

19. CONVERSOR ANALÓGICO-DIGITAL - ADC

Para o controle de variáveis externas por um sistema digital é necessária a interpretação de grandezas analógicas. Para tal, o emprego de conversores analógico-digitais torna-se imprescindível. Atualmente, os microcontroladores incorporam esses tipos de conversores, o que torna fácil o projeto de sistemas que necessitam ler variáveis analógicas. Neste capítulo, o conversor analógico-digital do ATmega328 é apresentado, juntamente com uma técnica para a leitura de um teclado matricial, o sensor de temperatura LM35, a sobreamostragem para aumentar a resolução do ADC e o filtro de média móvel para filtragem do sinal convertido.

O conversor AD do ATmega328 emprega o processo de aproximações sucessivas para converter um sinal analógico em digital. Suas principais características são:

- 10 bits de resolução (1024 pontos).
- Precisão de ± 2 LSBs (bits menos significativos).
- Tempo de conversão de 13 até 260 μ s.
- Até 76,9 kSPS (*kilo Samples Per Second*), 15 kSPS na resolução máxima.
- 6 canais de entrada multiplexados (+2 nos encapsulamentos TQFP e QFN/MLF).
- Faixa de tensão de entrada de 0 até VCC.
- Tensão de referência selecionável de 1,1 V.
- Modo de conversão simples ou contínua.
- Interrupção ao término da conversão.
- Eliminador de ruído para o modo *Sleep*.
- Sensor interno de temperatura com ± 10 °C de precisão.

O valor mínimo representado digitalmente é 0 V (GND) e o valor máximo corresponde à tensão do pino AREF menos 1 LSB; opcionalmente, a tensão do pino AVCC ou a tensão interna de referência de 1,1 V. O diagrama em blocos do conversor AD é ilustrado na fig. 19.1.

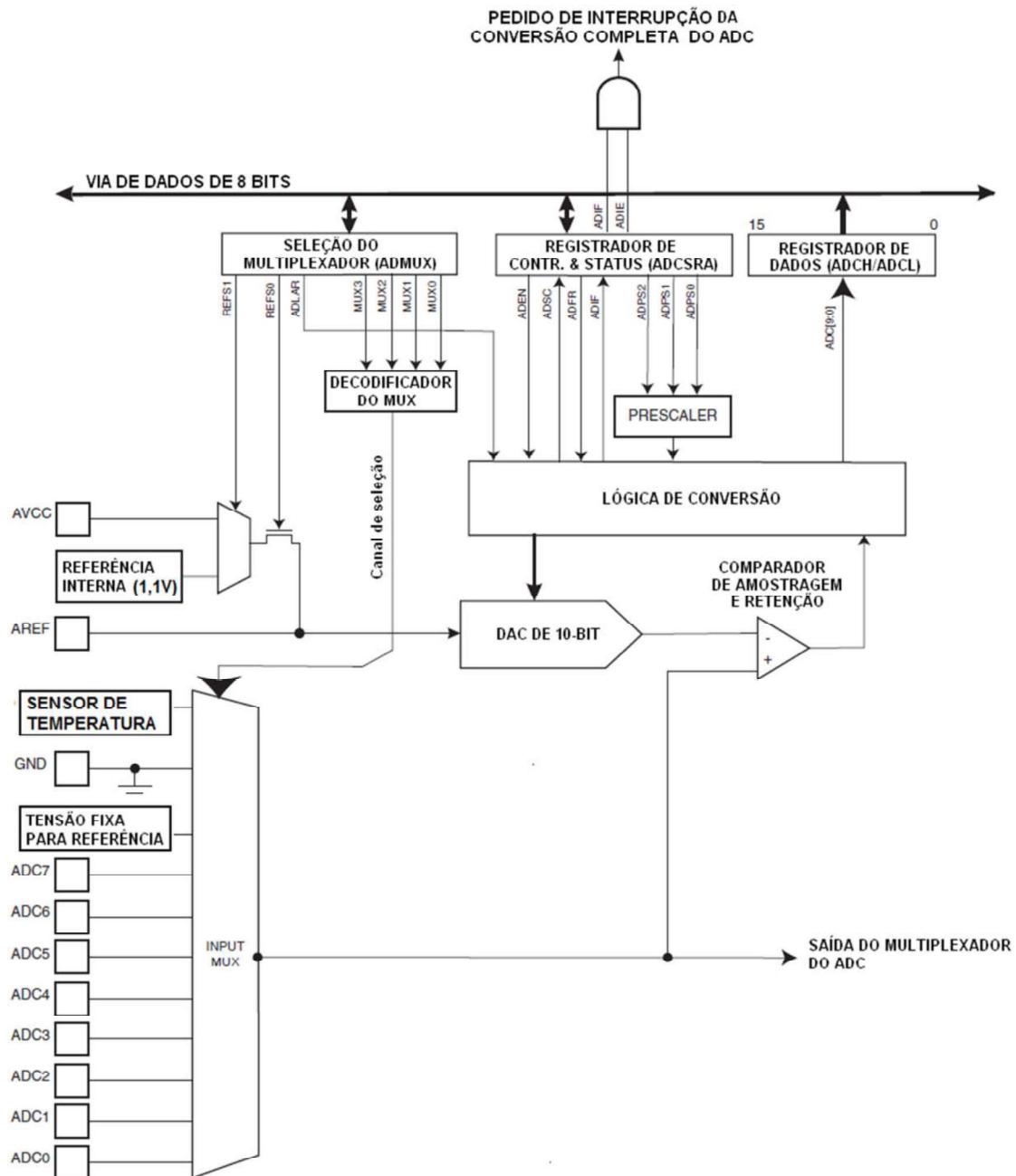


Fig. 19.1 – Diagrama do conversor AD do ATmega328.

O ADC produz um resultado de 10 bits que é escrito no registrador de 16 bits ADC (ADCH e ADCL). Por definição, o resultado é apresentado com ajuste à direita (ADCL tem os 8 LSBs). Opcionalmente, o resultado pode ter

ajuste à esquerda (ADCH com os 8 MSBs e os bits 7:6 do ADCL com os LSBs). Isso é feito, ajustando-se o bit ADLAR no registrador ADMUX. O ajuste à esquerda é interessante quando se deseja apenas 8 bits de resolução do ADC, bastando, nesse caso, apenas ler ADCH. Caso contrário, ADCL deve ser lido primeiro para garantir que o conteúdo do registrador pertence à mesma conversão. Quando a leitura do ADCL é realizada, os registradores de resultado são bloqueados para a atualização até que o ADCH seja lido. Nesse caso, o conteúdo de uma nova conversão não será salvo e a interrupção irá ocorrer mesmo com os registradores bloqueados.

Uma conversão simples é iniciada quando o bit ADSC é colocado em 1 no registrador ADCSRA. Esse bit se mantém em 1 durante a conversão e é limpo pelo hardware ao final desta. Se um canal diferente é escolhido para conversão, o ADC irá terminar a conversão corrente antes de mudar de canal. O resultado para uma conversão é dado por:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}} \quad (19.1)$$

onde V_{IN} é a tensão para conversão na entrada do pino e V_{REF} é a tensão selecionada de referência. O valor 0x000 representa a tensão zero e o valor 0x3FF representa V_{REF} menos 1 LSB.

Exemplo:

Se for empregada a tensão interna de referência de 1,1 V e a tensão de entrada for de 230 mV, qual o valor convertido pelo ADC? Qual o degrau de resolução do ADC (1 LSB)?

De acordo com a eq. 19.1, resultará:

$$ADC = \frac{0,23 \cdot 1024}{1,1} = 214 \quad \text{ou} \quad 0b11010110$$

A resolução do ADC será:

$$1LSB = \frac{V_{REF}}{1024} = \frac{1,1}{1024} = 1,07 \text{ mV}$$

No modo de conversão contínua (*Free Running*), o ADC é constantemente amostrado e os registradores de dados atualizados (bits ADTS2:0 = 0 no ADCSRB). A conversão começa escrevendo-se 1 no bit ADSC.

