16.1 I2C NO ATMEGA328

Como mencionado anteriormente, o ATmega usa o termo TWI para sua interface I2C. As principais características dessa interface no ATmega328 são:

- Operação como mestre ou escravo.
- Endereçamento de 7 bits.
- Permite a operação com vários mestres.
- Velocidade de até 400 kHz na transferência de dados.
- *Drivers* de saída com limitação da taxa de subida (*Slew Rate*).
- Circuito supressor de ruído no barramento.
- O reconhecimento de endereço permite 'despertar' a CPU do modo *Sleep*.

Para evitar conflitos na via de dados (SDA) do ATmega, os pinos são chaveados como I/O dependendo das necessidades do protocolo (dreno aberto). O pino de *clock* (SCL) também é alterado como I/O. Em ambos os pinos são necessários resistores de *pull-up* (comuns ao I2C). Assim, o valor padrão para SDA e SCL é o nível lógico alto (1). Os bits são sempre lidos na borda de descida do SCL. Os dispositivos ligados ao barramento possuem endereços individuais e os mecanismos inerentes ao trabalho com o I2C gerenciam o protocolo.

Cada bit transferido para o barramento é acompanhado por um pulso da linha de *clock*. O nível lógico do dado deve estar estável quando a linha de *clock* for para o nível alto. A única exceção acontece quando se geram a condição de início e parada da comunicação. Esse fato é ilustrado na fig. 16.2.

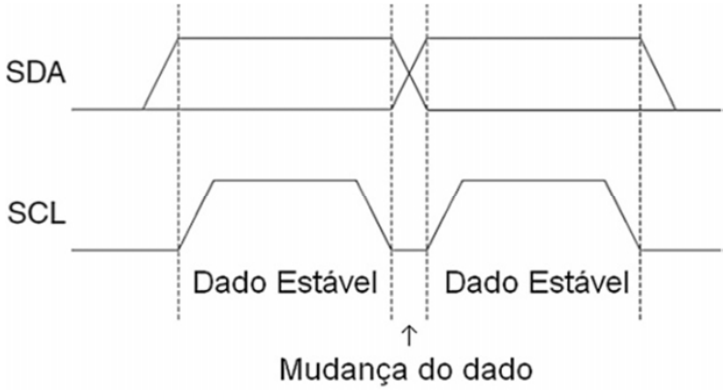

Fig. 16.2 – Validade dos dados no barramento I2C.

Os sinais possíveis no barramento são:

- **Condição de Início:** usada para informar à rede que uma transmissão irá ocorrer.
- **Condição de Parada:** usada para informar à rede que a transmissão acabou.
- **Condição de Reinício:** pode ser executada antes da parada ou na necessidade de reenvio do primeiro byte antes da finalização com a parada.
- **Condição de *Acknowledge* (ACK):** resposta pelo mestre ou escravo, no lugar do 9º bit, informando se o dado foi corretamente recebido ou não.
- **Transmissão de endereço**: independentemente da informação a ser transmitida (endereço de 7 bits e dados de 8 bits), a comunicação sempre é feita em pacotes de 8 bits e mais um retorno de *acknowledge*.
- **Transmissão de dados:** depois do término da transferência da parte relacionada ao endereço, o mestre ou o escravo deverão transmitir um ou mais bytes de dados.
- **Pausa:** Mantendo a linha de dados em nível baixo para processar uma informação, por exemplo.

A transmissão é iniciada quando o mestre executa uma condição de início no barramento e termina quando o mestre executa uma condição de parada. Entre o início e a parada, o barramento é considerado ocupado e nenhum outro mestre deve tentar controlar o barramento. Um caso

especial ocorre quando um novo início ocorre entre uma condição de início e parada. Isso é conhecido como reinício e é realizado quando o mestre deseja iniciar uma nova transmissão sem perder o controle sobre o barramento. Após o reinício, o barramento é considerado ocupado até a próxima parada. Esse fato é ilustrado na fig. 16.3.

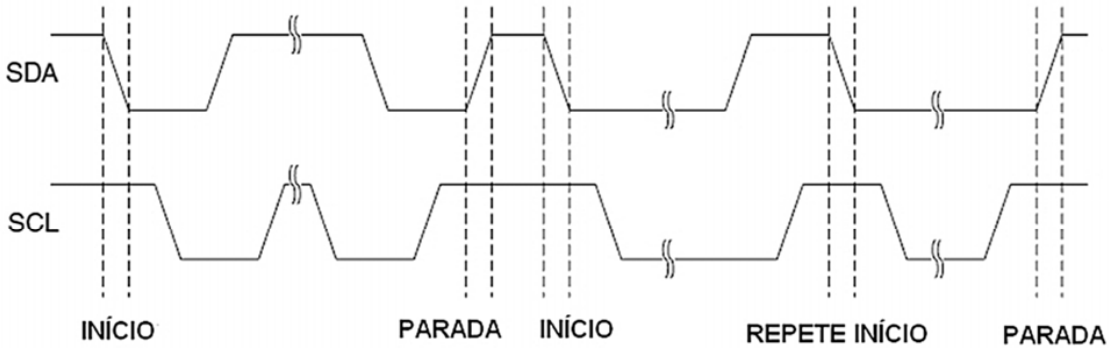

Fig. 16.3 – Condição de início, parada e reinício.

Todos os endereços transmitidos no barramento TWI são compostos por 9 bits, onde 7 são os bits referentes ao endereço, 1 bit de controle para leitura/escrita (L/E) e 1 bit de confirmação (ACK). Se o bit de leitura/escrita é 1, uma leitura será realizada, caso contrário, uma escrita. Quando o escravo reconhece que está sendo endereçado, ele sinaliza colocando a linha SDA em baixo no nono ciclo do SCL (ACK). Se o endereço do escravo estiver ocupado, ou por outra razão ele não puder atender ao mestre, a linha SDA deve ser mantida alta no ciclo de *clock* do ACK. O mestre pode, então, transmitir uma condição de parada ou de reinício para iniciar uma nova transmissão.

O bit mais significativo (MSB) do endereço ou dado é transmitido primeiro. O endereço do escravo pode ser determinado pelo projetista. Entretanto, o endereço 0x00 é reservado para uma chamada geral. Na fig. 16.4, é apresentado o formato do pacote de bits referente ao endereço.

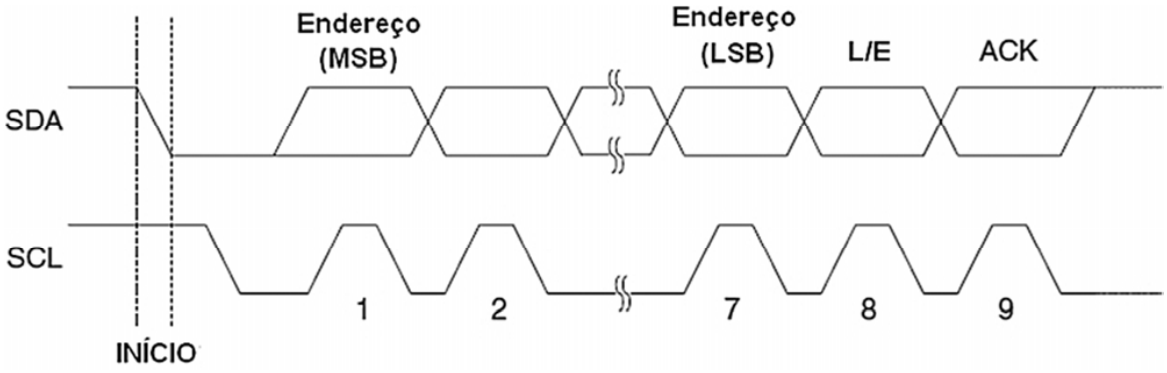

Fig. 16.4 – Formato do pacote de bits referentes ao endereço.

Quando uma chamada geral é realizada, todos os escravos devem responder colocando a linha SDA em nível baixo no ciclo de ACK. Essa chamada é realizada quando o mestre deseja transmitir a mesma mensagem aos escravos do sistema. Quando uma chamada geral é seguida por um bit de escrita, todos os escravos devem colocar a linha SDA em nível baixo no ciclo de ACK. Os pacotes seguintes de dados serão recebidos por todos os escravos que responderam. Não se deve transmitir uma chamada geral seguida de um bit de leitura, pois todos os escravos escreverão no barramento ao mesmo tempo.

Um sinal ACK é emitido ou transmitido pelo mestre ou escravo colocando a linha SDA em zero durante o nono ciclo do SCL. Se a linha for deixada em nível alto é sinalizado um NÃO-ACK (NACK). Quando o receptor receber o último byte ou por alguma razão não puder receber mais bytes, ele deve informar o transmissor enviando um NACK.

Em resumo, todos os pacotes transmitidos no barramento TWI possuem nove bits, um byte mais um bit de resposta. Durante a transmissão de dados, o mestre gera o *clock* e a condição de início e de parada, enquanto o escravo é responsável por responder se recebeu o dado.

Uma transmissão típica é apresentada na fig. 16.5. Notar que vários bytes de dados podem ser transmitidos entre o endereço do escravo mais o

bit de leitura/escrita e a condição de parada, dependendo do protocolo implementado pelo software.

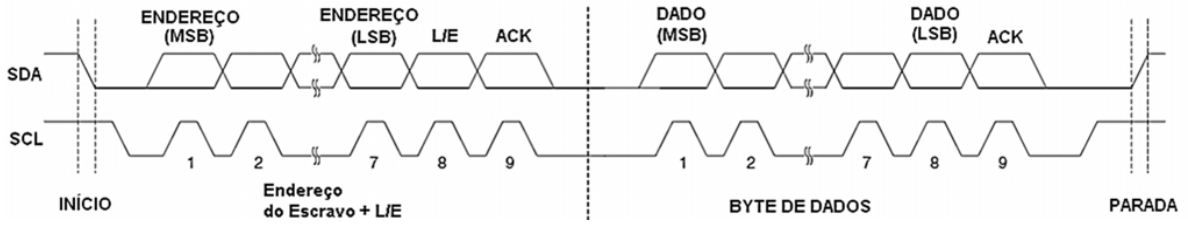

Fig. 16.5 – Transmissão de dados típica para o protocolo I2C.

