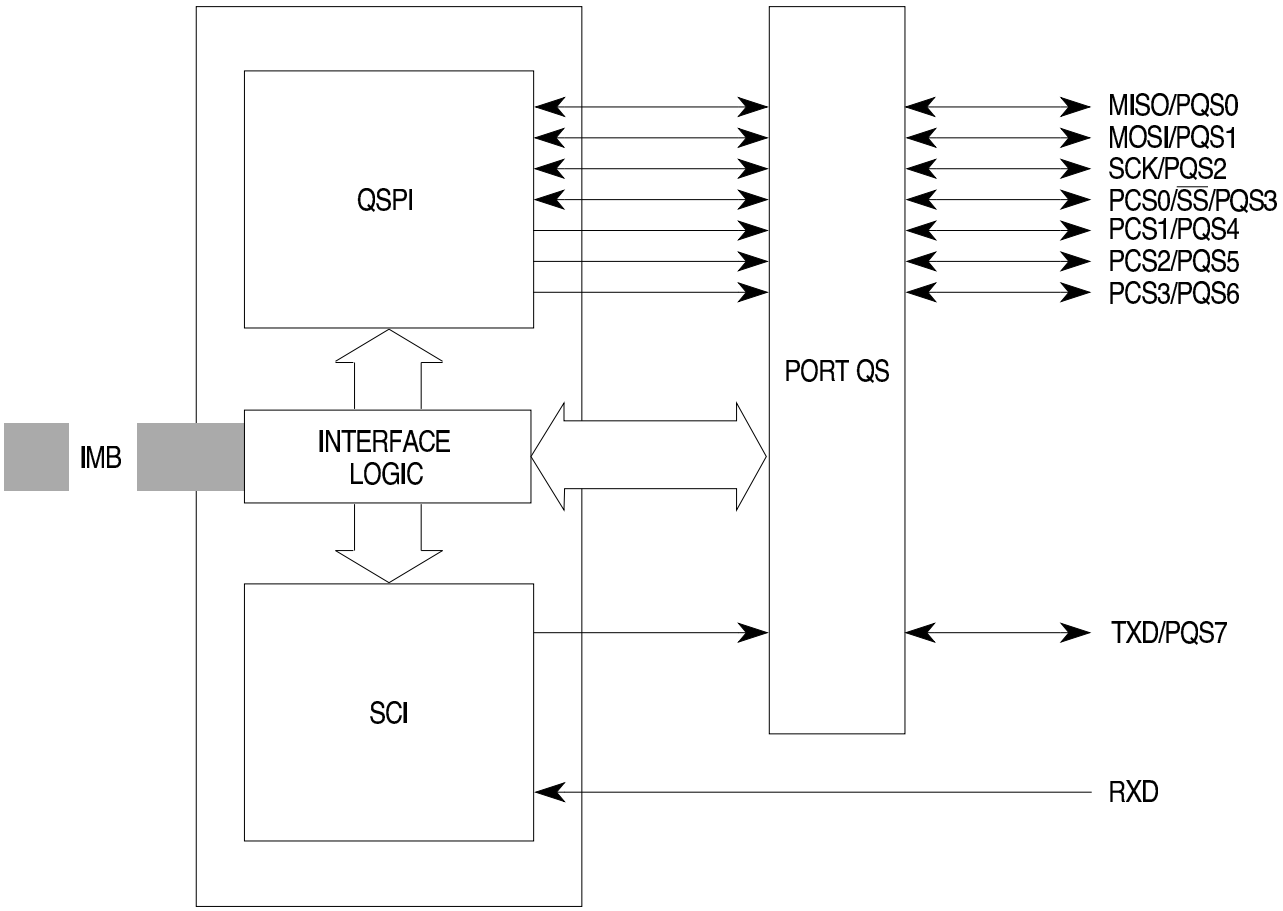
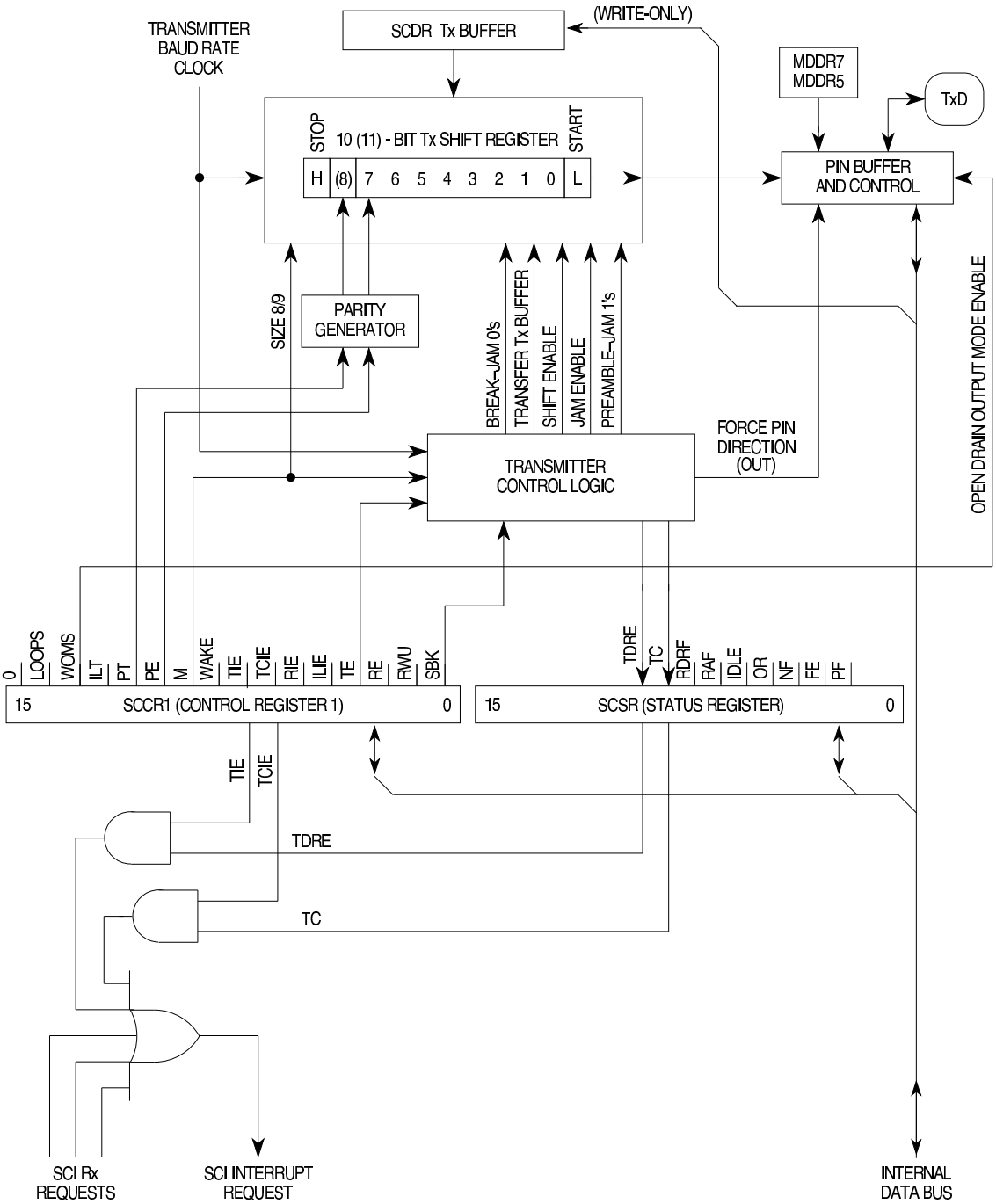