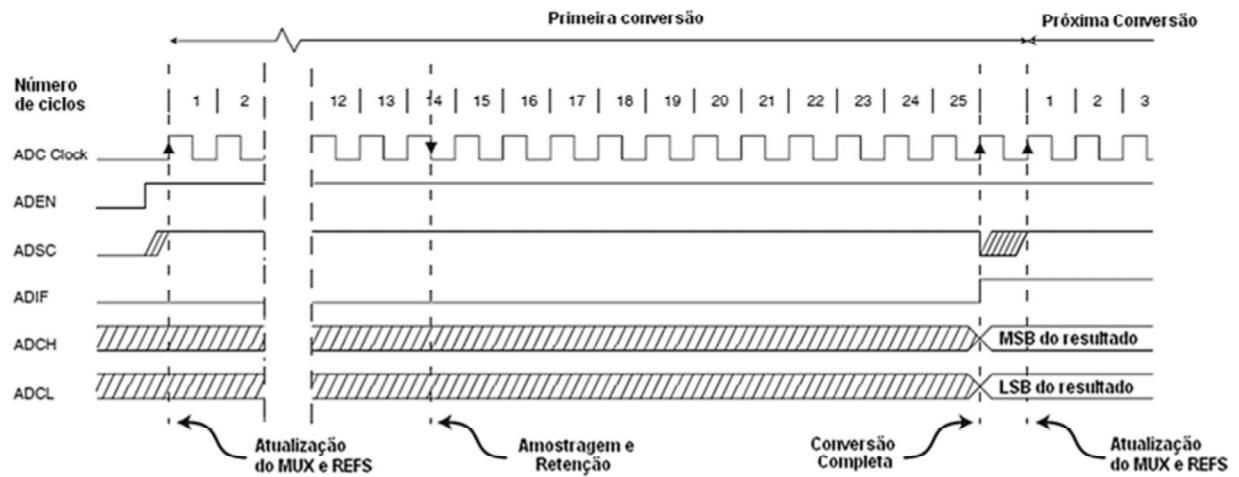
O circuito de aproximações sucessivas do ADC requer uma frequência de entrada entre 50 kHz e 200 kHz para obter a resolução máxima. Se uma resolução menor que 10 bits for desejada, uma frequência maior que 200 kHz pode ser empregada para se obter uma maior taxa de amostragem. Sinais de entrada com frequências maiores que a frequência de Nyquist ($f_{AD}/2$ - metade da frequência de amostragem) gerarão erros (*aliasing*). Eles devem ser previamente filtrados por um filtro passa baixa para eliminar o conteúdo de frequência acima da capacidade do ADC, para somente depois serem amostrados.

O ADC contém um módulo com *prescaler*, que aceita qualquer frequência de CPU acima de 100 kHz. O *prescaler* é ajustado nos bits ADPS do ADCSRA e começa a atuar quando o ADC é habilitado pelo bit ADEN. Enquanto esse bit for 1, o *prescaler* estará ativo.

Quando uma conversão simples é solicitada, colocando-se o bit ADSC em 1, a conversão inicia na próxima borda de subida do *clock* do ADC. Uma conversão normal leva 13 ciclos de *clock* do ADC. A primeira conversão leva 25 ciclos devido a inicialização do circuito analógico. A amostragem e retenção (*sample-and-hold*) leva 1,5 ciclos de *clock* após o início de uma conversão normal e 13,5 ciclos após o início da primeira conversão. Quando o resultado da conversão estiver completo, o resultado é escrito nos registradores de resultado, o bit ADIF é colocado em 1 e o ADSC é limpo. Os diagramas de tempo para a conversão simples são apresentados na fig. 19.2.

Quando uma conversão contínua estiver habilitada, uma nova conversão é iniciada logo após a anterior se completar, enquanto ADSC=1. O diagrama de tempo da conversão contínua é apresentado na fig. 19.3.

PRIMEIRA CONVERSÃO (MODO DE CONVERSÃO SIMPLES)



CONVERSÃO SIMPLES

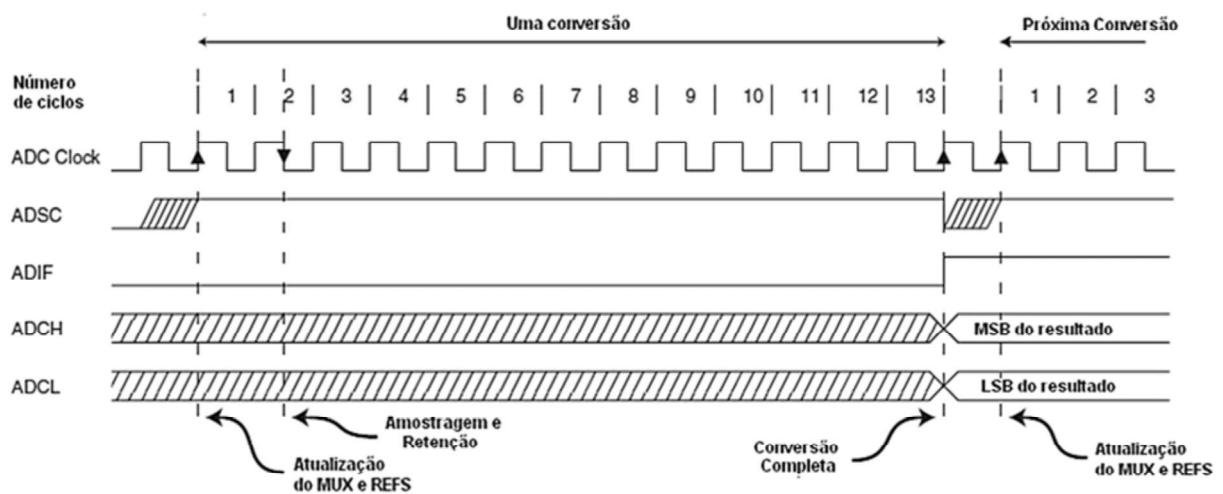


Fig. 19.2 – Diagramas de tempo para a conversão simples.

CONVERSÃO CONTÍNUA

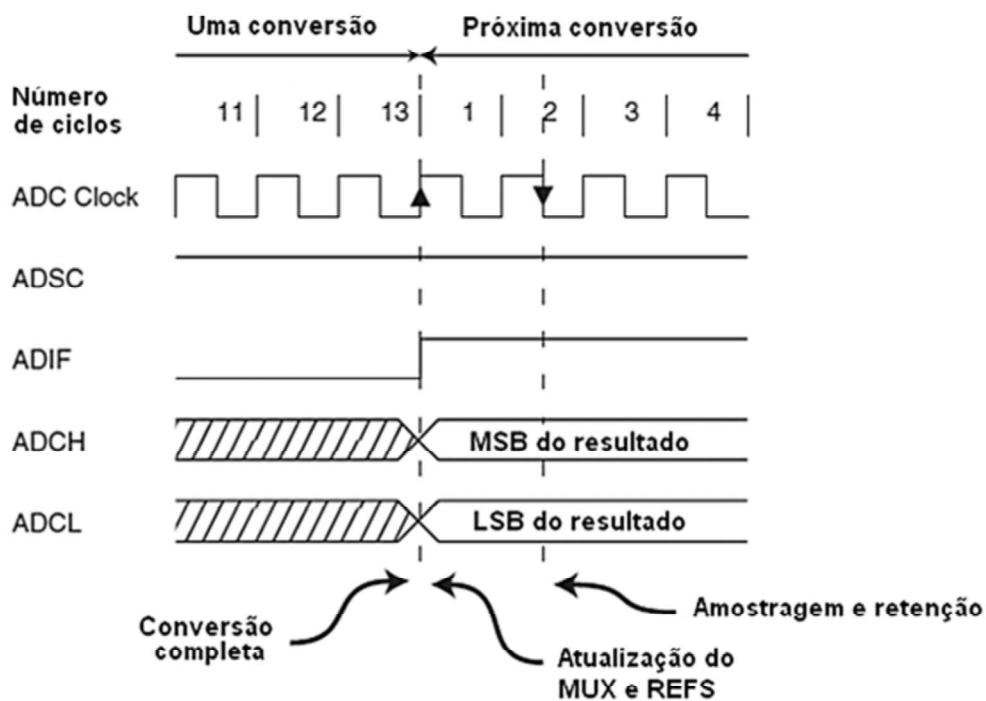


Fig. 19. 3 – Diagrama de tempo para a conversão contínua.