16.2 REGISTRADORES DO TWI

A frequência do sinal SCL é controlada ajustando-se o registrador da taxa de bits (TWBR) e os bits de *prescaler* (divisor de *clock*) no registrador de *status* do TWI (TWSR). O modo escravo não depende desse ajuste, entretanto, a frequência da CPU deve ser no mínimo 16 vezes maior que a frequência do sinal SCL. Assim, a frequência do sinal SCL é dada por:

$$f_{\mathrm{SCL}} = \frac{f_{osc}}{16+(2\times \mathrm{TWBR}\times \mathrm{TWPS})} \qquad [\mathrm{Hz}] \qquad (16.1)$$

onde f_{osc} é a frequência de trabalho da CPU, TWBR é o valor do registrador de ajuste da taxa de bits e TWPS é o valor do *prescaler* no registrador TWSR, definido pelos bits TWPS0:1.

TWBR – TWI Bit *Rate Register*

Bit		7	6	5	4	3	2	1	0
	TWBR	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Este registrador seleciona o fator de divisão do gerador da taxa de bits, conforme eq. 16.1.

TWSR – TWI *Status Register*

Bit		7	6	5	4	3	2	1	0
	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0
Lê/Escreve		L	L	L	L	L	L	L/E	L/E
Valor Inicial		1	1	1	1	1	0	0	0

Bits 7:3 – TWS7:3 – TWI *Status*
Estes 5 bits refletem o estado lógico do TWI. Como o registrador também contém 2 bits de *prescaler*, o software deve mascará-los durante a leitura dos bits de estado.

Bits 1:0 – TWPS1:0 – TWI *Prescaler Bits*
Estes bits podem ser lidos e escritos e controlam o *prescaler* da taxa de bits (eq. 16.1), conforme tab. 16.1.

Tab. 16.1 – *Prescaler* da taxa de bits do TWI.

TWPS1	TWPS0	Valor do *Prescaler*
0	0	1
0	1	4
1	0	16
1	1	64

TWCR – TWI *Control Register*

Bit		7	6	5	4	3	2	1	0
	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Lê/Escreve		L/E	L/E	L/E	L/E	L	L/E	L	L/E
Valor Inicial		0	0	0	0	0	0	0	0

É empregado para controlar a operação do TWI: habilitar, iniciar uma condição de início, gerar o ACK de recebimento, gerar uma condição de parada. Mantém o barramento em nível lógico alto enquanto o dado para o barramento é escrito no TWDR. Também indica uma colisão de dados se houver tentativa de escrita no TWDR enquanto esse estiver inacessível.

Bit 7 – TWINT – TWI *Interrupt Flag*
Este bit é ativo por hardware quando o TWI conclui sua atividade corrente e espera uma resposta do software de aplicação. Se o bit I do SREG e o bit TWIE do TWCR estiverem ativos, a CPU é desviada para o endereço do vetor de interrupção correspondente. Enquanto TWINT=1, o período baixo do SCL é mantido. Ele deve ser limpo por software pela escrita de 1. Limpar o bit TWINT prossegue a operação do TWI, logo os acessos aos registradores TWAR, TWSR e TWDR já devem estar completos.

Bit 6 – TWEA – TWI *Enable Acknowledge Bit*
Este bit controla a geração do pulso de ACK. Se for escrito 1, um pulso de ACK é gerado no barramento TWI se as seguintes condições forem encontradas:

1. O endereço próprio do dispositivo como escravo tenha sido recebido.

2. Uma chamada geral tenha sido recebida, enquanto o bit TWGCE do TWAR estiver em 1.
3. Um byte de dados tenha sido recebido no modo mestre ou escravo.

Zerando o bit, o microcontrolador pode ser virtualmente desconectado por um intervalo de tempo do barramento TWI. O reconhecimento de endereço pode ser reabilitado escrevendo-se 1 no bit.

Bit 5 – TWSTA – TWI *Start Condition Bit*
Este bit ativo indica que o modo de comunicação será mestre. Deve ser limpo por software após uma condição de início ser transmitida.

Bit 4 – TWSTO – TWI *Stop Condition Bit*
Quando em 1, gera uma condição de parada, sendo limpo automaticamente pelo hardware. No modo escravo, pode ser utilizado para a recuperação de uma condição de erro, não gerando uma condição de parada, mas colocando os pinos do barramento em alta impedância e em uma condição de não endereçamento.

Bit 3 – TWWC – TWI *Write Collision Flag*
Este bit é ativo quando se tenta escrever no registrador de dados TWDR enquanto TWINT estiver em 0. É limpo escrevendo-se em TWDR quando TWINT estiver em 1.

Bit 2 – TWEN – TWI *Enable Bit*
Este bit habilita a operação do TWI e ativa a interface TWI.

Bit 0 – TWIE – TWI *Interrupt Enable*
Quando este bit estiver em 1, bem como o bit I do registrador SREG, a interrupção do TWI estará ativa pelo tempo que o bit TWINT estiver em nível lógico alto.

TWDR – TWI *Data Register*

Bit		7	6	5	4	3	2	1	0
	TWDR	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		1	1	1	1	1	1	1	1

No modo de transmissão, o TWDR contém o próximo byte a ser transmitido. No modo de recepção, o TWDR contém o último byte recebido. Ele pode ser escrito se o TWI não estiver enviando ou recebendo um byte. O bit ACK é controlado automaticamente pelo TWI e a CPU não pode acessá-lo diretamente.

TWAR – TWI *(Slave) Address Register*

Bit		7	6	5	4	3	2	1	0
	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		1	1	1	1	1	1	1	0

Este registrador deve ser carregado com o endereço de 7 bits que identificará o ATmega328 no barramento TWI quando no modo escravo.

O bit 0 (TWGCE) é usado para habilitar o reconhecimento de chamada geral (0x00). Existe um comparador de endereço que gera um pedido de interrupção se houver igualdade no endereço do escravo ou se a chamada geral estiver habilitada.

16.3 USANDO O TWI

O TWI é orientado a byte e baseado em interrupções. Interrupções são feitas após a ocorrência de eventos no barramento, como a recepção de um byte ou a transmissão de uma condição de início. Sendo o TWI baseado em interrupções, o software fica livre para executar outras operações durante a transferência de bytes. Um exemplo simples da aplicação do TWI é apresentado na fig. 16.6, no qual o mestre deseja transmitir um simples byte de dados para o escravo.

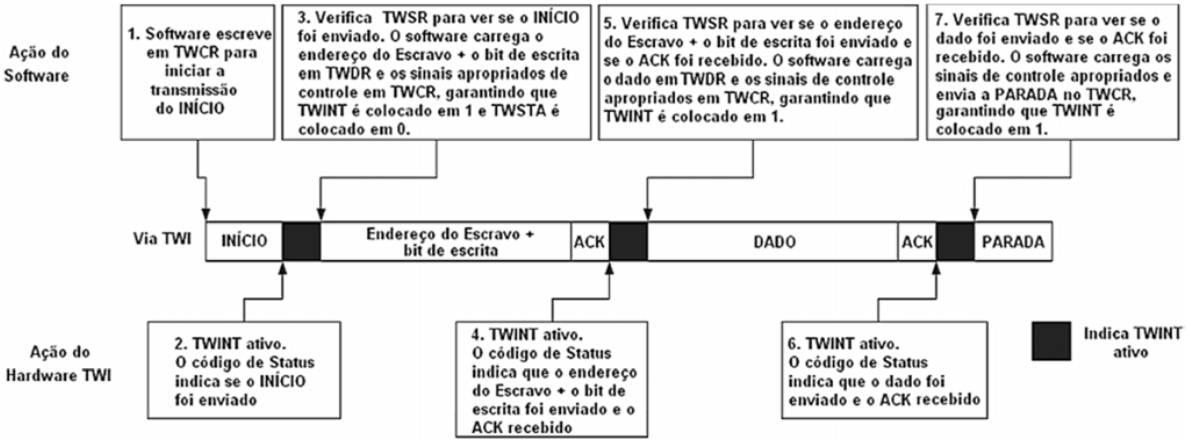

Fig. 16.6 – Transmissão típica com o TWI.

O código a seguir, sem o uso de interrupção, ilustra a programação para o exemplo acima, conforme manual do fabricante.

```
//============================================================================== //
//                  CÓDIGO EXEMPLO PARA USO DO TWI                              //
//============================================================================== //
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);//Envia a condição de início
//------------------------------------------------------------------------------
    while (!(TWCR & (1<<TWINT)));  //Espera o TWINT ser ativo indicando que a
                                   //condição de início foi transmitida
//------------------------------------------------------------------------------
    if ((TWSR & 0xF8) != START)    //Verifica o valor do TWI no registrador de status.
        ERROR();                   /*Mascara os bits do prescaler. Se o status
                                   for diferente da condição de início chama
                                   uma função para o tratamento do erro*/
//------------------------------------------------------------------------------
    TWDR = SLA_W;                  //Carrega o endereço do escravo para a escrita
    TWCR = (1<<TWINT) | (1<<TWEN); /*limpa o bit TWINT no TWCR para começar a
                                   transmissão do endereço*/
//------------------------------------------------------------------------------
    while (!(TWCR & (1<<TWINT)));  /*Espera pela ativação do bit TWINT
                                   indicando que o endereço do escravo + o bit de
                                   escrita foi enviado e que o ACK/NACK foi recebido*/
//------------------------------------------------------------------------------
    if ((TWSR & 0xF8) != MT_SLA_ACK)//Verifica o valor do registrador de
        ERROR();                   /*status do TWI. Mascara os bits do prescaler. Se
                                   o status for diferente de MT_SLA_ACK chama uma
                                   função para o tratamento do erro*/
//------------------------------------------------------------------------------
    TWDR = DATA;    //Carrega DATA no registrador TWDR. Limpa o bit TWINT no TWCR
    TWCR = (1<<TWINT) | (1<<TWEN); //para iniciar a transmissão do dado.
//------------------------------------------------------------------------------
    while (!(TWCR & (1<<TWINT)));/*Espera o bit TWINT ser ativo, indicando que
                                 o dado foi transmitido e que o ACK/NACK foi recebido*/
//------------------------------------------------------------------------------
    if ((TWSR & 0xF8) != MT_DATA_ACK)   //Verifica o valor do registrador de
         ERROR();             /*status do TWI. Mascara os bits do prescaler. Se o
                              status for diferente de MT_DATA_ACK chama uma função
                              para o tratamento do erro*/
//------------------------------------------------------------------------------
    TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);//Transmite a condição de parada
//==============================================================================
```