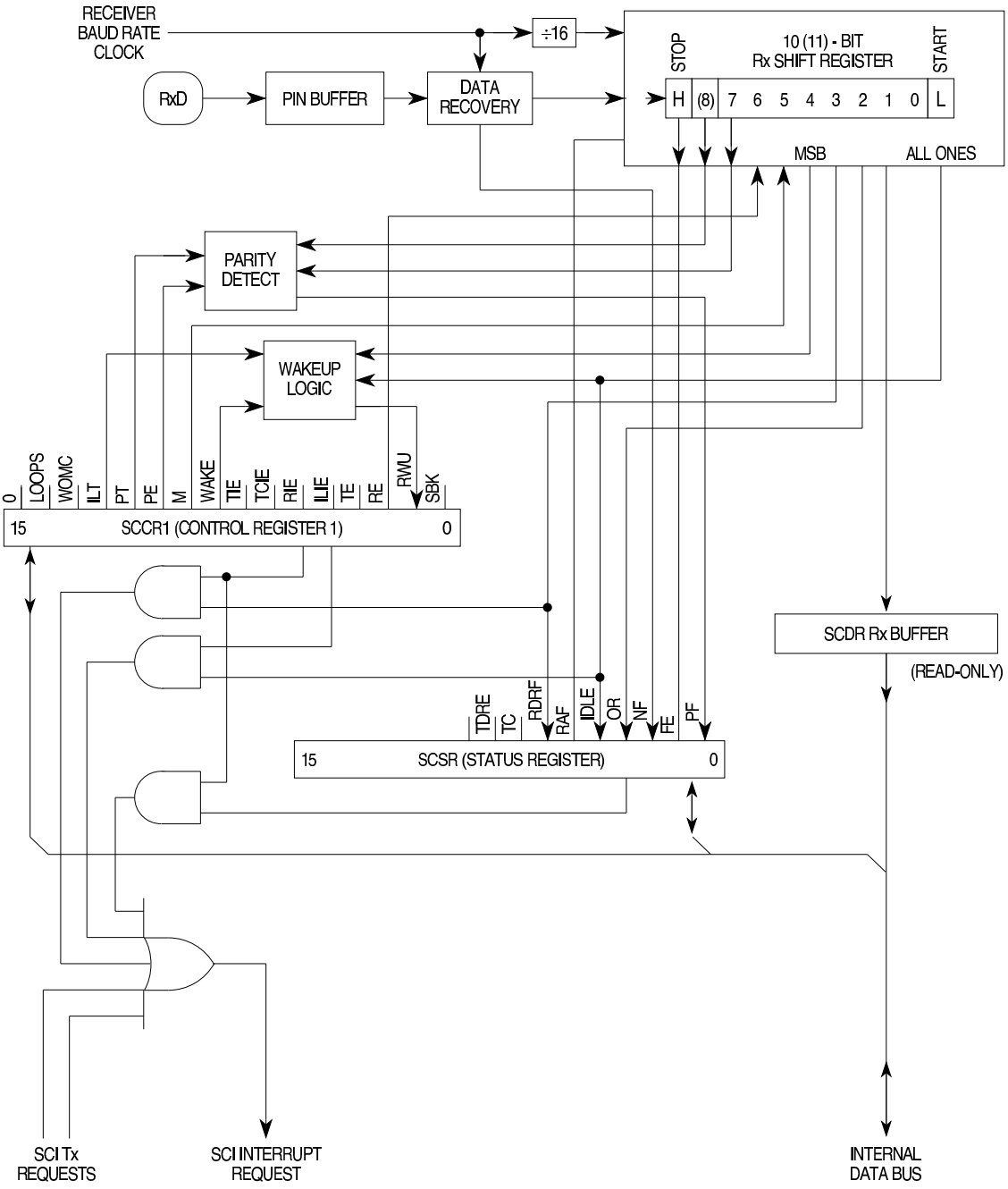
Struktura QSM (*Queued Serial Module*)