Cuidados devem ser tomados quando se muda o canal para a conversão. O registrador ADMUX responsável pela mudança pode ser atualizado seguramente nas seguintes ocasiões:

1. Quando o bit ADATE ou o bit ADEN for igual a zero.
2. Durante a conversão, pelos menos um ciclo de *clock* do ADC após o início da conversão.
3. Após a conversão, antes do *flag* de interrupção utilizado como disparo ser limpo.

Quando atualizado nessas condições, o ADMUX irá afetar a próxima conversão do ADC.

No modo de conversão simples, o canal deve sempre ser selecionado antes do início da conversão. No modo de conversão contínua, o canal deve ser selecionado antes da primeira conversão.

A referência de tensão para o ADC (V_{REF}) indica a faixa de conversão e pode ser: AVCC, referência interna de 1,1 V ou referência externa (no pino

AREF). Se a referência externa for utilizada, é recomendado o emprego de um capacitor entre o pino de AREF e o terra. Independente disso, a alimentação do ADC é feita pelo pino AVCC que não pode diferir de $\pm 0,3$ V de VCC. O circuito recomendado pela Atmel para a alimentação do ADC é apresentado na fig. 19.4 (filtro passa-baixa). Mesmo que o ADC não seja empregado, o pino AVCC deve ser ligado ao VCC.

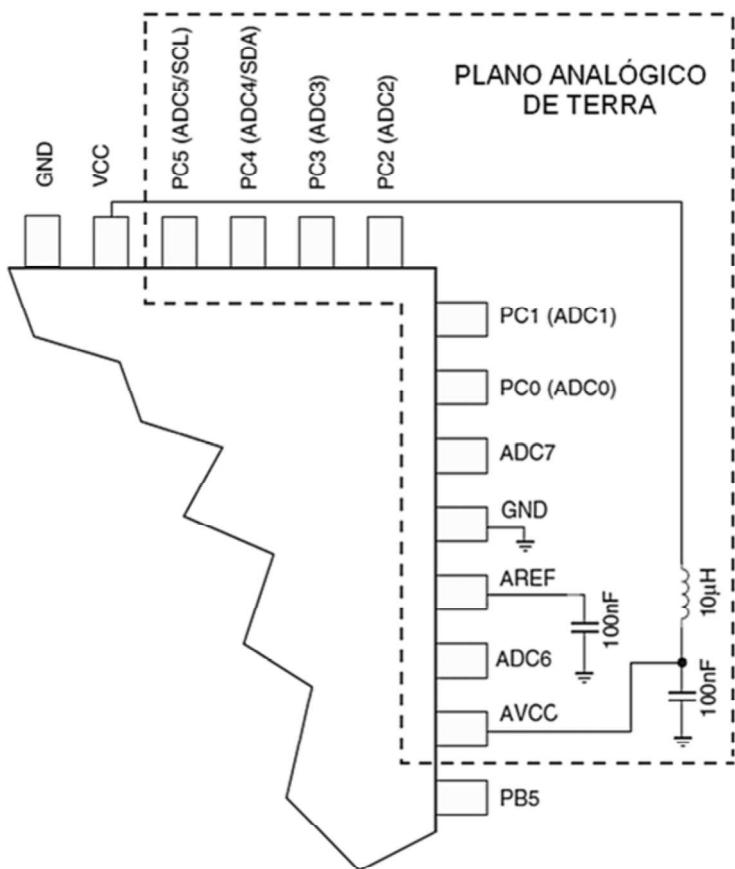


Fig. 19.4 – Circuito de alimentação recomendado para o ADC (encapsulamento TQFP).

O ADC possui um circuito eliminador de ruído que permite a conversão durante o modo *sleep*, visando reduzir o ruído induzido pela CPU e pelos periféricos de I/O. O eliminador de ruído pode ser usado no modo de redução de ruído do ADC ou no modo *Idle*. Para empregar essa característica, os seguintes passos devem ser observados:

1. O ADC deve estar habilitado e não estar realizando uma conversão. O modo de conversão simples deve estar selecionado e a interrupção de conversão completa habilitada.

2. Deve-se entrar no modo de redução de ruído do ADC ou modo *idle*. O ADC começará a conversão após a CPU parar.

Se nenhuma outra interrupção ocorrer antes da conversão do ADC estar completa, a interrupção do ADC irá despertar a CPU e executar a rotina de interrupção para a conversão completa do ADC. Se outra interrupção despertar a CPU antes do ADC completar a conversão, a interrupção será executada e o ADC só gerará a interrupção ao final de sua conversão. A CPU se manterá ativa até que um novo comando de *sleep* seja executado.

O ADC não será desligado automaticamente no modo *idle* ou de redução de ruído do ADC. Para evitar consumo de energia, o ADC deve ser desabilitado antes de se entrar num desses modos.

19.1 REGISTRADORES DO ADC

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Lê/Escrive	L/E	L/E	L/E	L	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bits 7:6 – REFS1:0 – Reference Selection Bit

Estes bits selecionam a fonte de tensão para o ADC, conforme tab. 19.1. Se uma mudança ocorrer durante uma conversão, a mesma não terá efeito até a conversão ser completada (ADIF no ADCSRA estar ativo). A referência interna não pode ser utilizada se um tensão externa estiver sendo aplicada ao pino AREF.

Tab. 19.1 – Bits para a seleção da tensão de referência do ADC.

REFS1	REFS0	Seleção da Tensão de Referência
0	0	AREF, tensão interna V _{REF} desligada.
0	1	AVCC. Deve-se empregar um capacitor de 100 nF entre o pino AREF e o GND.
1	0	Reservado.
1	1	Tensão interna de referência de 1,1 V. Deve-se empregar um capacitor de 100 nF entre o pino AREF e o GND.

Bit 5 – ADLAR – ADC Left Adjust Result

Afeta a representação do resultado da conversão dos registradores de dados do ADC. ADLAR = 1, alinhado à esquerda; ADLAR = 0, alinhado à direita (ver a fig. 19.5). A alteração deste bit afeta imediatamente os registradores de dados.

Bits 3:0 – MUX3:0 – Analog Channel Selection Bits

Selecionam qual entrada analógica será conectada ao ADC (tab. 19.2).

Tab. 19.2 – Seleção do canal de entrada.

MUX3..0	Entrada
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Sensor interno de temperatura
1001-1101	reservado
1110	1,1 V (tensão fixa para referência)
1111	0 V (GND)

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Lê/Escrive	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

Bit 7 – ADEN – ADC Enable

Habilita o ADC. Se ADEN=0, o ADC é desligado. Desligar o ADC durante uma conversão irá finalizá-la.

Bit 6 – ADSC – ADC Start Conversion

No modo de conversão simples, ADSC=1 irá iniciar uma conversão. No modo de conversão contínuo, ADSC=1 irá iniciar a primeira conversão. ADSC ficará em 1 durante todo o processo de conversão e será zerado automaticamente ao seu término. Se o ADSC é escrito ao mesmo tempo em que o ADC é habilitado, a primeira conversão levará 25 ciclos de *clock* do ADC ao invés dos 13 normais.

Bit 5 – ADATE – ADC Auto Trigger Enable

Ativa o modo de auto disparo. O ADC começará uma conversão na borda positiva do sinal selecionado de disparo. A fonte de disparo é selecionada nos bits ADTS2:0 do registrador ADCSRB.