Quando se utiliza o AVR-GCC como compilador, pode-se empregar sua biblioteca para o trabalho com o TWI. Nela encontram-se as definições dos estados que podem ocorrer no registrador TWSR. Toda e qualquer operação que é realizada pelo módulo TWI será armazenada nesse registrador de *status*. Basta, então, analisar o seu conteúdo para se saber o que aconteceu. O TWSR indicará sucesso ou erro na transmissão/recepção dos dados (ver o manual do ATmega para maiores detalhes). A seguir, é apresentado um conjunto de definições que podem facilitar a vida do programador para o emprego do módulo TWI. As definições de teste verificam que houve erro na ação efetuada e chamam uma rotina para o tratamento do erro. Essa rotina deve ser escrita pelo programador e dependerá de como ele deseja tratar o erro, como, por exemplo, repetir um comando mal sucedido.

```
#include <util/twi.h>               //definições para o uso da interface i2c
...
//-------------------------------------------------------------------------------
// Definições para o uso da comunicação I2C
//-------------------------------------------------------------------------------
#define start_bit()                 TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
#define stop_bit()                  TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO)
#define espera_envio()              while (!(TWCR & (1<<TWINT)))
#define envia_byte()                TWCR = (1<<TWINT) | (1<<TWEN)
#define recebe_byte()               TWCR = (1<<TWINT) | (1<<TWEN)
#define espera_recebimento()        while (!(TWCR & (1<<TWINT)))
#define recebe_byte_ret_nack()      TWCR = (1<<TWINT) | (1<<TWEN)

//A rotina de tratamento de erro é por conta do programador e da sua lógica de programação

#define teste_envio_start()             if((TWSR & 0xF8) != TW_START)
                                            trata_erro();
#define teste_envio_end_escrita()       if((TWSR & 0xF8) != TW_MT_SLA_ACK)
                                            trata_erro();
#define teste_envio_dado()              if((TWSR & 0xF8) != TW_MT_DATA_ACK)
                                            trata_erro();
#define teste_envio_restart()           if((TWSR & 0xF8) != TW_REP_START)
                                            trata_erro();
#define teste_envio_end_leitura()       if((TWSR & 0xF8) != TW_MR_SLA_ACK)
                                            trata_erro();
#define teste_recebe_byte_ret_nack()    if((TWSR & 0xF8) != TW_MR_DATA_NACK)
                                            trata_erro();
//-------------------------------------------------------------------------------
```

O TWI pode operar em 4 modos principais: Mestre Transmissor (MT), Mestre Receptor (MR), Escravo Transmissor (ST) e Escravo Receptor (SR). Vários desses modos podem ser empregados em uma mesma aplicação. Por exemplo, o modo MT pode ser usado para escrever dados em um EEPROM I2C e o modo MR para ler dados da mesma. Existe a possibilidade da existência de múltiplos mestres no barramento. Caso ocorra o início simultâneo de uma transmissão por dois ou mais deles, o TWI deve garantir que um deles efetue a transmissão, sem ocasionar a perda de dados.

Os 4 modos serão descritos sucintamente de acordo com as tabelas e figuras a seguir. Foram consideradas as abreviações conforme o manual do ATmega328:

S = condição de início (***S****tart condition*)

Rs = repetição da condição de início (***R****epeated* ***s****tart condition*)

R = bit de leitura (nível lógico alto no pino SDA) – ***R****ead bit*

W = bit de escrita (nível lógico baixo no pino SDA) – ***W****rite bit*

A = bit de reconhecimento (nível lógico baixo no pino SDA) – ***A**cknowledge bit*

$\overline{\mathbf{A}}$ = bit de reconhecimento negado (nível lógico alto no pino SDA) – ***N**ot **A**cknowledge bit*

Data= byte com 8 bits de **D**ados

P = condição de parada (*sto**P** condition*)

SLA = endereço do escravo (***SL**ave **A**ddress*)

Tab. 16.2 – Códigos de *status* para o modo mestre transmissor.

Código de *Status* (TWSR). Bits de *prescaler* são 0.	***Status* da via e da interface de Hardware TWI**	**Resposta do Software de Controle**					**Próxima ação do módulo TWI**
		Para/do TWDR	**Para o TWCR**				
			STA	**STO**	**TWIN**	**TWE**	
0x08	Uma condição de início foi transmitida.	Carga do SLA+W	0	0	1	X	SLA+W será transmitido e um ACK ou NÃO ACK recebido.
0x10	Uma condição de repetição de início foi transmitida.	Carga do SLA+W ou	0	0	1	X	SLA+W será transmitido e um ACK ou NÃO ACK recebido.
		SLA+R	0	0	1	X	SLA+R será transmitido. A lógica irá mudar para o modo de recepção mestre.
0x18	SLA+W foi transmitido e o ACK foi recebido	Carga do byte de dados ou nenhuma ação TWDR	0	0	1	X	O byte de dados será transmitido e um ACK ou NÃO ACK recebido.
			1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.
0x20	SLA+W foi transmitido e um NÃO ACK recebido	Carga do byte de dados ou nenhuma ação TWDR	0	0	1	X	O byte de dados será transmitido e um ACK ou NÃO ACK recebido.
			1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.
0x28	O byte de dados foi transmitido e o ACK recebido	Carga do byte de dados ou nenhuma ação TWDR	0	0	1	X	O byte de dados será transmitido e um ACK ou NÃO ACK recebido.
			1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.
0x30	O byte de dados foi transmitido e o NÃO ACK recebido	Carga do byte de dados ou nenhuma ação TWDR	0	0	1	X	O byte de dados será transmitido e um ACK ou NÃO ACK recebido.
			1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.
0x38	Perda de controle no SLA+W ou no byte de dados	nenhuma ação TWDR	0	0	1	X	A via TWI será liberada e o modo de endereçamento escravo não será ativo.
			1	0	1	X	Uma condição de início será enviada quando a via ficar livre.

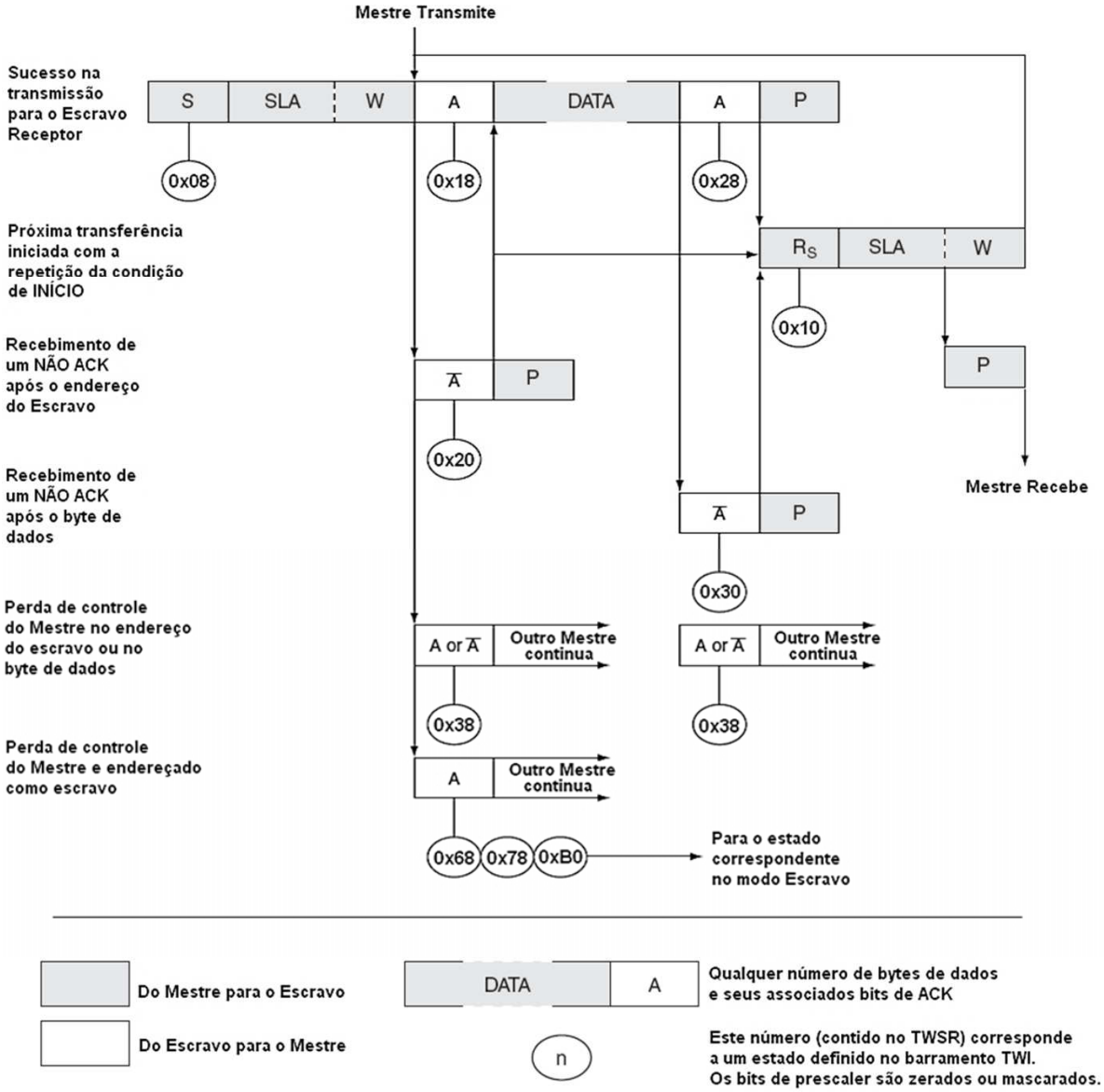

Fig. 16.7 – Formatos e estados do modo mestre transmissor.

Tab. 16.3 – Códigos de *status* para o modo mestre receptor.