Nadajnik transmisji asynchronicznej (SCI)



Odbiornik transmisji asynchronicznej (SCI)



QSMCR — QSM Configuration Register **$YFFC00**

15	14	13	12	11	10	9	8	7	6	5	4	3			0
STOP	FRZ1	FRZ0	0	0	0	0	0	SUPV	0	0	0	IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

QILR — QSM Interrupt Levels Register **$YFFC04**

15	14	13		11	10		8	7							0
0	0	ILQSPI			ILSCI			QIVR							
RESET:															
0	0	0	0	0	0	0	0								

QIVR — QSM Interrupt Vector Register **$YFFC05**

15							8	7							0
QILR								INTV							
RESET:															
								0	0	0	0	1	1	1	1

SCCR0 — SCI Control Register 0 **$YFFC08**

15	14	13	12												0
0	0	0	SCBR												
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

$$SCIbaud = \frac{f_{SYS}}{32 \cdot SCBR}$$

SCCR1 — SCI Control Register 1 **$YFFC0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SCSR — SCI Status Register **$YFFC0C**

15							9	8	7	6	5	4	3	2	1	0
NOT USED								TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF
RESET:																
								1	1	0	0	0	0	0	0	0

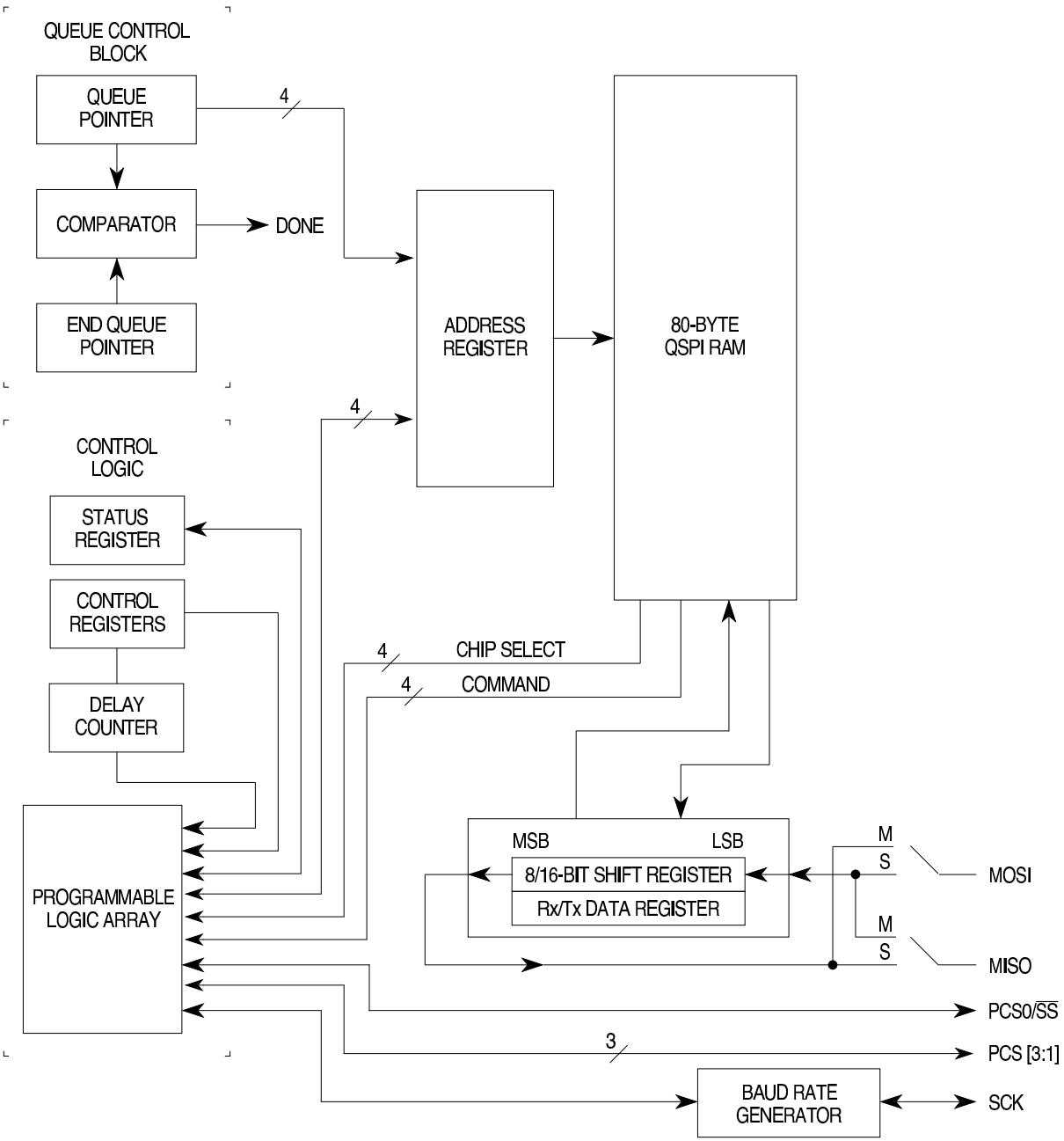
Obsługa programowa SCI

```
#define QSMBASE    0xfffffc00
#define QMCR    (*((WORD *) (QSMBASE+0x00)))        /* QSM Configuration */
#define QILR    (*((BYTE *) (QSMBASE+0x04)))      /* QSM Interrupt Level */
#define QIVR    (*((BYTE *) (QSMBASE+0x05)))     /* QSM Interrupt Vector */
#define SCCR0   (*((WORD *) (QSMBASE+0x08)))           /* SCI Control 0 */
#define SCCR1   (*((WORD *) (QSMBASE+0x0a)))           /* SCI Control 1 */
#define QsmRIE     0x0020                   /* Receive Interrupt Enable */
#define QsmTE      0x0008                          /* Transmit Enable */
#define QsmRE      0x0004                           /* Receive Enable */
#define SCSR    (*((volatile WORD *) (QSMBASE+0x0c)))      /* SCI Status */
#define QsmTDRE    0x0100               /* Transmit DAta Register Empty */
#define QsmRDRF    0x0040                 /* Receive Data Register Full */
#define QsmERR     0x000f                  /* Receiver error flags mask */
#define SCDR    (*((volatile WORD *) (QSMBASE+0x0e)))        /* SCI Data */

/* Stale i makra */
#define CLOCK       16777216                /* czestotliwosc zegara CPU */
#define ScBr(x)     (CLOCK/32 + ((x)/2))/(x)   /* dzielnik predkosci SCI */
#define RxIEnable   SCCR1 |= 0x0020          /* dblokowanie przerwan Rx */

/* Parametry wybierane przez uzytkownika */