Bit 4 – ADIF – ADC Interrupt Flag

Este bit é ativo quando uma conversão for completada e o registrador de dados atualizado.

Bit 3 – ADIE – ADC Interrupt Enable

Este bit habilita a interrupção do ADC após a conversão se o bit I do SREG estiver habilitado.

Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits

Determinam a divisão do *clock* da CPU para o *clock* do ADC, conforme a tab. 19.3.

Tab. 19.3 – Seleção da divisão de *clock* para o ADC.

ADPS2	ADPS1	ADPS0	Fator de Divisão
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Exemplo do uso do prescaler do ADC:

É importante respeitar a máxima frequência de amostragem para o ADC, caso contrário a leitura apresentará erros. Assim, supondo o ADC trabalhando na sua máxima resolução (10 bits), a taxa máxima de trabalho é de 15 kSPS e, sabendo-se que uma conversão leva 13 ciclos de *clock* do AD, pode-se, então, determinar a máxima frequência desse *clock*.

Supondo o ATmega trabalhando a 16 MHz, resulta:

$$max_clk_{AD} = 15k \times 13 = 195 \text{ kHz}$$

Resultando num prescaler de:

$$prescaler = \frac{F_{CPU}}{max_clk_{AD}} = \frac{16 \text{ MHz}}{195 \text{ kHz}} = 82$$

Como não existe um prescaler com esse valor, escolhe-se 128, que permite respeitar o máximo de 15 kSPS, o que resultará em 9,6 kSPS.

Para diminuir possíveis erros devido a ruídos na conversão, quando não existe necessidade de se trabalhar na máxima frequência, é aconselhado utilizar a menor frequência possível para o ADC.

ADCL/ADCH – ADC Data Register

Quando a conversão estiver completa, o resultado é encontrado nesses dois registradores. Quando ADCL é lido, o registrador de dados não é atualizado até que o ADCH seja lido também. Consequentemente, se o resultado é alinhado à esquerda e não se necessitam mais que 8 bits de resolução, basta ler somente ADCH. Caso contrário, ADCL deve ser lido primeiro. Na fig. 19.5, é apresentado o alinhamento do resultado da conversão à direita e à esquerda, bit ADLAR=0 e ADLAR=1, respectivamente (do registrador ADMUX).

		15	14	13	12	11	10	9	8	
ADLAR=0		ADCH	-	-	-	-	-	-	ADC9	ADC8
		ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Bit		7	6	5	4	3	2	1	0	
Lê/Escrive		L	L	L	L	L	L	L	L	
		L	L	L	L	L	L	L	L	
Valor Inicial		0	0	0	0	0	0	0	0	
		0	0	0	0	0	0	0	0	

		15	14	13	12	11	10	9	8	
ADLAR=1		ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
		ADCL	ADC1	ADC0	-	-	-	-	-	
Bit		7	6	5	4	3	2	1	0	
Lê/Escrive		L	L	L	L	L	L	L	L	
		L	L	L	L	L	L	L	L	
Valor Inicial		0	0	0	0	0	0	0	0	
		0	0	0	0	0	0	0	0	

Fig. 19.5 – Justificação do resultado à direita e à esquerda nos registradores de resultado do ADC.

ADCSR – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0
ADCSR	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0
Lê/Escrive	L	L/E	L	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bits 2:0 – ADTS2:0 - ADC Auto Trigger Source

Se o bit ADATE no registrador ADCSRA for 1, o valor desses bits seleciona a fonte para o disparo da conversão. Se ADATE é zero esses bits não têm efeito. As possíveis configurações para os bits ADTS2:0 são apresentadas na tab. 19.4.

Tab. 19.4 – Configurações para os bits ADTS2:0.

ADTS2	ADTS1	ADTS0	Fonte de disparo
0	0	0	conversão contínua
0	0	1	comparador Analógico
0	1	0	interrupção Externa 0
0	1	1	igualdade de comparação A do TC0
1	0	0	estouro de contagem do TC0
1	0	1	igualdade de comparação B do TC1
1	1	0	estouro de contagem do TC1
1	1	1	evento de captura do TC1

DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0
DIDR0	-	-	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Lê/Escrive	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
---	---	---	---	---	---	---	---

Bits 5:0 – ADC5D:0D – ADC5:0 Digital Input Disable

Estes bits desabilitam individualmente as entradas digitais dos pinos do ADC. Deve(m) estar ativo(s) sempre que o pino correspondente for utilizado como entrada para o ADC, caso contrário deve(m) estar em zero.

19.2 MEDIÇÃO DE TEMPERATURA

O ATmega328 possui internamente um sensor de temperatura que é conectado a um canal do ADC. Ele é selecionado nos bits MUX3:0 do registrador ADMUX. Para o seu funcionamento é necessário selecionar a tensão interna de referência de 1,1 V. A relação entre a tensão gerada e a temperatura é de aproximadamente 1 mV/°C (mais ou menos 1 LSB/°C) e a precisão é de ± 10 °C. Alguns valores típicos são apresentados na tab. 19.5.

Tab. 19.5 – Temperatura versus tensão de saída do sensor interno do ATmega328.

Temperatura	- 45 °C	+ 25 °C	+ 85 °C
Tensão	242 mV	314 mV	380 mV

Devido ao processo de fabricação, o comportamento com a temperatura varia de chip para chip. Todavia, é possível obter resultados mais exatos, até ± 2 °C ou ainda melhores, realizando a calibração do sensor através do software de aplicação⁴⁹. Entretanto, isso não é conveniente quando se deseja obter resultados precisos, reproduzíveis e quando se quer simplicidade na programação.

Considerando-se 10 bits para a conversão do AD, a temperatura é determinada por:

$$Temp = \frac{(ADCL |(ADCH \ll 8)) - Offset}{k} \quad (19.2)$$

onde ADCL e ADCH são os registradores do ADC que possuem o valor da conversão, *Offset* é um erro fixo, ajustado manualmente, e *k* é um coeficiente fixo (também deve ser ajustado manualmente).

A seguir, é apresentado um programa para enviar a temperatura do ATmega para um computador utilizando a USART (ver o capítulo 15).

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 atraso
#include <avr/pgmspace.h> //para o uso do PROGMEM, gravação de dados na flash
#include <avr/interrupt.h> //para o uso de interrupções

//Definições de macros para o trabalho com bits
#define set_bit(y,bit) (y|=(1<<bit))
#define clr_bit(y,bit) (y&=~(1<<bit))
#define cpl_bit(y,bit) (y^=(1<<bit))
#define tst_bit(y,bit) (y&(1<<bit))

#endif
```

⁴⁹ Application note da Atmel: AVR122: *Calibration of the AVR's internal temperature reference.*

Sensor_temp_int_ATmega.c (programa principal)

```
//=====
// Lendo o sensor interno de temperatura do ATmega328          //
//=====
#include "def_principais.h"
#include "USART.h"

#define Offset_temp 363           //valor do offset de temperatura

unsigned char digitos[tam_vetor];/*declaração da variável para armazenagem dos dígitos
                                    (tam_vetor está em USART.h)*/
const char msg[] PROGMEM = "Sensor Interno de Temperatura do ATmega328\n\0";

signed int le_temp();
//-----
int main(void)
{
    USART_Inic(MYUBRR);

    //Tensão interna de ref (1.1V), sensor de temp. do chip
    ADMUX = (1<<REFS1) | (1<<REFS0) | (1<<MUX3);