Código de *Status* (TWSR). Bits de *prescaler* são 0.	***Status* da via e da interface de Hardware TWI**	**Resposta do Software de Controle**					**Próxima ação do módulo TWI**
		Para/do TWDR	**Para o TWCR**				
			STA	**STO**	**TWINT**	**TWEA**	
0x08	Uma condição de início foi transmitida.	carga do SLA+R	0	0	1	X	SLA+R será transmitido e um ACK ou NÃO ACK recebido.
0x10	Uma condição de repetição de início foi transmitida.	carga do SLA+R ou	0	0	1	X	SLA+R será transmitido e um ACK ou NÃO ACK recebido.
		SLA+W	0	0	1	X	SLA+W será transmitido. A lógica irá mudar para o modo de transmissão mestre.
0x38	Perda de controle no SLA+R ou no bit NÃO ACK	nenhuma ação TWDR	0	0	1	X	A via TWI será liberada e o modo de endereçamento escravo não será ativo.
			1	0	1	X	Uma condição de início será enviada quando a via ficar livre.
0x40	SLA+R foi transmitido e um ACK recebido	nenhuma ação TWDR	0	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			0	0	1	1	O byte de dados será recebido e um ACK retornado.
0x48	SLA+R foi transmitido e um NÃO ACK recebido	nenhuma ação TWDR	1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.
0x50	O byte de dados foi recebido e um ACK retornado.	Leitura do byte de dados	0	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			0	0	1	1	O byte de dados será recebido e um ACK retornado.
0x58	O byte de dados foi recebido e um NÃO ACK retornado.	Leitura do byte de dados	1	0	1	X	Repetição da condição de início.
			0	1	1	X	A parada será transmitida e o bit TWSTO será limpo.
			1	1	1	X	Uma parada seguida de um início serão transmitidos e o bit TWSTO será limpo.

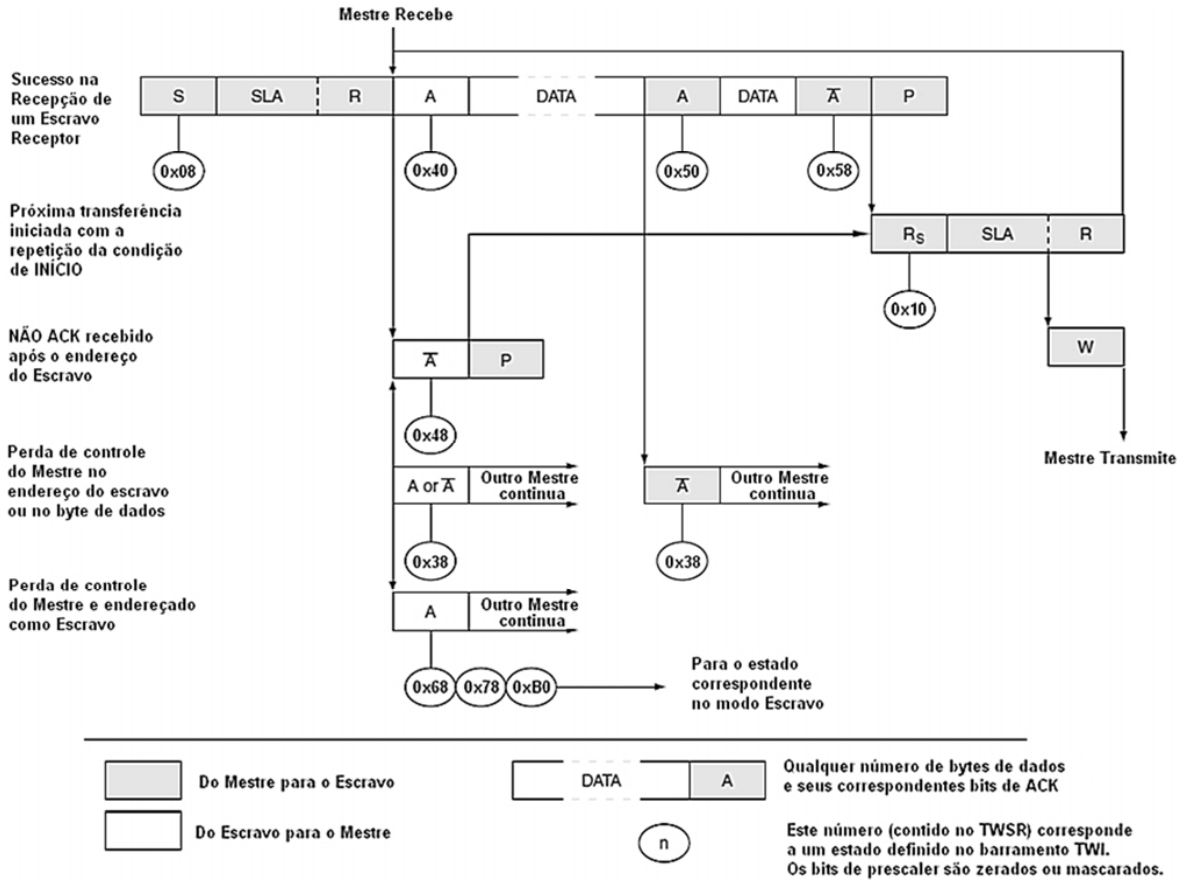

Fig. 16.8 – Formatos e estados do modo mestre receptor.

Tab. 16.4 – Códigos de *status* para o modo escravo receptor.

Código de *Status* (TWSR). Bits de *prescaler* são 0.	***Status* da via e da interface de Hardware TWI**	**Resposta do Software de Controle**					**Próxima ação do módulo TWI**
		Para/do TWDR	**Para o TWCR**				
			STA	**STO**	**TWINT**	**TWEA**	
0x60	O próprio endereço SLA+W foi recebido e um ACK retornado.	nenhuma ação TWDR	X	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			X	0	1	1	O byte de dados será recebido e um ACK retornado.
0x68	Perda de controle como mestre no SLA+R/W. O próprio SLA+W foi recebido e um ACK retornado.	nenhuma ação TWDR	X	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			X	0	1	1	O byte de dados será recebido e um ACK retornado.
0x70	Uma chamada geral de endereço foi recebida e um ACK retornado.	nenhuma ação TWDR	X	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			X	0	1	1	O byte de dados será recebido e um ACK retornado.
0x78	Perda de controle como mestre no SLA+R/W. A chamada geral de endereço foi recebida e um ACK retornado.	nenhuma ação TWDR	X	0	1	0	O byte de dados será recebido e um NÃO ACK retornado.
			X	0	1	1	O byte de dados será recebido e um ACK retornado.

0x80	Endereçado previamente com o próprio SLA+W. Um dado foi recebido e um ACK retornado.	Leitura do byte de dados	X X	0 0	1 1	0 1	O byte de dados será recebido e um NÃO ACK retornado. O byte de dados será recebido e um ACK retornado.
0x88	Endereçado previamente com o próprio SLA+W. Um dado foi recebido e um NÃO ACK retornado.	Leitura do byte de dados	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA (*General Call Address*). Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Uma condição de início será transmitida quando a via ficar livre. Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Uma condição de início será transmitida quando a via ficar livre.
0x90	Endereçado previamente pela chamada geral. Um dado foi recebido e um ACK retornado.	Leitura do byte de dados	X X	0 0	1 1	0 1	O byte de dados será recebido e um NÃO ACK retornado. O byte de dados será recebido e um ACK retornado.
0x98	Endereçado previamente pela chamada geral. Um dado foi recebido e um NÃO ACK retornado.	Leitura do byte de dados	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Uma condição de início será transmitida quando a via ficar livre. Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Uma condição de início será transmitida quando a via ficar livre.
0xA0	Uma parada ou repetição de início foi recebida enquanto ainda endereçado como escravo.	Nenhuma ação	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Uma condição de início será transmitida quando a via ficar livre. Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Uma condição de início será transmitida quando a via ficar livre.

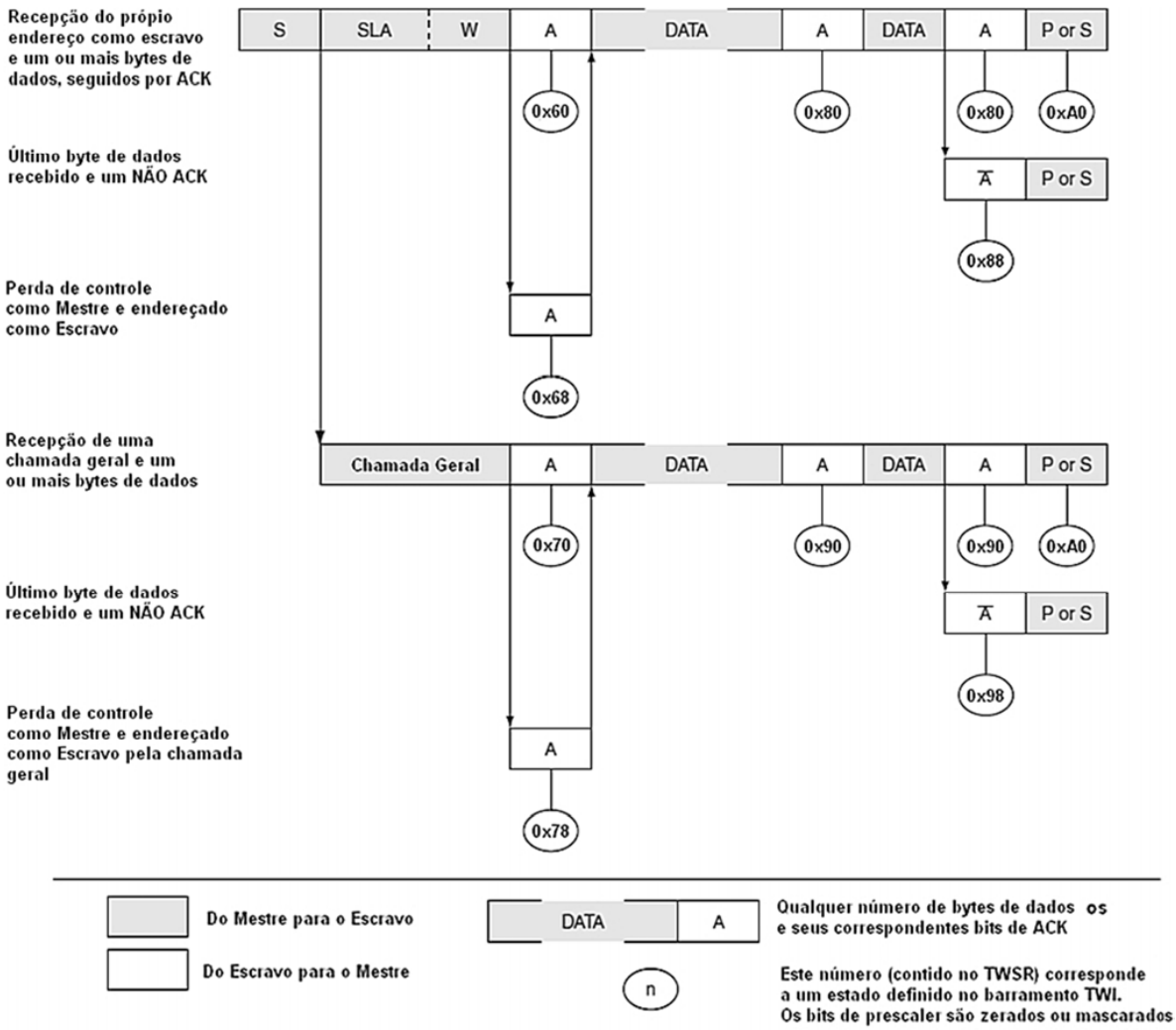

Fig. 16.9 – Formatos e estados do modo escravo receptor.