#define BAUD        9600                         /* predkosc transmisji */
#define SCI_VECT    112                            /* wektor przerwania */
#define SCI_LEVEL   6                              /* poziom przerwania */
#define BUF_LEN     16                   /* wielkosc bufora odbiornika */

/* Deklaracje zmiennych globalnych */
BYTE                *RxWrPtr;
BYTE                *RxRdPtr;
BYTE                RxBuf[BUF_LEN];
int                 RxCnt;
```

```
/* Procedura obslugi przerwania */
interrupt void SciSvc()
{
    register char ch;

    if((SCSR & (QsmRDRF | QsmOR | QsmNF | QsmFE | QsmPF))==QsmRDRF)
    {
        ch = SCDR;                          /* odczytanie - zgaszenie RDRF */
        if(RxCnt < BUF_LEN)                     /* jest miejsce w buforze? */
        {
            RxCnt += 1;
            *RxWrPtr++ = ch;
            if(RxWrPtr >= RxBuf + BUF_LEN) RxWrPtr = RxBuf;/* cyklicznosc */
        }
        return;
    }
}

/* Czytanie znaku z bufora cyklicznego */
int GetChar()
{
    register int ch;

    if(RxCnt)                                   /* sa znaki do odczytania? */
    {
        ch = *RxRdPtr++;
        if(RxRdPtr >= RxBuf + BUF_LEN) RxRdPtr = RxBuf;   /* cyklicznosc */
        RxCnt -= 1;
        return ch;
    }
    else return -1;
}
```

```
main()
{
    int ch;
    int pom;

    *((void (**) ())(4*SCI_VECT)) = SciSvc;     /* wektor przerwania SCI */
    QMCR    = 0x0083;                               /* SUPV, IARB=0x3 */
    QILR    = SCI_LEVEL;                          /* poziom przerwania */
    QIVR    = SCI_VECT;                           /* wektor przerwania */

    SCCR0   = ScBr(BAUD);                       /* dzielnik zegara SCI */
    SCCR1   = QsmTE | QsmRE;                        /* 8N1, TxEn RxEn */

    RxRdPtr = RxWrPtr = RxBuf; /* ustawienie wskaznikow bufora odiornika */
    RxCnt   = 0;                           /* wyzerowanie licznika znakow */

    pom     = SCSR;
    pom     += SCDR;              /* usuniecie ew. wczesniejszych znakow */
    RxIEnable;                         /* wlaczenie przerwan odbiornika  */
    asm { ANDI #0xf8ff,SR
    }                                 /* ustawienie poziomu przerwan na 0 */

    while(1)
    {
        if((ch = GetChar()) != -1)                  /* jesli jest znak */
        {
            while(!(SCSR & QsmTDRE));  /* czekanie na gotowosc nadajnika */
            SCDR = ch;
        }
    }
}
```