    //habilita o AD e define um prescaler de 128 (clk_AD = F_CPU/128), 125 kHz
    ADCSRA = (1<<ADEN) | (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
    escreve_USART_Flash(msg);

    while(1)
    {
        ident_num((unsigned int)le_temp(),digitos); //leitura de temperatura sem sinal
        USART_Transmite(digitos[2]);
        USART_Transmite(digitos[1]);
        USART_Transmite(digitos[0]);
        USART_Transmite(176);           //símbolo 'o'
        USART_Transmite('C');
        USART_Transmite('\n');        //nova linha
        _delay_ms(1000);
    }
}
//-----
signed int le_temp()
{
    set_bit(ADCSRA, ADSC);         //inicia a conversão
    while(tst_bit(ADCSRA,ADSC));   //espera a conversão ser finalizada

    return (ADC - Offset_temp);    //fator k de divisão = 1
}
//-----
```

USART.h e **USART.c** vistos no capítulo 15.

Exercício:

19.1 – Com base no *application note* AVR122 faça um programa para ler com a maior precisão possível o sensor de temperatura interno do ATmega328. Utilize uma média de leituras para apresentar o resultado.

19.3 LENDO UM TECLADO MATRICIAL⁵⁰

Uma técnica interessante para ler um teclado no formato matricial é utilizar o ADC para converter um sinal de tensão proveniente de uma rede resistiva. Na fig. 19.5, é apresentado um teclado com 16 teclas com um arranjo adequado de resistores. Assim, quando uma determinada tecla for pressionada, será gerada uma tensão única sobre o resistor R9. De acordo com a tensão gerada, se determina qual tecla foi pressionada. Com nenhuma tecla pressionada, a tensão sobre o resistor R9 será nula.

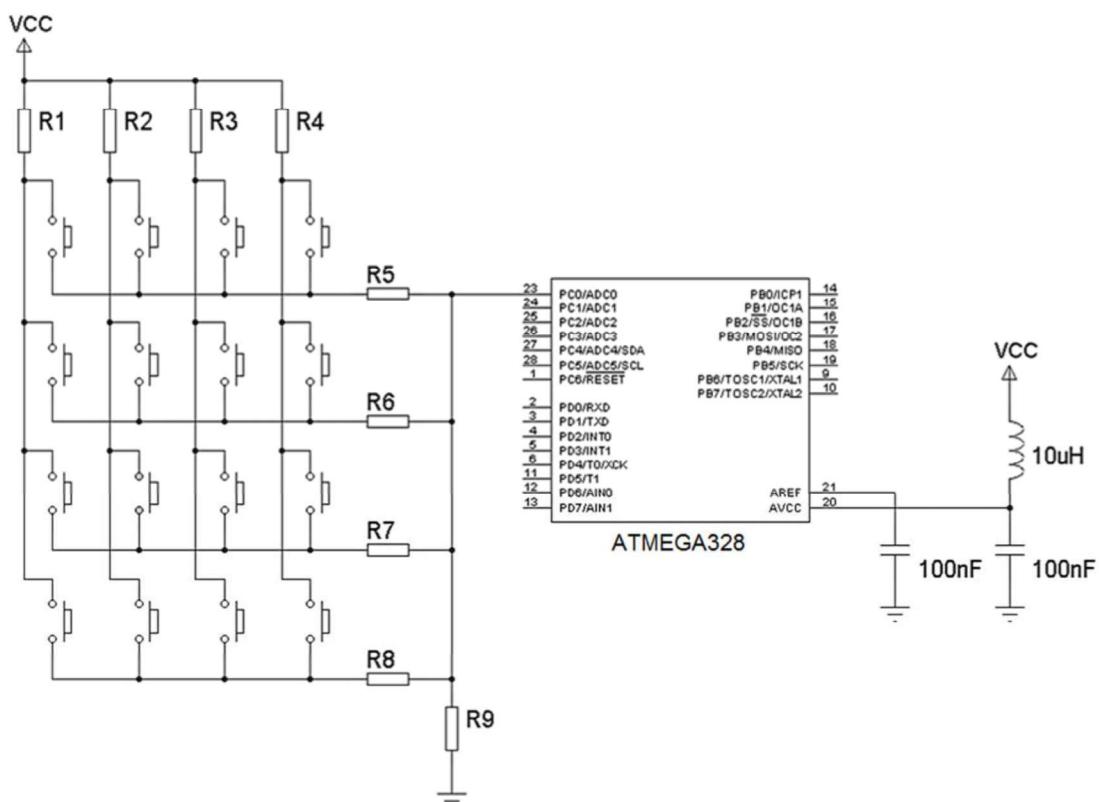


Fig. 19.5 – Leitura de um teclado 4×4 com o uso do ADC.

⁵⁰ Application note da Microchip: *Tips 'n Tricks 8 pin PIC Microcontrollers*.

Para que o sistema funcione adequadamente, é necessário calcular os resistores empregados garantindo 16 faixas de tensão diferentes para a entrada do ADC (fig. 19.5). Nesse caso, é importante empregar resistores de precisão para evitar variações consideráveis de tensão para cada tecla. Para a estabilidade da leitura, na avaliação dos dados do ADC, o programa deve considerar uma faixa de tensão pertinente para cada tecla. A tensão de referência para o ADC deve ser a tensão de alimentação do sistema.

Apesar de ser possível realizar leituras de várias teclas com o ADC, um número inferior a 16 teclas é mais fácil de ser implementado, pois não existirão valores de resistores muito próximos entre si. Assim, teclados matriciais com 12 teclas, 9 teclas ou menos poderão ser feitos com resistores comerciais facilmente encontrados.

Exercícios:

19.2 – Determine valores para os resistores da fig. 19.5. Depois repita o cálculo para um teclado matricial de 12 e 9 teclas.

19.3 – Empregando o ADC do ATmega328, elaborar um circuito e o respectivo programa para medir com precisão uma resistência desconhecida.

19.4 – Faça um programa para a leitura do teclado da fig. 19.6.

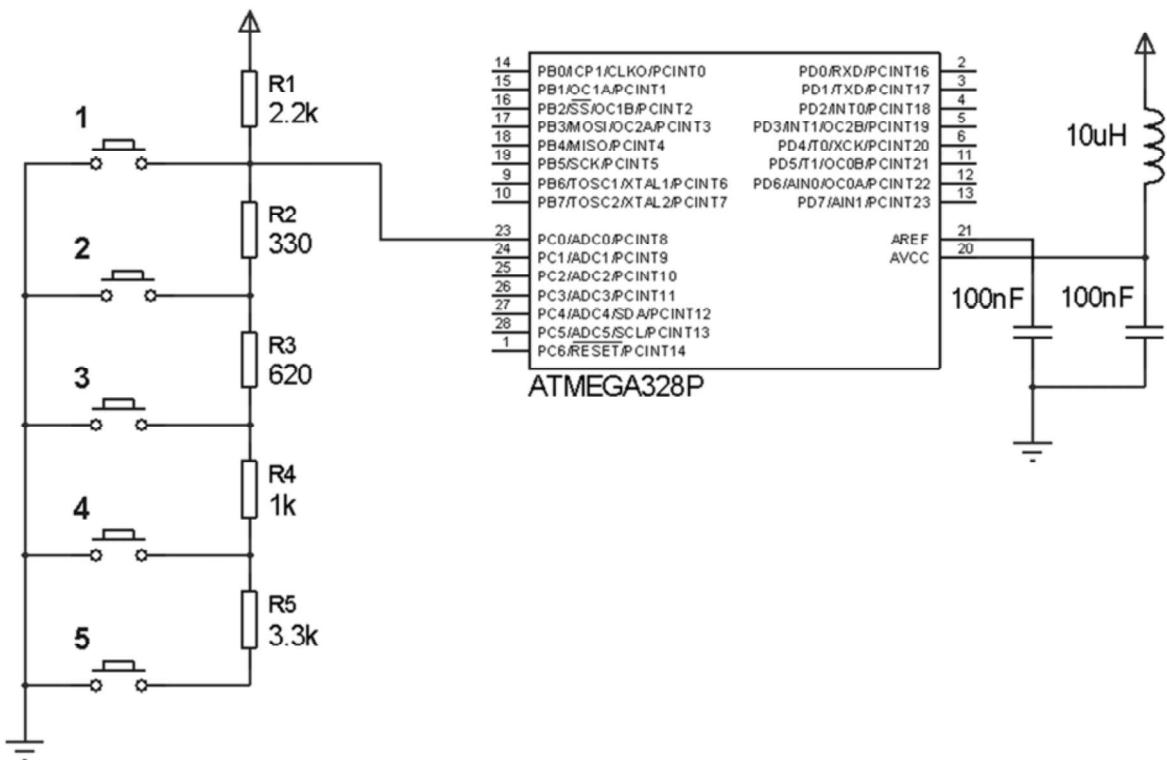


Fig. 19.6 – Leitura de 5 teclas com o ADC do ATmega328.

19.4 SENSOR DE TEMPERATURA LM35

O LM35 é um sensor de temperatura analógico de precisão de aproximadamente $0,25\text{ }^{\circ}\text{C}$ produzido pela National Semiconductors. Apresenta resposta linear de $10\text{ mV}/^{\circ}\text{C}$ (a $0\text{ }^{\circ}\text{C}$ deve fornecer 0 V de saída) e uma faixa de operação de $-55\text{ }^{\circ}\text{C}$ até $+150\text{ }^{\circ}\text{C}$ (dependendo do modelo). Está disponível em vários encapsulamentos, mas o mais comum é o T092 (similar ao dos transistores BC). As vantagens do LM35 são seu custo e a facilidade de integração com microcontroladores que dispõem de ADC.

Caso haja necessidade da leitura de temperaturas negativas, é necessário o emprego de uma fonte negativa de tensão em conjunto com um resistor, ou o uso de um diodo entre o pino de terra e o terra do circuito, com um resistor entre o pino de saída e o terra.

Por ser mais barato e responder a maioria dos requisitos de projeto, geralmente se emprega o LM35D, cuja faixa de temperatura é só positiva, se estendendo de 0 °C até + 100 °C.

Na fig. 19.7, é apresentado o circuito resumido para o teste do LM35 no Arduino⁵¹. A informação de temperatura é enviada a um computador (ver o capítulo 15). Na sequência, é apresentado o programa desenvolvido.

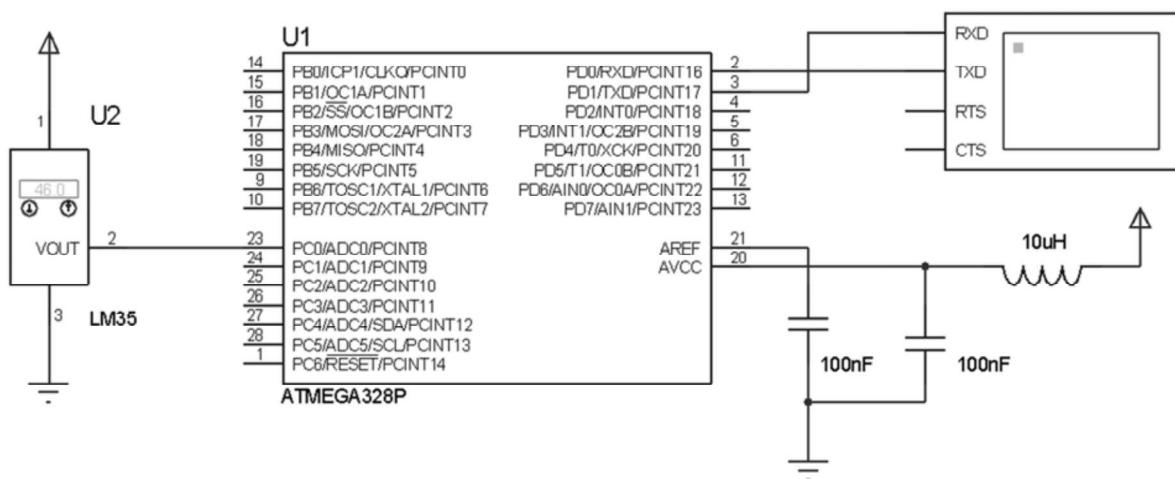


Fig. 19.7 – Analise de temperatura com o LM35 com envio de dados a um computador.

LM35.c (programa principal)