Tab. 16.5 – Códigos de *status* para o modo escravo transmissor.

Código de *Status* (TWSR). Bits de *prescaler* são 0.	***Status* da via e da interface de Hardware TWI**	**Resposta do Software de Controle**					**Próxima ação do módulo TWI**
		Para/do TWDR	**Para o TWCR**				
			STA	**STO**	**TWINT**	**TWEA**	
0xA8	O próprio endereço SLA+R foi recebido e um ACK retornado.	carrega o byte de dados	X	0	1	0	O último byte de dados será transmitido e um NÃO ACK deve ser recebido.
			X	0	1	1	O byte de dados será transmitido e um ACK deve ser recebido.
0xB0	Perda de controle como mestre no SLA+R/W. O próprio SLA+R foi recebido e um ACK retornado.	carrega o byte de dados	X	0	1	0	O último byte de dados será transmitido e um NÃO ACK deve ser recebido.
			X	0	1	1	O byte de dados será transmitido e um ACK deve ser recebido.
0xB8	O byte de dados do TWDR foi transmitido e um ACK recebido	carrega o byte de dados	X	0	1	0	O último byte de dados será transmitido e um NÃO ACK deve ser recebido.
			X	0	1	1	O byte de dados será transmitido e um ACK deve ser recebido.
0xC0	O byte de dados do TWDR foi transmitido e um ACK recebido	nenhuma ação TWDR	0	0	1	0	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA.
			0	0	1	1	Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1.
			1	0	1	0	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Uma condição de início será transmitida quando a via ficar livre.
			1	0	1	1	Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Uma condição de início será transmitida quando a via ficar livre.
0xC8	Último byte de dados do TWDR foi transmitido (TWEA =0) e um ACK foi recebido.	nenhuma ação TWDR	0	0	1	0	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA.
			0	0	1	1	Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1.
			1	0	1	0	Mudança para o modo não endereçado escravo. Não reconhecimento do próprio SLA ou GCA. Uma condição de início será transmitida quando a via ficar livre.
			1	0	1	1	Mudança para o modo não endereçado escravo. O próprio SLA será reconhecido. GCA será reconhecido se TWGCE = 1. Uma condição de início será transmitida quando a via ficar livre.

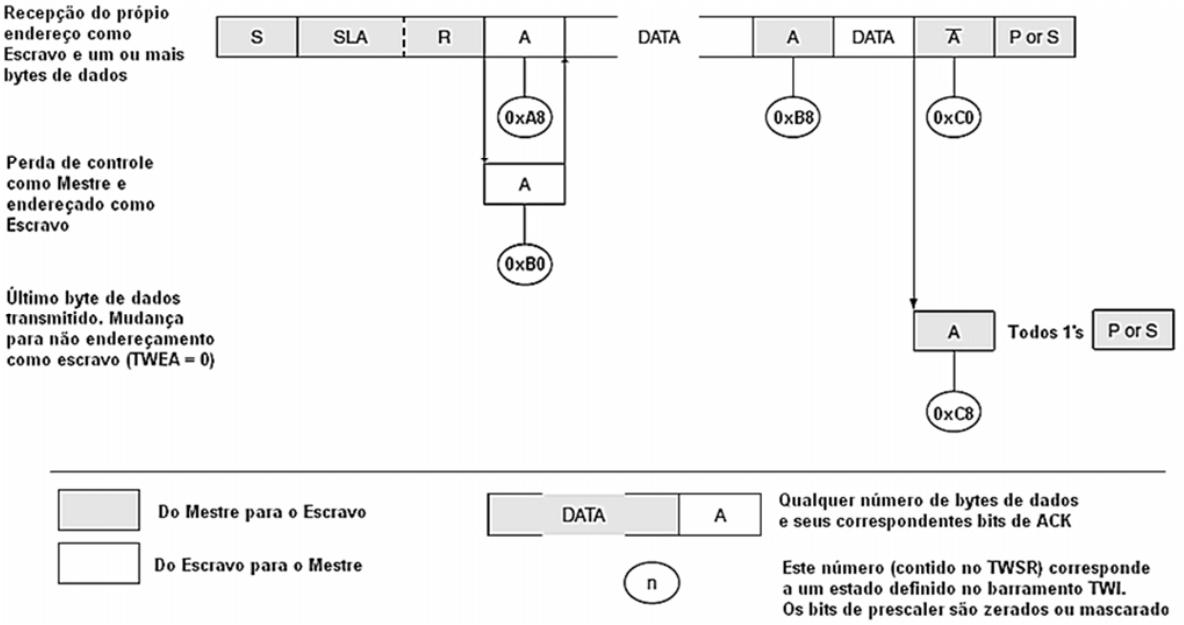

Fig. 16.10 – Formatos e estados do modo escravo transmissor.

16.4 RELÓGIO DE TEMPO REAL DS1307

O circuito integrado DS1307 é um relógio de tempo real de baixo consumo com calendário completo até o ano de 2100, que emprega o protocolo I2C. Para sua operação, é necessário o emprego de um cristal externo de 32,768 kHz e uma bateria que garante sua operação na falta da tensão de alimentação. O CI fará a divisão adequada (por 2^{15}) para obter a base de tempo de 1 s. Na fig. 16.11, é apresentado um módulo comercial com o DS1307, pronto para o uso.

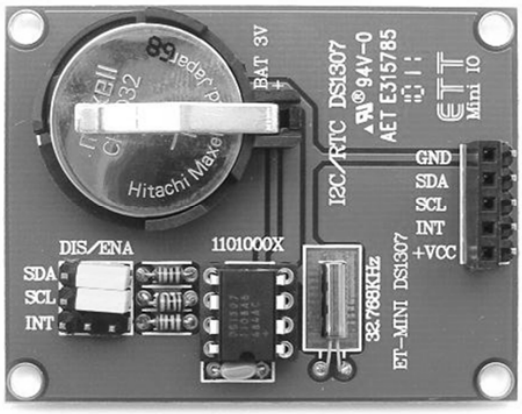

Fig. 16.11 – Módulo com o DS1307 para uso com microcontroladores (www.futurlec.com).

O mapa da memória do DS1307 é apresentado na fig. 16.12. Os dados na sua maioria empregam a codificação BCD, permitindo a leitura direta dos dois dígitos do tempo (4 LSB e 4 MSB). No endereço 0x07, é possível habilitar uma onda quadrada no pino SOUT (dreno aberto) com frequência de 1 Hz, 4096 Hz, 8192 Hz ou 32768 Hz. O bit SQWE habilita esse sinal e os bits RS0:RS1 a sua frequência de saída.

		Bit	7	6	5	4	3	2	1	0	
SEGUNDOS	0x00		CH	10.segundos			segundos				00 - 59
MINUTOS	0x01		0	10.minutos			minutos				00 - 59
HORAS	0x02		0	12 / 24	10.h A/P	10.h	horas				01 - 12 00 - 23
DIA	0x03		0	0	0	0	0	dia			1 - 7
DATA	0x04		0	0	10.data		data				01-28/29 01-30/31
MÊS	0x05		0	0	0	10. mês	mês				01 - 12
ANO	0x06		10.ano				ano				00 - 99
CONTROLES	0x07		OUT	0	0	SQWE	0	0	RS1	RS0	
RAM 56 x 8	0x08 0x3F										

Fig. 16.12 – Mapa de memória do DS1307.

Na fig. 16.13, é apresentado o formato do protocolo I2C para escrita e leitura do DS1307 conforme o manual do fabricante. Após o bit de início, o endereço do componente vem acompanhado de um bit indicando uma operação de escrita ou leitura.

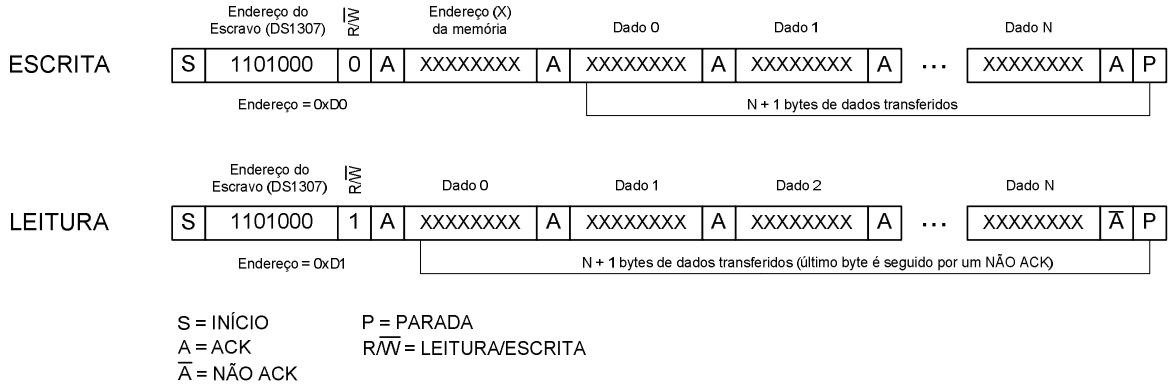

Fig. 16.13 - Formato do protocolo de comunicação I2C para o DS1307.

Um detalhe importante é que, em uma operação de leitura, é necessário primeiro realizar uma operação de escrita para atualizar o endereço de início da leitura. Na fig. 16.14, é apresentado o formato do protocolo para a leitura e escrita de um único byte de dados no DS1307.