Blok QSPI (*Queued Serial Peripheral Interface*)



D.4.9 PQSPAR — PORT QS Pin Assignment Register $YFFC16
DDRQS — PORT QS Data Direction Register $YFFC17

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PQSPA6	PQSPA5	PQSPA4	PQSPA3	0	PQSPA1	PQSPA0	DDQS7	DDQS6	DDQS5	DDQS4	DDQS3	DDQS2	DDQS1	DDQS0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

D.4.10 SPCR0 — QSPI Control Register 0 $YFFC18

15	14	13	10	9	8	7	0
MSTR	WOMQ	BITS		CPOL	CPHA	SP	
RESET:							
0	0	0	0	0	1	0	0

D.4.11 SPCR1 — QSPI Control Register 1 $YFFC1A

15	14	8	7	0
SPE	DSCKL		DTL	
RESET:				
0	0	0	0	0

D.4.12 SPCR2 — QSPI Control Register 2 $YFFC1C

15	14	13	12	11	8	7	6	5	4	3	0
SPIFIE	WREN	WRTO	0	ENDQP		0	0	0	0	NEWQP	
RESET:											
0	0	0	0	0	0	0	0	0	0	0	0

D.4.13 SPCR3 — QSPI Control Register 3 $YFFC1E
SPSR — QSPI Status Register $YFFC1F

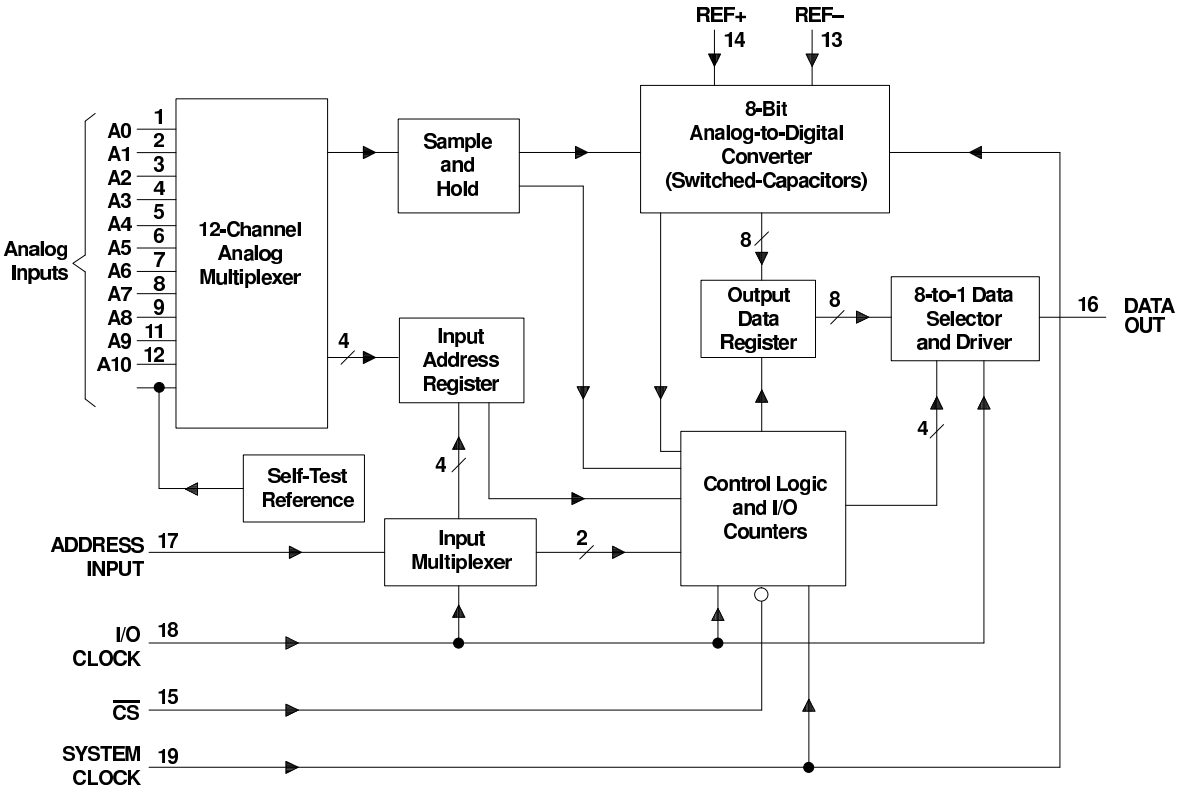
15	14	13	12	11	10	9	8	7	6	5	4	3	0
0	0	0	0	0	LOOPQ	HMIE	HALT	SPIF	MODF	HALTA	0	CPTQP	
RESET:													
0	0	0	0	0	0	0	0	0	0	0	0	0	0