```
===== //
// Sensor de temperatura LM35 - envio de dados para o PC //
// Resolução de 1 grau Centigrado //
===== //
#include "def_principais.h"
#include "USART.h"

const char msg[] PROGMEM = "Sensor de Temperatura LM35\n\0";

unsigned int temp;
unsigned char digitos[tam_vetor];
//-----
int main()
{
    DDRC = 0x00;
    PORTC = 0xFE;

    USART_Inic(MYUBRR);
}
```

⁵¹ A plataforma Arduino não segue a recomendação da Atmel quanto ao uso do indutor ligado ao pino AVCC e do capacitor ligado ao pino AREF (ver circuito na fig. 3.2).

```

//configura ADC
ADMUX = 0b11000000; //Tensão interna de ref (1.1V), canal 0
ADCSRA = 0b11101111; /*habilita o AD, habilita interrupção, modo de conversão
contínua, prescaler = 128*/
ADCSRB = 0x00; //modo de conversão contínua
set_bit(DIDR0,0); //desabilita pino PC0 como I/O, entrada do ADC0

sei();

escreve_USART_Flash(msg);

while(1)
{
    ident_num(temp,digitos);
    USART_Transmite(digitos[3]);
    USART_Transmite(digitos[2]);
    USART_Transmite(digitos[1]);
    USART_Transmite(',');
    USART_Transmite(digitos[0]);
    USART_Transmite(176); //simbolo 'o'
    USART_Transmite('C');
    USART_Transmite('\n');
    _delay_ms(1000);
}
//-----
ISR(ADC_vect)
{
    temp = ADC + (ADC*19)/256;

    /* O LM35 apresenta uma saída de 10mV/°C. O valor de leitura do AD é dado por
    ADC = Vin*1024/Vref, como Vref = 1,1V, para converter o valor do AD para graus
    Celsius, é necessário multiplicar o valor ADC por 1100/1024 (considerando uma
    casa decimal antes da vírgula). Utilizando a simplificação matemática e mantendo
    a variável temp com 16 bits, resulta: 1100/1024 = 1 + 19/256 */

