

void st_8 (void)
{
//apel sonerie la abonatul chemat
char tmp;
// renuntare la apel
tmp = (PIND>>(2*crt_pid))&0x01;
if (tmp==0) Install(ST_18);
//abonatul chemat a raspuns
// si abonatul chemat nu are un apel in desfasurare
if ((apel[crt_pid]!=0xff)&(apel[crt_pid]!=20))
 Install(ST_11);

tmp=~(1<<(2*nr[crt_pid]));
PORTB=PORTB&tmp; // Lnr = 0
contor[crt_pid]++;
if (contor[crt_pid]==N) {
                  contor[crt_pid]=0;
                  Install(ST_9);
                  }
}
```

```
void st_9 (void)
{
//apel sonerie la abonatul chemat
char tmp;
tmp=(1<<(2*nr[crt_pid]));
PORTB=PORTB|tmp;   // Lnr = 1
contor[crt_pid]++;
if (contor[crt_pid]==N) {
                  contor[crt_pid]=0;
                  Install(ST_8);
                  }
}

void st_10 (void)
{
//raspuns la apel
char tmp;
tmp=~(1<<(2*crt_pid));
PORTB=PORTB&tmp;  // Li = 0
tmp = apel[crt_pid]-10;
apel[tmp]=crt_pid;
Install(ST_11);
}

void st_11 (void)
{
//convorbire
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01;
if (tmp==0) Install(ST_12); // Bi = 0
}

void st_12 (void)
{
//incheiere convorbire de catre abonatul chemator ( apel <9)
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01;
if ((tmp==1)&&(apel[crt_pid]==0xff)) // abonatul chemat
      {
      tmp=(1<<(2*crt_pid));
      PORTB=PORTB|tmp; // Li = 1
      Install(ST_0);
      }
if ((tmp==1)&&(apel[crt_pid]<9)) // abonatul chemator
      {
      tmp=apel[crt_pid];     // inchide comunicatia
      apel[crt_pid]=-1;
      apel[tmp]=-1;
      tmp=(1<<(2*crt_pid));
      PORTB=PORTB|tmp; // Li = 1
      Install(ST_0);
      }
}

void st_13 (void)
{
//ocupat
char tmp;
```

```
tmp=~(1<<(2*crt_pid));
PORTB=PORTB&tmp; // Li = 0
contor1[crt_pid]++;
if (contor1[crt_pid]==N) {
                  contor1[crt_pid]=0;
                  Install(ST_14);
                  }
tmp = (PIND>>(2*crt_pid))&0x01;  //Bi = 0
if (tmp==0) Install(ST_15);
}

void st_14 (void)
{
//ocupat
char tmp;
tmp=(1<<(2*crt_pid));
PORTB=PORTB|tmp;  //Li = 1
contor1[crt_pid]++;
if (contor1[crt_pid]==N) {
                  contor1[crt_pid]=0;
                  Install(ST_13);
                  }
}

void st_15 (void)
{
// incheiere daca a fost ocupat
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01; //Bi = 1
if (tmp==1) Install(ST_16);
}

void st_16 (void)
{
// incheiere daca a fost ocupat
char tmp;
tmp=(1<<(2*crt_pid));
PORTB=PORTB|tmp;  //Li = 1
Install(ST_0);
}

void st_17 (void)
{
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01; // Bi = 1
if (tmp==1) Install(ST_10);
}

void st_18 (void)
{
char tmp;
tmp = (PIND>>(2*crt_pid))&0x01; // Bi = 1
if (tmp==1) Install(ST_19);
}

void st_19 (void)
{
char tmp;
apel[nr[crt_pid]]=0xff;
```

```
apel[crt_pid]=0xff;
tmp=(1<<(2*crt_pid));
PORTB=PORTB|tmp; // Li = 1
tmp=(1<<(2*nr[crt_pid]));
PORTB=PORTB|tmp; // Lnr = 1
Install(ST_0);
}
```

Functiile utilizate in program (fc.c)

```
#include "defs.h"
#include "st.h"
#include "fc.h"

extern unsigned char crt_pid;
extern unsigned char CRT_ST[];
extern unsigned int timer[];

//locatia de la adresa adresa+crt_pid ia valoarea val
void PutXVal(unsigned char *adresa, unsigned char val)
{
      p_write(adresa+crt_pid,val);
}

//intoarce valoarea locatiei de la adresa adresa+crt_pid
/*
unsigned char GetXVal(unsigned char *adresa)
{
      return(p_read(adresa+crt_pid));
}
*/

//scrie in portul de adresa padd valoarea pdat
void p_write(unsigned char *padd, unsigned char pdat)
{
        *padd=pdat;
}

//citeste portul de adresa padd
/*
unsigned char p_read(unsigned char *padd)
{
        return(*padd);
}
*/
// initializeaza starea initiala a proceselor
void init_plan()
{
      for(crt_pid=0; crt_pid<NRPROC; crt_pid++)
      {
      apel[crt_pid]=0xff;
// apel[crt_pid]=0xff - abonat liber
// apel[crt_pid]=20 - apel in desfasurare
// apel[crt_pid]= 1x pt. chemat ( x este abonatul chemator)
// apel[crt_pid]= y pt. chemator ( y este abonatul chemat)
      cnt[crt_pid]=0;
      contor[crt_pid]=0;
      contor1[crt_pid]=0;
```

```
      SetTimer(0);
      cifra[crt_pid][0]=0;
      cifra[crt_pid][1]=0;
      Install(ST_0);
      }
}

// instalare in alta stare a procesului crt_pid
void Install(unsigned char new_st)
{
      PutXVal(CRT_ST, new_st);
}


//incarca valoarea new_time in timerul procesului curent
void SetTimer(unsigned int new_time)
{
      timer[crt_pid]=new_time;
}

// intoarce 1 daca timerul procesului curenta ajuns la 0
char TimeOut()
{
      if(!(timer[crt_pid])) return(1);
      else return(0);
}
```

Fisiere header

St.h

```
void st_0(void);
void st_1(void);
void st_2(void);
void st_3(void);
void st_4(void);
void st_5(void);
void st_6(void);
void st_7(void);
void st_8(void);
void st_9(void);
void st_10(void);
void st_11(void);
void st_12(void);
void st_13(void);
void st_14(void);
void st_15(void);
void st_16(void);
void st_17(void);
void st_18(void);
void st_19(void);
```

Fc.h

```
void init_plan(void);
void Install(unsigned char new_st);
```

```
void SetTimer(unsigned int new_time);
char TimeOut(void);
void p_write(unsigned char *padd, unsigned char pdat);
//unsigned char p_read(unsigned char *padd);
```

Defs.h

```
#define NRPROC 0x04
#define N   10
#define WAIT_CIFRA      500

// stari
#define ST_0 0x00
#define ST_1 0x01
#define ST_2 0x02
#define ST_3 0x03
#define ST_4 0x04
#define ST_5 0x05
#define ST_6 0x06
#define ST_7 0x07
#define ST_8 0x08
#define ST_9 0x09
#define ST_10 0x0a
#define ST_11 0x0b
#define ST_12 0x0c
#define ST_13 0x0d
#define ST_14 0x0e
#define ST_15 0x0f
#define ST_16 0x10
#define ST_17 0x11
#define ST_18 0x12
#define ST_19 0x13
```