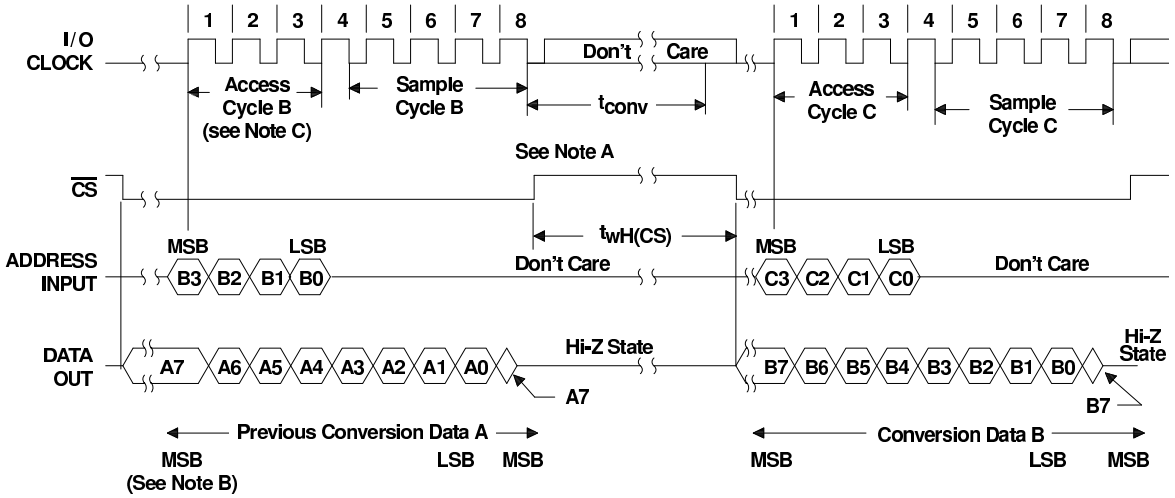
D.4.16 CR[0:F] — Command RAM $YFFD40–$YFFD4F

7	6	5	4	3	2	1	0
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*
—	—	—	—	—	—	—	—
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

COMMAND CONTROL | PERIPHERAL CHIP SELECT

Przetwornik A/C TLC541





NOTES: A. The conversion cycle, which requires 36 system clock periods, is initiated on the 8th falling edge of I/O CLOCK after $\overline{CS}$ goes low for the channel whose address exists in memory at that time. If $\overline{CS}$ is kept low during conversion, I/O CLOCK must remain low for at least 36 system clock cycles to allow conversion to be completed.

B. The most significant bit (MSB) will automatically be placed on the DATA OUT bus after $\overline{CS}$ is brought low. The remaining seven bits (A6–A0) will be clocked out on the first seven I/O CLOCK falling edges.

C. To minimize errors caused by noise at $\overline{CS}$, the internal circuitry waits for three system clock cycles (or less) after a chip select falling edge is detected before responding to control input signals. Therefore, no attempt should be made to clock-in address data until the minimum chip-select setup time has elapsed.

Obsługa programowa QSPI

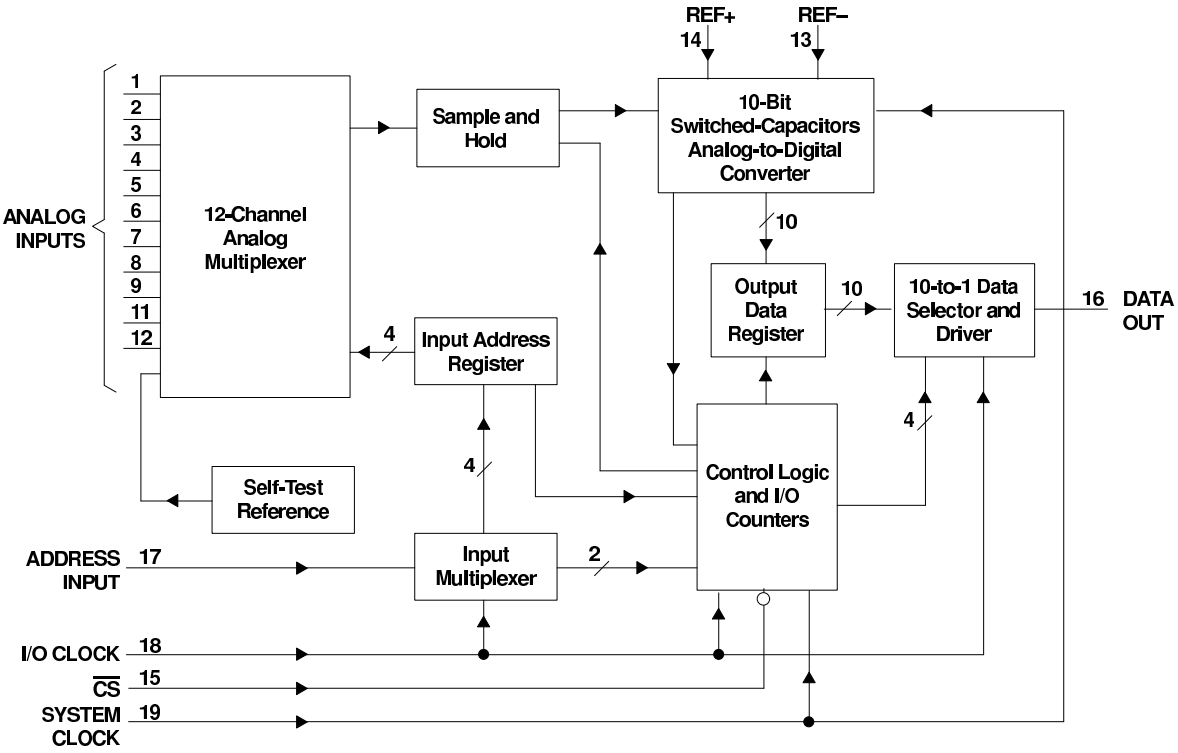
```
#define QSMBASE 0xfffffc00

#define QPDR    (*((volatile BYTE *) (QSMBASE+0x15)))     /* QSM Port Data */
#define QPAR    (*((BYTE *) (QSMBASE+0x16)))         /* QSM Pin Assignment */
#define QDDR    (*((BYTE *) (QSMBASE+0x17)))         /* QSM Data Direction */

#define SPCR0   (*((WORD *) (QSMBASE+0x18)))            /* QSPI Control 0  */

#define SPCR1   (*((WORD *) (QSMBASE+0x1a)))             /* QSPI Control 1 */
#define QsmSPE     0x8000                          /* flaga uruchomienia QSPI */

#define SPCR2   (*((WORD *) (QSMBASE+0x1c)))             /* QSPI Control 2 */

#define SPCR3   (*((BYTE *) (QSMBASE+0x1e)))             /* QSPI Control 3 */

#define SPSR    (*((volatile BYTE *) (QSMBASE+0x1f)))       /* QSPI Status */
#define QsmSPIF    0x80                     /* flaga zakonczenia transmisji */

#define QREC    ( ((volatile WORD *) (QSMBASE+0x100)))  /* QSPI Rx RAM ptr */

#define QTRAN   ( ((WORD *) (QSMBASE+0x120)))           /* QSPI Tx RAM ptr */

#define QCOMD   ( ((BYTE *) (QSMBASE+0x140)))          /* QSPI Cmd RAM ptr */
#define QsmCONT    0x80                               /* nastapi kontynuacja */
#define QsmBITSE   0x40                /* wybor dlugosci slowa wedlug BITS */
#define QsmDT      0x20                  /* wlaczenie opoznienia koncowego */
#define QsmDSCK    0x10               /* wlaczenie opoznienia poczatkowego */
#define QsmSELMASK 0x0f                                /* maska adresu SLAVE */
```