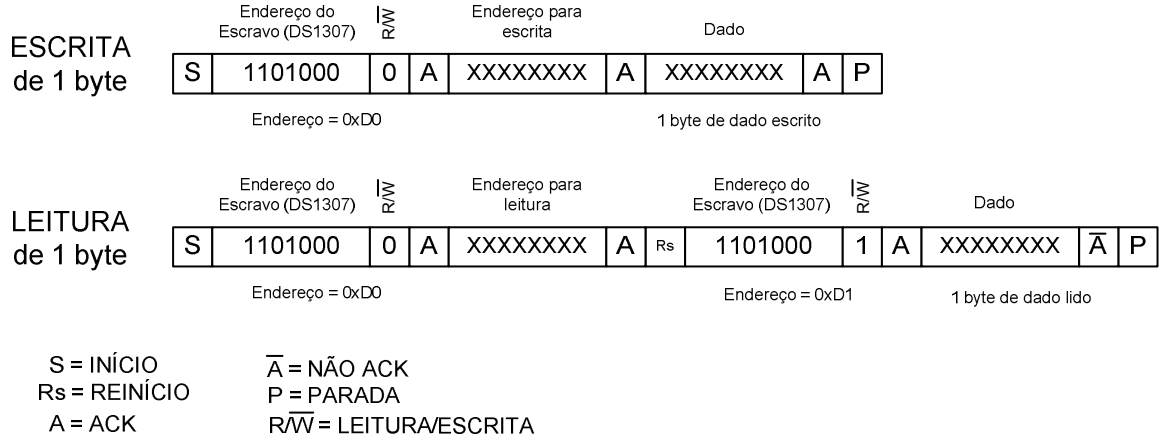

Fig. 16.14 - Formato do protocolo de comunicação I2C para a escrita e leitura de um único byte de dados no DS1307.

Abaixo, são apresentadas sub-rotinas exemplo para a escrita e leitura de 1 byte no DS1307 (suas definições foram apresentadas anteriormente). Para o emprego das sub-rotinas com outros CIs, basta trocar o endereço para a comunicação. As sub-rotinas enviam e recebem apenas 1 byte de informação. O protocolo I2C, entretanto, permite o envio e recepção de vários bytes, sem a necessidade de novos *start* bits e do endereço do componente entre os bytes.

```
//-------------------------------------------------------------------------------
//Sub-rotina para ler apenas um byte do barramento I2C do DS1307
//-------------------------------------------------------------------------------
unsigned char le_RTC(unsigned char endereco)
{
     start_bit();
     espera_envio();
     teste_envio_start();

     TWDR = 0xD0;    //carrega o endereço para acesso do DS1307 (bit0 = 0, escrita)
                     //para outro CI basta trocar este endereço
     envia_byte();
     espera_envio();
     teste_envio_end_escrita();

     TWDR = endereco;     //ajuste do ponteiro de endereço para a leitura do DS1307

     envia_byte();
     espera_envio();
     teste_envio_dado();

     start_bit();    //reinício
     espera_envio();
     teste_envio_restart();

     TWDR = 0xD1;    //carrega o endereço para acesso do DS1307 (bit0 = 1 é leitura)
                     //automaticamente o ATmega chaveia para o estado de recepção
     envia_byte();
     espera_envio();
     teste_envio_end_leitura();

     recebe_byte_ret_nack();   //só lê um byte, por isso retorna um NACK
     espera_recebimento();
     teste_recebe_byte_ret_nack();

     stop_bit();
     return TWDR;    //retorna byte recebido
}
//-------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------
//Sub-rotina para escrever apenas um byte no barramento I2C do DS1307
//-------------------------------------------------------------------------------
void escreve_RTC(unsigned char dado_i2c, unsigned char endereco)
{
     start_bit();
     espera_envio();
     teste_envio_start();

     TWDR = 0xD0;    //carrega o endereço para acesso do DS1307 (bit0 = 0 é escrita)
                     //para outro CI basta trocar este endereço
     envia_byte();
     espera_envio();
     teste_envio_end_escrita();

     TWDR = endereco;     //carrega o endereço para escrita do dado no DS1307

     envia_byte();
     espera_envio();
     teste_envio_dado();

     TWDR = dado_i2c;     //carrega o dado para escrita no endereço especificado

     envia_byte();
     espera_envio();
     teste_envio_dado();

     stop_bit();
}
//-------------------------------------------------------------------------------
```

As sub-rotinas anteriores seguem um fluxo sequencial, ou seja, o programa executa um passo por vez na sequência de escrita, o que não permite que o microcontrolador execute outra ação durante esse trabalho. Uma abordagem mais eficiente seria utilizar dentro da interrupção do TWI um programa funcionando como uma máquina de estados. Dessa forma, as ações do TWI não precisam seguir um fluxo sequencial e permitem que o programa principal possa executar outras atividades em paralelo. Na fig. 16.15, é apresentado o algoritmo no formato de máquina de estados para a escrita e leitura de um único byte no DS1307. O algoritmo prevê a correção de erros durante a execução do protocolo I2C com a repetição de um comando mal sucedido. Todo o algoritmo é feito dentro da sub-rotina de interrupção do módulo TWI.

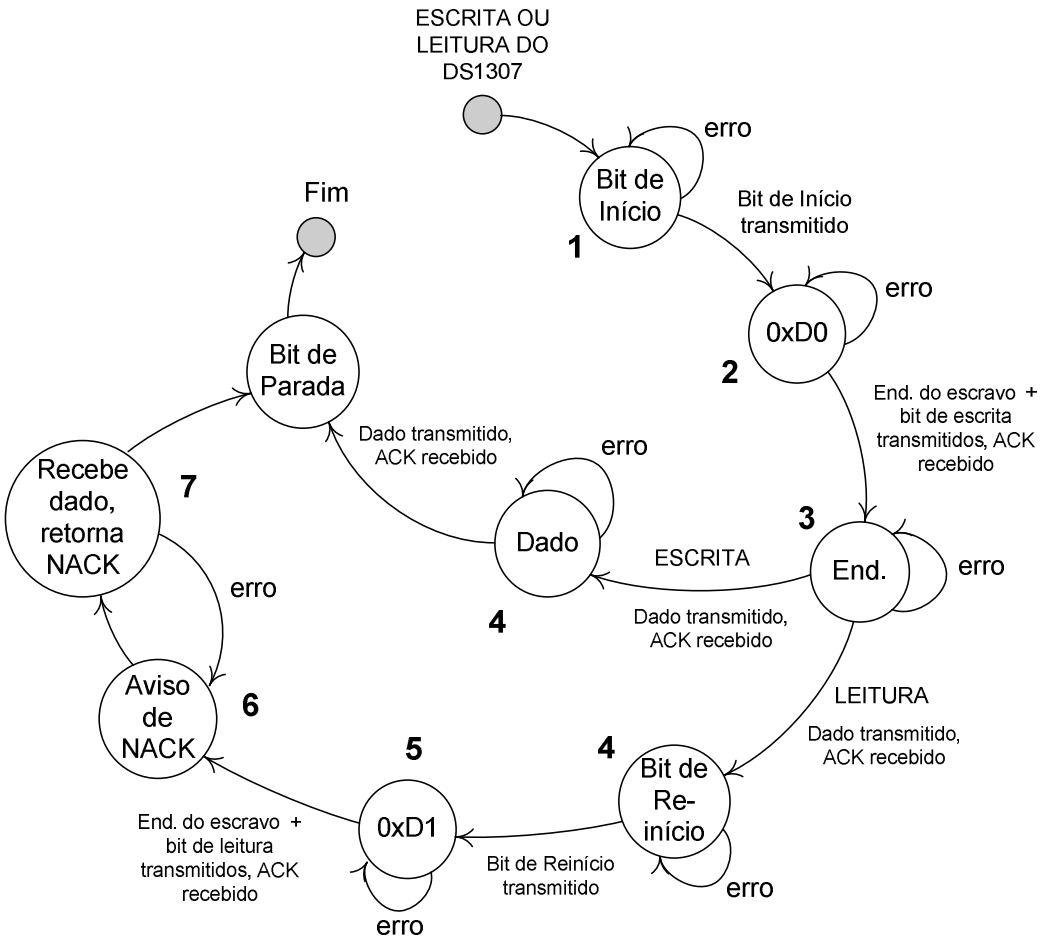

Fig. 16.15 - Máquina de estado representado o algoritmo para a escrita e leitura de um único byte no DS1307.

Além da escrita e leitura do DS1307, o microcontrolador deve ser capaz de ajustar e apresentar o horário. Na fig. 16.16, é apresentado um circuito para realizar essa tarefa. No circuito, empregam-se dois botões para o ajuste do tempo e um LCD 16×2. Na sequência, é apresentado o programa de controle do sistema, testado com o Arduino. É possível acompanhar o algoritmo da sub-rotina de interrupção do TWI na fig. 16.15. Os erros são tratados em conjunto no final do código, mas a lógica não se altera. Foi utilizado um contador (decrescente) que permite no máximo 256 tentativas de correções de erro, impedindo o travamento das atividades do TWI em algum estágio da correção de um possível erro.