}
//-----

```

def_principais.h foi apresentado anteriormente e **USART.h** foi apresentado no capítulo 15.

Exercício:

19.5 – Elaborar um programa para ler os sensores de temperatura LM35 conforme o circuito da fig. 19.8. Escreva a temperatura de cada um deles em uma linha do LCD 16×2.

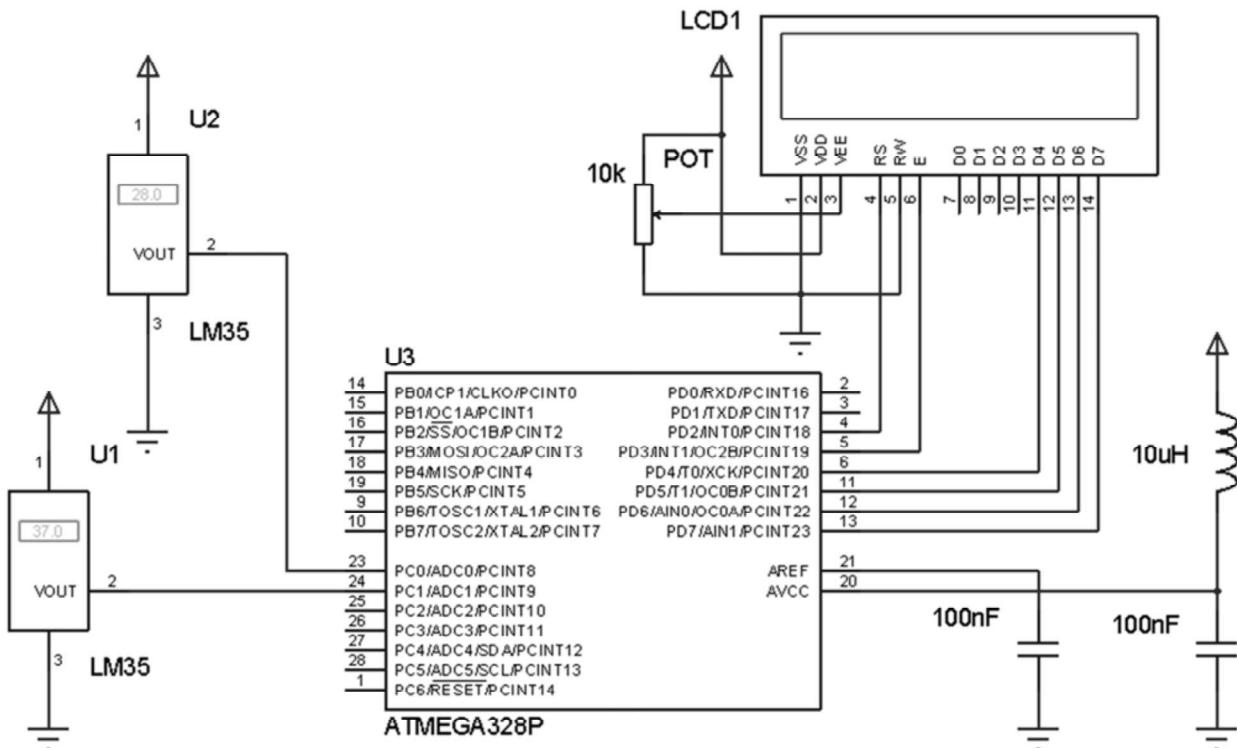


Fig. 19.8 – Empregando o ATmega328 para ler dois sensores de temperatura LM35.

19.5 SOBREAMOSTRAGEM

O ADC do ATmega possui uma resolução máxima de 10 bits, ou seja permite 1024 níveis de discretização. Entretanto, os dois bits menos significativos são imprecisos, o que faz com que muitos programadores utilizem somente os 8 bits mais significativos, resultando em 256 níveis de discretização. Nesse caso, é só habilitar o bit ADLAR no registrador ADMUX e ler somente o registrador ADCH.

É possível aumentar a resolução do ADC com uma técnica simples, chamada sobreamostragem. Dessa forma, a imprecisão da leitura pode ser diminuída. Essa técnica só pode ser empregada no modo de conversão contínua.

A cada diminuição de 4 vezes na frequência de amostragem do AD, a sua resolução pode ser aumentada em um bit⁵²:

$$\frac{f_{amostragem\,AD}}{4} \rightarrow (\text{ADC_soma} \gg 1)$$

onde ADC_soma é a soma de 4 valores consecutivos amostrados normalmente pelo ADC e “ $\gg 1$ ” é o deslocamento dessa soma um bit para a direita, o que elimina o seu bit menos significativo. Assim, para uma frequência de amostragem de 15 kHz, se essa taxa for diminuída para 3,75 KHz, a resolução do ADC do ATmega328 pode passar de 10 bits para 11 bits. Se a frequência for dividida de novo por 4 (divisão de 15 kHz pelo total de 16) a resolução pode passar para 12 bits e, assim por diante. O valor da conversão é somado durante o intervalo da amostragem para o efetivo aumento da resolução da conversão. Depois, o valor somado é deslocado um bit para a direita a cada diminuição de 4 vezes na frequência de amostragem. O processo é ilustrado na fig. 19.9.

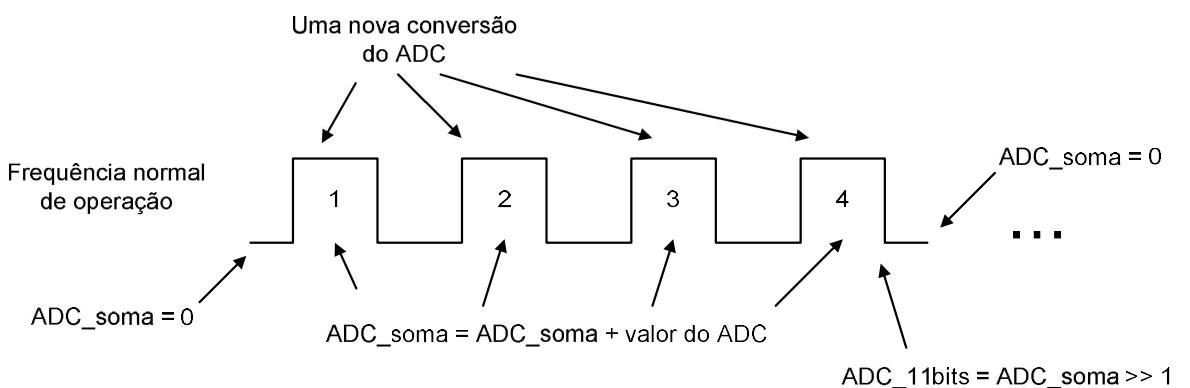


Fig. 19.9 – Esquema de sobreamostragem para aumentar a resolução do ADC de 10 para 11 bits.

Em resumo, a frequência normal de trabalho do ADC gera uma nova amostra a cada intervalo de conversão, representados pelos números 1 a 4 na fig. 19.9. A cada nova conversão, o valor do ADC deve ser somado em uma variável auxiliar, no caso ADC_soma. Após 4 conversões, a variável que contém a soma deve ser deslocada 1 bit para a direita (dividida por 2). Esse valor deslocado é o novo valor do ADC, agora com uma resolução de 11

⁵² Application Note da Atmel: AVR121: *Enhancing ADC resolution by oversampling*.

bits. A frequência de amostragem é 4 vezes menor porque somente após quatro conversões normais do ADC é que é gerado um novo valor com 11 bits. A seguir, é apresentado um trecho de código que ilustra a aplicação da sobreamostragem para se obter o aumento de um bit na resolução do ADC.

```
-----
// Sobreamostragem - aumentando a resolução do ADC de 10 para 11 bits
-----
unsigned int ADC_11bits;
. .
ISR(ADC_vect)
{
    static unsigned int ADC_soma=0;
    static unsigned char cont=4;