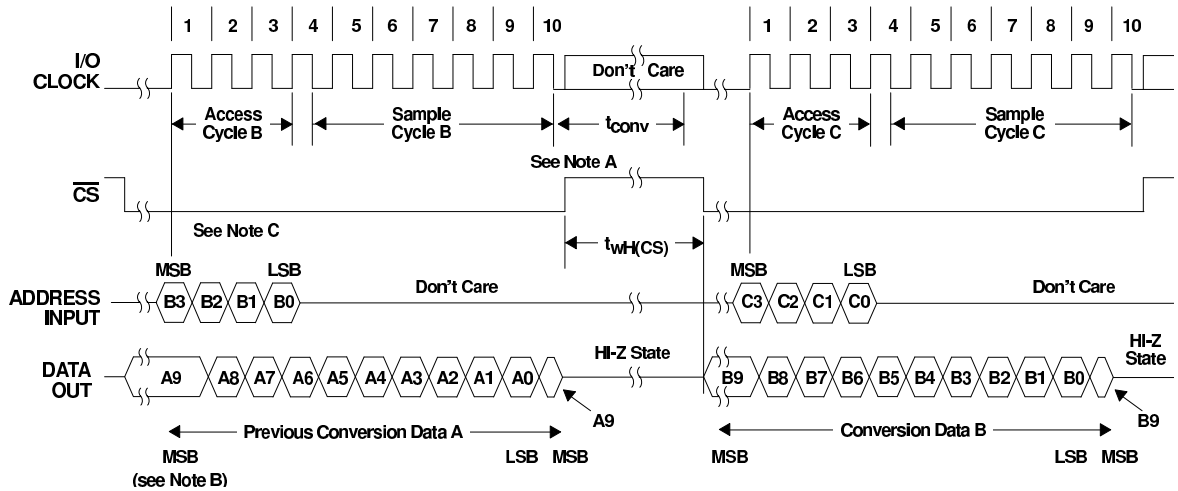
```
#define ADCSEL      0xc                 /* wybor ADC stanem niskim na PCS1 */
#define DESELECT    0xe                   /* zaden SLAVE nie jest wybrany */

int ReadAdc(WORD kanal)
{
    if(SPCR1 & QsmSPE)                   /* jesli QSPI jest wlaczone, to: */
    {
        while(!(SPSR & QsmSPIF));            /* czekaj na koniec operacji */
        SPSR &= (~QsmSPIF);                     /* zgas flage zakonczenia */
    }

    QDDR = 0xfe;                            /* PCSx, SCK, MOSI - wyjscia */
    QPDR = DESELECT<<3;                        /* wylaczenie wszystkiego */
    QPAR = 0x7b;

    SPCR0 = 0xa817;                     /* master, cpol 0, cpha 0, 493kHz */
    SPCR1 = 0x7fc0;                                 /* DSCKL=2us, DTL=22us */
    SPCR2 = 0x0100;                       /* kolejka od 0 do 1 (2 slowa) */

    QTRAN[0] = kanal<<4;                                  /* kanal mierzony */
    QTRAN[1] = 11<<4;                                      /* kanal testowy */

    QCOMD[0] = QsmDT | QsmDSCK | ADCSEL;                          /* 8 bitow */
    QCOMD[1] = QsmDT | QsmDSCK | ADCSEL;                          /* 8 bitow */

    SPCR1 |= QsmSPE;                                        /* wlacz QSPI */
    while(!(SPSR & QsmSPIF));                /* czekaj na koniec operacji */
    SPSR &= (~QsmSPIF);                         /* zgas flage zakonczenia */

    return (QREC[1]);                                  /* wynik w QREC[1] */
}
```