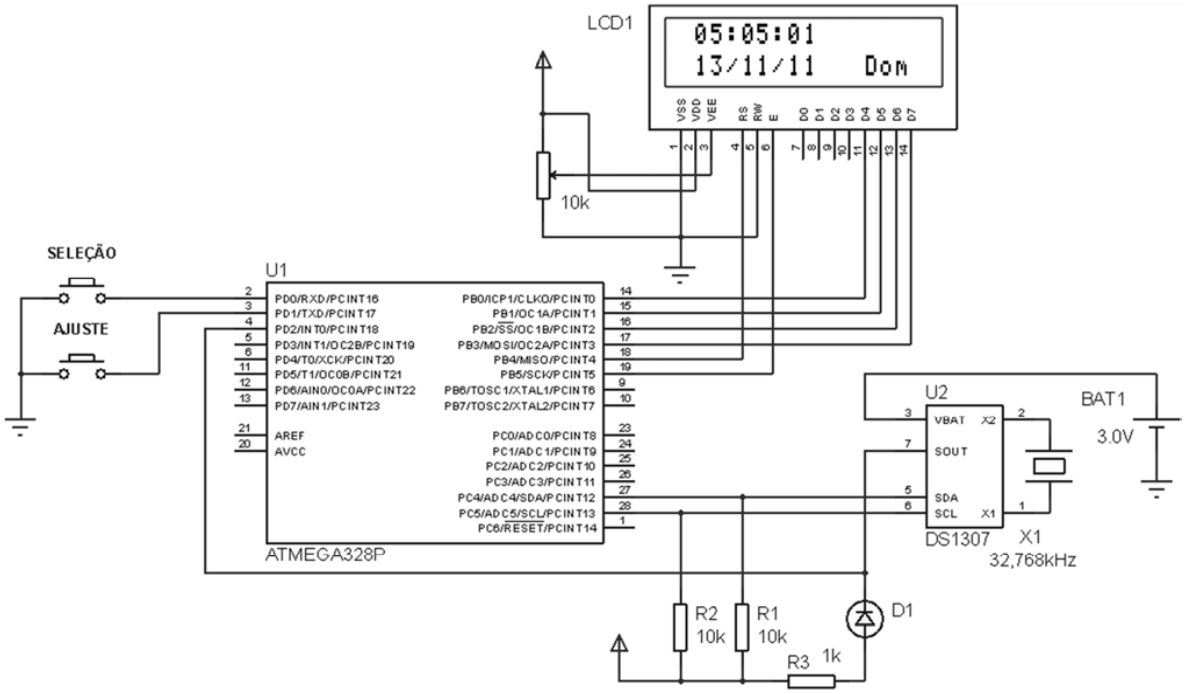

Fig. 16.16 - Circuito para o uso do DS1307.

def.principais.h (arquivo de cabeçalho do programa principal)

```
#ifndef _DEF_PRINCIPAIS_H
#define _DEF_PRINCIPAIS_H

#define F_CPU 16000000UL  //define a frequência do microcontrolador - 16MHz

#include <avr/io.h>       //definições do componente especificado
#include <util/delay.h>   //biblioteca para o uso das rotinas de delay
#include <avr/pgmspace.h> //para o uso do PROGMEM, gravação de dados na memória flash
#include <avr/interrupt.h>//para uso das interrupções
#include <util/twi.h>     //para uso das definições do TWI

//Definições de macros para o trabalho com bits
#define   set_bit(y,bit)  (y|=(1<<bit))//coloca em 1 o bit x da variável Y
#define   clr_bit(y,bit)  (y&=~(1<<bit))//coloca em 0 o bit x da variável Y
#define   cpl_bit(y,bit)  (y^=(1<<bit))//troca o estado lógico do bit x da variável Y
#define   tst_bit(y,bit)  (y&(1<<bit))//retorna 0 ou 1 conforme leitura do bit

//botões para ajuste do tempo
#define SELECAO PD0
#define AJUSTE  PD1

#endif
```

RTC_DS1307.c (programa principal)

```
//------------------------------------------------------------------------------ //
//        DS1307 - RTC com LCD 16 x 2 utilizando I2C                              //
//------------------------------------------------------------------------------//
#include "def_principais.h"
#include "LCD.h"
#include "DS1307.h"

extern unsigned char flag_pontos, cont;

int main()
{
     DDRB = 0xFF; //LCD
     DDRD = 0x00; //botões
     PORTD = 0x03;//pull-ups habilitados

     //TWI
     DDRC = (1<<4)|(1<<5);

     inic_TWI();
     sei();

     inic_LCD_4bits();

     escreve_DS1307(0x00, 0x00);
     escreve_DS1307(0x07,0b00010000);//habilita 1Hz no pino SOUT do DS1307

     //-----------------------------------------------------------------------------
     while(1)
     {
          ler_convert_tudo(0x00);   //lê toda a memória de tempo do RTC
          mostra_tempo();
          mostra_pontos(flag_pontos);//flag indica se deve ligar os pontos e traços
          _delay_ms(200);

          if(!tst_bit(PIND,SELECAO))
          {
               flag_pontos = 0;//avisa para não imprimir mais os pontos e traços
               cont++;//conta o nr. de vezes que o botão SELECAO foi pressionado

               if (cont>7)
               {    cont = 0;
                    flag_pontos = 1;//habilita novamente os pontos e traços
               }
               alerta_display(cont);//coloca a seta no local para ajuste do tempo
               while(!tst_bit(PIND,SELECAO));//aguarda soltar SELECAO
          }
          //-------------------------------------------------------------------
          if (flag_pontos==0)
          {
               if(!tst_bit(PIND,AJUSTE))
                    ajusta_tempo2(cont);//ajusta RTC ou ajusta alarme
          }//if !flag
          //-------------------------------------------------------------------
     }//while forever
     //-----------------------------------------------------------------------------
}
```

DS1307.h (arquivo de cabeçalho do DS1307.c)

```
#ifndef _DS1307_H
#define _DS1307_H

#include "def_principais.h"

#define start_bit()       TWCR |= (1<<TWSTA)
#define stop_bit()        TWCR |= (1<<TWSTO)
#define clr_start_bit()   TWCR &= ~(1<<TWSTA)

//TWI
void inic_TWI();
void escreve_DS1307(unsigned char end_escrita, unsigned char dado);
unsigned char le_DS1307(unsigned char end_leitura);
ISR(TWI_vect);

//trabalho com o LCD para mostrar o tempo
void converte2BCD(unsigned char dado, unsigned char endereco);
void ler_convert_tudo(unsigned char ende);
void ajusta_tempo(unsigned char end, unsigned char limite,unsigned char ini_nr1);
void ajusta_tempo2(unsigned char qual);
void mostra_tempo();
void mostra_pontos(unsigned char desliga);
void alerta_display(unsigned char qual);

#endif
```

DS1307.c (aqui estão todas as funções para o trabalho[40] com o DS1307, incluindo o LCD, que é a maior parte do código)

```
#include "DS1307.h"
#include "LCD.h"

volatile unsigned char escrita, oper_TWI_concl, end_escr_leit, dado_escr, dado_leit,
                                                                    passo, cont_max_erro;
unsigned char tempo[7], segs[14], aux_disp[4], flag_pontos = 1, cont = 0;
char escr[4];
//-------------------------------------------------------------------------------
void inic_TWI() //SCL = 100 kHz (limite do DS1307) com F_CPU = 16 MHz
{
     //Ajuste da frequência de trabalho - SCL = F_CPU/(16+2.TWBR.Prescaler)
     TWBR = 18;
     TWSR |= 0x01;//prescaler =  4;
     TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWIE); //habilita o TWI e a interrupção
}
//-------------------------------------------------------------------------------
void escreve_DS1307(unsigned char end_escrita, unsigned char dado)
{
     //passa variáveis da função para as variáveis globais alteradas na ISR
     escrita = 1;         //1 para escrita, 0 para leitura
     end_escr_leit = end_escrita;
     dado_escr = dado;
```