    ADC_soma += ADC;      //soma o valor do ADC na variável para o aumento da resolução
    cont--;
    if(cont==0)
    {
        ADC_11bits = ADC_soma >> 1;    //deslocamento de 1 bit para a direita
        ADC_soma=0;
        cont=4;
    }
}
//-----
```

Exercício:

19.6 – Elaborar um programa para ler a tensão proveniente de um potenciômetro com o ADC do ATmega328 (ver a fig. 19.10). Apresente o valor binário no computador (ver o capítulo 15). Depois aumente a resolução do ADC para 11 bits e 12 bits, respectivamente. Verifique os resultados.

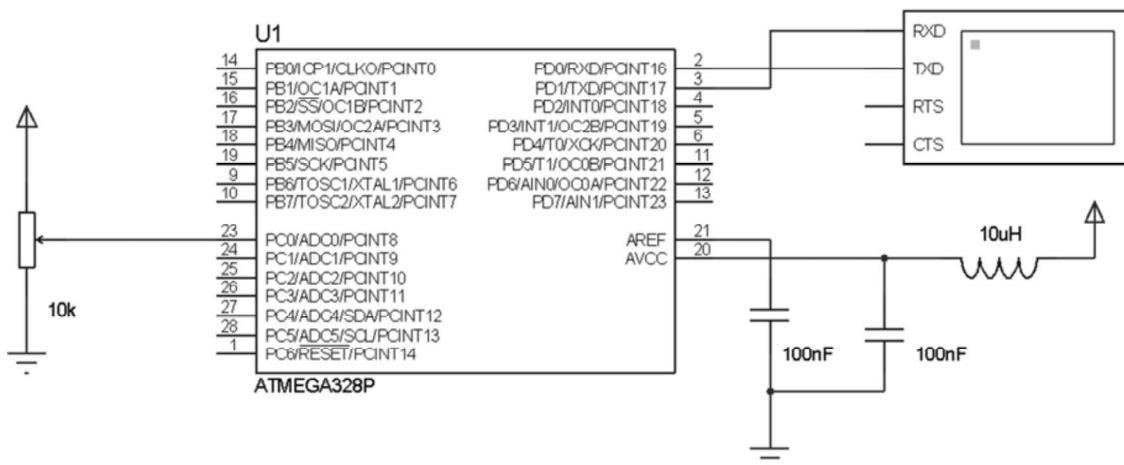


Fig. 19.10 – Lendo a tensão proveniente de um potenciômetro.

Obs.: potenciômetros são comuns em dispositivos de controle de posição, como por exemplo, joysticks.

19.6 FILTRO DE MÉDIA MÓVEL

Muitas vezes, para eliminar ou diminuir algum ruído indesejável em um sinal, é necessário filtrá-lo. Isso é comum quando se trabalha com sinais provenientes do ADC. Essa filtragem é, então, feita através da programação. O problema é que a teoria sobre filtros digitais é complexa e o programador a evita sempre que possível. Todavia, pode-se utilizar um filtro bem simples, chamado de média móvel, que pode resolver o problema do ruído.

O filtro de média móvel é obtido calculando-se a média de um conjunto de valores, sempre se adicionando um novo valor ao conjunto e se descartando o mais velho. Não é apenas uma média de um conjunto isolado de valores. O filtro de média móvel é representado por:

$$y[n] = \frac{1}{N+1} \sum_{k=0}^N x[n-k] \quad (19.3)$$

onde n é o tempo atual (é o índice dos vetores utilizados), $N + 1$ é o número de amostras utilizadas para a filtragem, $y[n]$ é o sinal filtrado e $x[n-k]$ representa o conjunto dos valores a serem somados. A eq. 19.3 pode ser representada pelo diagrama da fig. 19.11, com os valores de $b_0, b_1 \dots b_N$ iguais a $1/(N+1)$.

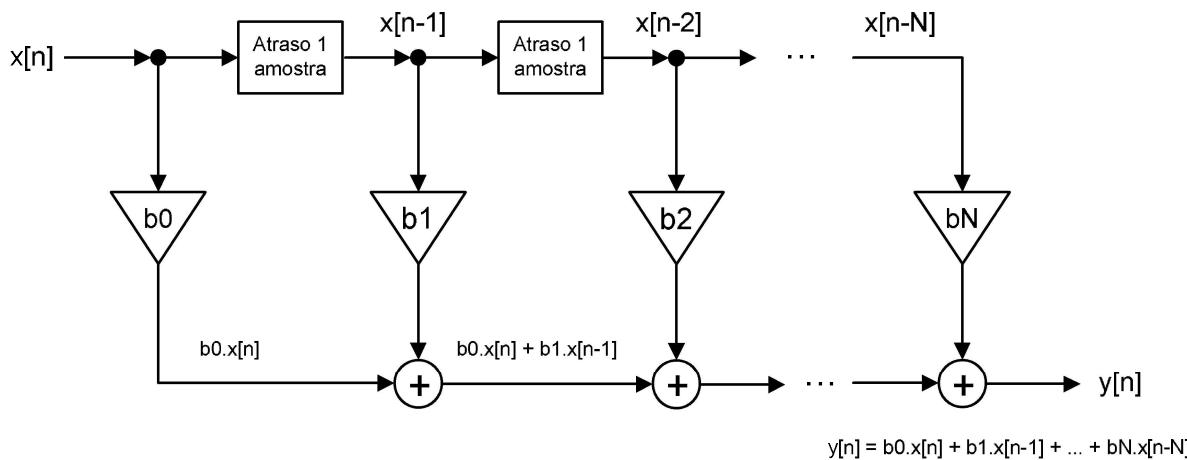


Fig. 19.11 – Diagrama de um filtro digital não recursivo.

O diagrama da fig. 19.11 representa um filtro chamado não recursivo, pois o sinal de saída $y[n]$ depende somente do sinal de entrada $x[n]$. Caso o sinal de saída dependesse de valores passados da saída, o filtro seria chamada recursivo. No projeto de um filtro digital, determinam-se os coeficientes de multiplicação para as amostras, no caso b_0, b_1, \dots, b_N para o diagrama citado. Esses coeficientes podem ser determinados com o uso de alguma ferramenta computacional como o MATLAB® ou MATCAD®, e o programador apenas necessita realizar as somas e multiplicações nas amostras certas. A qualidade do filtro e o tipo de filtro (passa baixa, passa alta, passa faixa) vai depender do número de coeficientes utilizados (quantidade de amostras) e dos seus valores.

Ao utilizar coeficientes fixos, o filtro de média móvel produz um filtro passa baixa suave, reduzindo os sinais de alta frequência. Caso se deseje outro tipo de filtragem, será necessário a multiplicação com valores fracionários, o que exigirá o uso de ponto flutuante no programa. Isso pode ser um problema para microcontroladores de 8 bits pelo consumo maior de memória e da limitada capacidade de processamento da CPU.

Na fig. 19.12, é apresentada a resposta em frequência de um filtro de média móvel para 16 amostras. O eixo horizontal é a frequência em Hertz, o número 1 representa a frequência de amostragem do sinal dividida por 2. Desta forma, supondo uma frequência de amostragem 20 kHz, cada linha vertical do gráfico corresponderia a 1 kHz.

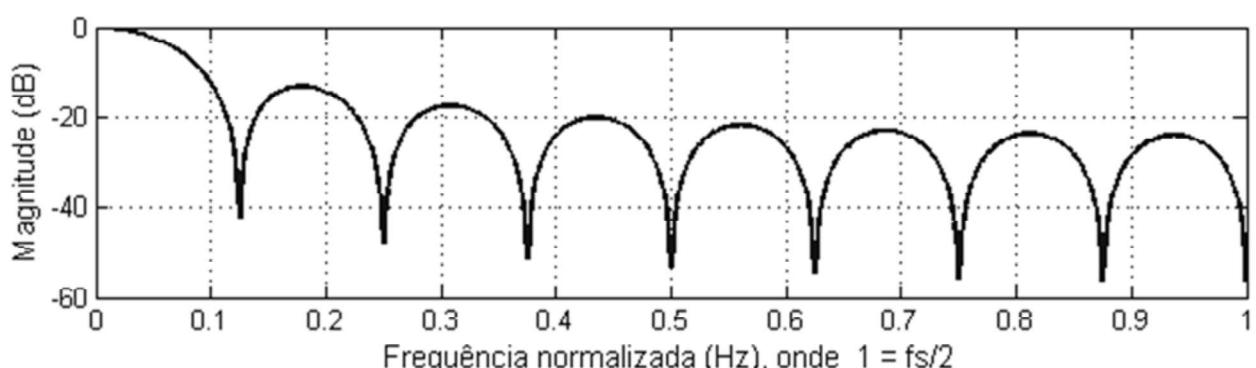


Fig. 19.12 – Resposta em frequência de um filtro de média móvel de 16 amostras.

A resposta apresentada na fig. 19.12 mostra que o desempenho de um filtro de média móvel é razoável, estando a atenuação dos sinais indesejados na faixa de aproximadamente -20 dB (90 %).

A seguir, é apresentado um exemplo para a programação de um filtro de média móvel de 16 amostras para os valores convertidos pelo ADC do ATmega.

```
-----  
// Filtro de média móvel com 16 amostras para o sinal do ADC  
-----  
unsigned int x[16], y;  
. . .  
ISR(ADC_vect)  
{  
    unsigned char i=15;  
  
    y=0;  
    do  
    {  
        x[i] = x[i-1]; //anda um passo nas amostras anteriores  
        y += x[i];  
        i--;  
    } while (i!=0);  
  
    x[0]=ADC;          //nova amostra entra no filtro  
    y = (y + x[0])/16; //sinal filtrado  
}  
-----
```

Exercício:

19.7 – Utilize um filtro de média móvel de 5 amostras para o exercício 19.6.