Uruchomienie QSPI w trybie cyklicznym

```
main()
{
    if(SPCR1 & QsmSPE)                  /* jesli QSPI jest wlaczone, to: */
    {
        while(!(SPSR & QsmSPIF));           /* czekaj na koniec operacji */
        SPSR &= (~QsmSPIF);                    /* zgas flage zakonczenia */
    }

    QDDR = 0xfe;                            /* PCSx, SCK, MOSI - wyjscia */
    QPDR = DESELECT<<3;                       /* wylaczenie wszystkiego */
    QPAR = 0x7b;

    SPCR0 = 0x8017;                     /* master, cpol 0, cpha 0, 493kHz */
    SPCR1 = 0x7fc0;                                 /* DSCKL=2us, DTL=22us */
    SPCR2 = 0x430F;       /* WREN, petla od 0 do 3 (4 slowa), start od F */

    QTRAN[15]= 0<<4;                                                /* rozruch */
    QTRAN[0] = 1<<4;                                                /* kanal 0 */
    QTRAN[1] = 2<<4;                                                /* kanal 1 */
    QTRAN[2] = 3<<4;                                                /* kanal 2 */
    QTRAN[3] = 0<<4;                                                /* kanal 3 */

    QCOMD[15]= QsmDT | QsmDSCK | ADCSEL;                            /* 8 bitow */
    QCOMD[0] = QsmDT | QsmDSCK | ADCSEL;                            /* 8 bitow */
    QCOMD[1] = QsmDT | QsmDSCK | ADCSEL;                            /* 8 bitow */
    QCOMD[2] = QsmDT | QsmDSCK | ADCSEL;                            /* 8 bitow */
    QCOMD[3] = QsmDT | QsmDSCK | ADCSEL;                            /* 8 bitow */

    SPCR1 |= QsmSPE;                                          /* wlacz QSPI */
    while(!(SPSR & QsmSPIF));           /* czekaj na koniec operacji */

    while(1);
}
```

Przetwornik A/C TLC1541





NOTES: A. The conversion cycle, which requires 44 system clock periods, initiates on the tenth falling edge of the I/O clock after $\overline{CS}$ goes low for the channel whose address exists in memory at that time. When $\overline{CS}$ is kept low during conversion, the I/O clock must remain low for at least 44 system clock cycles to allow the conversion to complete.

B. The most significant bit (MSB) is automatically placed on the DATA OUT bus after $\overline{CS}$ is brought low. The remaining nine bits (A8–A0) clock out on the first nine I/O clock falling edges.

C. To minimize errors caused by noise at the $\overline{CS}$ input, the internal circuitry waits for three system clock cycles (or less) after a chip-select falling edge is detected before responding to control input signals. Therefore, no attempt should be made to clock-in address data until the minimum chip-select setup time elapses.

Obsluga przetwornika 10-bitowego

```
main()
{
    if(SPCR1 & QsmSPE)                  /* jesli QSPI jest wlaczone, to: */
    {
        while(!(SPSR & QsmSPIF));           /* czekaj na koniec operacji */
        SPSR &= (~QsmSPIF);                    /* zgas flage zakonczenia */
    }

    QDDR = 0xfe;                            /* PCSx, SCK, MOSI - wyjscia */
    QPDR = DESELECT<<3;                       /* wylaczenie wszystkiego */
    QPAR = 0x7b;

    SPCR0 = 0xa817;            /* master, 10-bit, cpol 0, cpha 0, 493kHz */
    SPCR1 = 0x7fc0;                                 /* DSCKL=2us, DTL=22us */
    SPCR2 = 0x430F;       /* WREN, petla od 0 do 3 (4 slowa), start od F */

    QTRAN[15]= 0<<6;                                              /* rozruch */
    QTRAN[0] = 1<<6;                                              /* kanal 0 */
    QTRAN[1] = 2<<6;                                              /* kanal 1 */
    QTRAN[2] = 3<<6;                                              /* kanal 2 */
    QTRAN[3] = 0<<6;                                              /* kanal 3 */

    QCOMD[15]= QsmDT | QsmBITSE | QsmDSCK | ADCSEL;          /* 10 bitow */
    QCOMD[0] = QsmDT | QsmBITSE | QsmDSCK | ADCSEL;          /* 10 bitow */
    QCOMD[1] = QsmDT | QsmBITSE | QsmDSCK | ADCSEL;          /* 10 bitow */
    QCOMD[2] = QsmDT | QsmBITSE | QsmDSCK | ADCSEL;          /* 10 bitow */
    QCOMD[3] = QsmDT | QsmBITSE | QsmDSCK | ADCSEL;          /* 10 bitow */

    SPCR1 |= QsmSPE;                                        /* wlacz QSPI */
    while(!(SPSR & QsmSPIF));               /* czekaj na koniec operacji */

    while(1);
}
```