[40] A parte de tratamento de erros na comunicação não foi testada. A análise exige o uso de equipamento de medição específico.

```
     oper_TWI_concl=0;    //trava do sistema até a conclusão da transmissão
     start_bit();         //envia o Start bit. Passo (1)
     passo = 1;
     cont_max_erro = 0xFF;

     while(oper_TWI_concl==0);/*se for crítica a espera, o programa principal pode
                                                              gerenciar esta operação*/
}
//---------------------------------------------------------------------------
unsigned char le_DS1307(unsigned char end_leitura)
{
     //passa variáveis da função para as variáveis globais alteradas na ISR
     escrita = 0;         //1 para escrita 0 para leitura
     end_escr_leit = end_leitura;

     oper_TWI_concl=0;    //trava do sistema até a conclusão da transmissão

     start_bit();         //envia o Start bit. Passo (1)
     passo = 1;
     cont_max_erro=0xFF;

     while(oper_TWI_concl==0); /*se for crítica a espera, o programa principal pode
                                                              gerenciar esta operação*/
     return dado_leit;
}
//---------------------------------------------------------------------------
ISR(TWI_vect)//Rotina de interrupção da TWI
{
     static unsigned char fim_escrita;

     switch (TWSR & 0xF8) //lê o código de resultado do TWI e executa a próxima ação
     {
          /*LEITURA E ESCRITA
          PASSO 2 <start condition transmitted>. Passo (1) concluído, executa passo (2)*/
          case (TW_START):
                    TWDR = 0xD0;    //envia endereço do dispositivo e o bit de escrita
                    clr_start_bit();//limpa o start bit
                    passo = 2;
                    break;

          /*LEITURA E ESCRITA
          PASSO 3 <SLA+W transmitted, ACK received>. Passo (2) concluído, executa passo (3)*/
          case (TW_MT_SLA_ACK):
                    TWDR = end_escr_leit;//envia o endereço de escrita ou leitura
                    passo=3;
                    fim_escrita=0;  //inicializa variável para uso na escrita, PASSO 4
                    break;

          /*LEITURA E ESCRITA
          PASSO 4 <data transmitted, ACK received>. Passo (3) concluído, executa passo (4).
          Passo (4) concluído, executa passo (5) (só na escrita). O passo (4) para uma leitura é o
                                                                                  reinício*/
          case (TW_MT_DATA_ACK):
                    if(fim_escrita) //se o passo (4) foi concluído executa o (5), escrita
                    {
                         stop_bit();
                         oper_TWI_concl = 1; //avisa que operação foi concluída
                         break;
                    }
                    //envia um único dado quando for operação de escrita e depois um stop_bit()
                    if(escrita)
                    {
                         TWDR = dado_escr;//dado para escrita no endereço de escrita
                         fim_escrita = 1;//avisa que é o último dado a ser escrito
                    }
```

```
                else
                     start_bit();//envia reinício (só para operação de leitura)

                passo = 4;
                break;

        /*LEITURA
        PASSO 5 <repeated start condition transmitted>. Passo (4) concluído, executa o (5)*/
        case (TW_REP_START):
                TWDR  = 0xD1;        //Envia endereço e read bit (4)
                clr_start_bit();     //limpa start bit
                passo = 5;
                break;

        /*LEITURA
        PASSO 6 <SLA+R transmitted, ACK received>. Passo (5) concluído, prepara para a
                                                                        execução do NACK*/
        case (TW_MR_SLA_ACK) :
                TWCR &=~(1<<TWEA);   //enviará um NACK após a recepção do dado
                passo=6;
                break;

        /*LEITURA
        PASSO 7 <data received, NACK returned>. Passo (6) concluído, NACK recebido, executa
                                                                            passo (7)*/
        case (TW_MR_DATA_NACK):
                dado_leit = TWDR;    //dado lido
                stop_bit();
                oper_TWI_concl = 1; //avisa que operação foi concluída
                break;

        /*TRATAMENTO DOS POSSÍVEIS ERROS
         Quando um erro acontece a operação errada é repetida até funcionar ou até
         que o contador para o número máximo de tentavas chegue a zero*/

        default:
                cont_max_erro--;

                switch(passo)
                {
                     case(1): start_bit(); break;
                     case(2): TWDR = 0xD0; break;
                     case(3): TWDR = end_escr_leit; break;
                     case(4):
                          if(escrita)
                               TWDR = dado_escr;
                          else
                               start_bit();//reinício
                          break;
                     case(5): TWDR  = 0xD1;         break;
                     case(6): TWCR &=~(1<<TWEA); break;
                }
                /*para saber se houve estouro na contagem ou insucesso na
                                  correção dos erros, basta ler cont_max_erro.*/
                if(cont_max_erro==0)
                {
                     stop_bit();
                     oper_TWI_concl = 1; //libera o sistema
                }

     }
     TWCR |= (1<<TWINT); //limpa flag de interrupção
}
```

```
//---------------------------------------------------------------------------------------
//Rotinas complementares para o trabalho com o LCD
//---------------------------------------------------------------------------------------
void converte2BCD(unsigned char dado, unsigned char endereco)
{
   endereco = endereco + endereco;       //2 x endereço, são dois digitos por dado
   segs[endereco] = dado & 0b00001111;   //BCD - primeiro digito - direita
   segs[endereco+1] = (dado & 0b11110000)>>4;  //BCD - segundo digito - esquerda

}//rotina para ler o dado do RTC e converter para o valor de segmento correto
//---------------------------------------------------------------------------------------
void ler_convert_tudo(unsigned char ende)//lê todo o tempo do RTC e converte para o LCD
{
     unsigned char j;

     for (j=0;j<7;j++)//leitura e conversão do tempo do RTC
     {
          tempo[j] = le_DS1307(j+ende);//salva todos os dados do relógio
          converte2BCD(tempo[j], j);
     }/*j -> 0x00 segundos, 0x01 minutos, 0x02 horas, 0x03 dia semana, 0x04 dia mês,
                                                                  0x05 mês, 0x06 ano.*/
}
//---------------------------------------------------------------------------------------
//endereco de ajuste e seu limite
void ajusta_tempo(unsigned char end, unsigned char limite,unsigned char ini_nr1)
{
     //ini_nr1 -> flag para avisar se o nr. começa em 1 (dia e mês)
     unsigned char aux, conta=0;

     aux = segs[end+end] + 10*segs[end+end+1];//converte 2 BCDs em 1 byte

     if (aux<limite)
          aux++; //incremento no tempo
     else
          aux = 0;

     //converte aux para  2 BCD
     if (aux<10)
          tempo[end] = aux;
     else
     {    do
          {    aux = aux - 10;
               conta++;
          }while(aux>9);
     }

     if(ini_nr1==1)
     {
          if (aux==0 && conta==0)//se for dia e mês não pode mostrar o zero
               aux = 1;
     }

     tempo[end] = aux + (conta<<4);//monta o tempo ajustado – 2 BCDs
     escreve_DS1307(end,tempo[end]);//altera o valor no RTC
}
//---------------------------------------------------------------------------------------
// Rotina que chama o ajuste_tempo conforme seleção de botao
//---------------------------------------------------------------------------------------
void ajusta_tempo2(unsigned char qual)
{
     switch(qual)//qual ajuste será efetuado
     {
        case 0: break;
        case 1: ajusta_tempo(0x02, 23,0); break;//ajusta as horas (0x02) e seu limite (23)
```

```
        case 2: ajusta_tempo(0x01, 59,0);   break;//ajuste dos minutos
        case 3: escreve_DS1307(0x00, 0x00); break;//só zera os segundos
        case 4: ajusta_tempo(0x04, 31,1);   break;//ajuste dos dias
        case 5: ajusta_tempo(0x05, 12,1);   break;//ajuste dos meses
        case 6: ajusta_tempo(0x06, 99,0);   break;//ajuste do ano
        case 7: ajusta_tempo(0x03, 7,1);    break;
      }//switch(count)
}
//---------------------------------------------------------------------------
void mostra_tempo()//coloca os dados do RTC no formato padrão no LCD
{
     //mostrar tempo no LCD -> 1a linha horas
     cmd_LCD(0x81,0);
     cmd_LCD(segs[5]+48,1);    //horas
     cmd_LCD(segs[4]+48,1);
     cmd_LCD(0x84,0);
     cmd_LCD(segs[3]+48,1);    //minutos
     cmd_LCD(segs[2]+48,1);
     cmd_LCD(0x87,0);          //espaço em branco
     cmd_LCD(segs[1]+48,1);    //segundos
     cmd_LCD(segs[0]+48,1);
     //mostrar tempo no LCD -> 2 linha calendário
     cmd_LCD(0xC1,0);
     cmd_LCD(segs[9]+48,1);    //dia do mês
     cmd_LCD(segs[8]+48,1);
     cmd_LCD(0xC4,0);          //espaço em branco
     cmd_LCD(segs[11]+48,1);   //mês
     cmd_LCD(segs[10]+48,1);
     cmd_LCD(0xC7,0);          //espaço em branco
     cmd_LCD(segs[13]+48,1);   //ano
     cmd_LCD(segs[12]+48,1);

     switch(segs[6])//dia da semana
     {
          case 1:  escr[0] = 'D'; escr[1] = 'o'; escr[2] = 'm'; break;
          case 2:  escr[0] = 'S'; escr[1] = 'e'; escr[2] = 'g'; break;
          case 3:  escr[0] = 'T'; escr[1] = 'e'; escr[2] = 'r'; break;
          case 4:  escr[0] = 'Q'; escr[1] = 'u'; escr[2] = 'a'; break;
          case 5:  escr[0] = 'Q'; escr[1] = 'u'; escr[2] = 'i'; break;
          case 6:  escr[0] = 'S'; escr[1] = 'e'; escr[2] = 'x'; break;
          case 7:  escr[0] = 'S'; escr[1] = 'a'; escr[2] = 'b'; break;
          default: escr[0] = 'x'; escr[1] = 'x'; escr[2] = 'x'; break;
     }//switch

     cmd_LCD(0xCC,0);
     escreve_LCD(escr);//mostra dia da semana
}
//---------------------------------------------------------------------------
void mostra_pontos(unsigned char desliga)//coloca pontos e barra no display e limpa
{                                        //algumas posições
     if (desliga==1)
     {
          cmd_LCD(0x80,0);cmd_LCD(0x20,1);//limpa antes da hora
          cmd_LCD(0x83,0);cmd_LCD(0x3A,1);//dois pontos depois das horas
          cmd_LCD(0x86,0);cmd_LCD(0x3A,1);//dois pontos depois dos segundos
          cmd_LCD(0xC0,0);cmd_LCD(0x20,1);//limpa antes do dia do mês
          cmd_LCD(0xC3,0);cmd_LCD(0x2F,1);//barra inclinada depois do dia do mês
          cmd_LCD(0xC6,0);cmd_LCD(0x2F,1);//barra inclinada antes do ano
          cmd_LCD(0xCB,0);cmd_LCD(0x20,1);//limpa antes do dia da semana
     }
}
//---------------------------------------------------------------------------
```

```
void alerta_display(unsigned char qual)//seta no display para o ajuste
{
     switch(qual)
     {
          case 1: cmd_LCD(0x80,0); break;     //horas
          case 2: cmd_LCD(0x83,0); break;     //minutos
          case 3: cmd_LCD(0x86,0); break;     //segundos
          case 4: cmd_LCD(0xC0,0); break;     //dia mês
          case 5: cmd_LCD(0xC3,0); break;     //mês
          case 6: cmd_LCD(0xC6,0); break;     //ano
          case 7: cmd_LCD(0xCB,0); break;     //dia da semana
     }//switch(count)

     //este if é só para corrigir a seta na última pressionada do botao SELECAO, após o dia.
     if(cont!=0)
          cmd_LCD(0x7E,1);//-> seta para ajuste
}
//---------------------------------------------------------------------------------
```

LCD.h e LCD.c (vistos no capítulo 5 – com a definição #define conv_ascii 0)

Um resultado prático para o programa supracitado é apresentado na fig. 16.17.

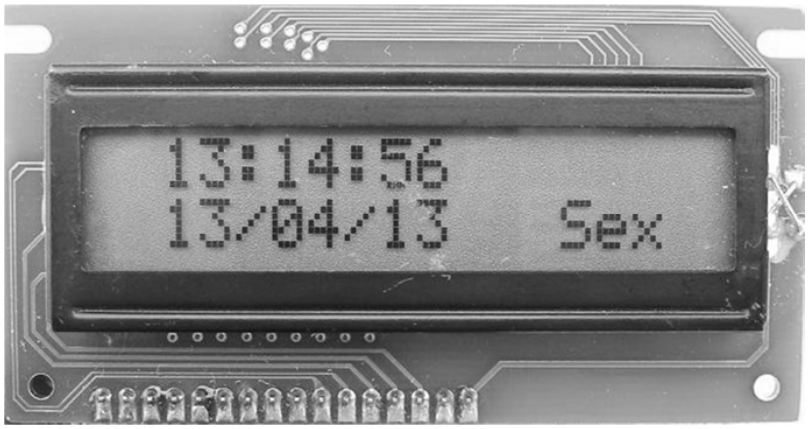

Fig. 16.17 - Resultado prático para teste do programa de controle do TWI empregando máquina de estados e o DS1307.

Exercícios:

16.1 – Elaborar um programa para ler e escrever em um DS1307 (*Real Time Clock*) empregando o módulo TWI do ATmega328, conforme circuito da fig. 16.16. Altere o programa apresentado para que ele não utilize a interrupção do TWI e faça a atualização do LCD a cada 1 segundo utilizando o sinal proveniente do pino SOUT do DS1307.

16.2 – Baseado no programa apresentado anteriormente, altere as rotinas de trabalho com o DS1307 para a leitura e escrita sequencial de vários bytes.
