Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Curso de Engenharia de Telecomunicações – TGT

Programa de Educação Tutorial – PET
Grupo PET-Tele

Tutoriais PET-Tele

Introdução à Amostragem de Sinais com o *kit* Arduino

(Versão: A2016M06D21)

Autores: Lorraine de Miranda Paiva (2016)
Lucas Pontes Siqueira (2016)

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Junho / 2016

Sumário

1 Introdução

O Arduino é uma plataforma de prototipagem de uso bastante simples comparando-se com outras e, por isso, é bastante utilizada no mundo inteiro. Pensando nisso, o grupo PET-Tele resolveu usar sua experiência de uso da plataforma para aplicar a estudos relacionados com processamento digital de sinais, matéria que é ministrada pelo tutor do grupo.

Neste tutorial será mostrado como fazer leituras de sinais usando o Arduino. Esses sinais podem ser sensores como de temperatura, luminosidade ou sinais mais complexos. O dispositivo que irá gerar o sinal estará conectado a uma porta analógica e a partir daí, o Arduino fará a conversão A/D dos valores desse sinal. Esses valores serão armazenados no Arduino e poderão ser usados para muitas utilidades.

Existem duas formas de se fazer amostragem de um sinal no Arduino. A forma mais simples é feita com a função `analogRead()`, que é a função responsável por ler cada amostra de uma vez de um sinal e, a outra, é feita ativando um modo de operação do microcontrolador chamado de *free running*.

2 Usando a função analogRead()

Para fazer uma amostragem com a função `analogRead()` é necessário fazer uma pequena rotina usando uma estrutura de *loop* e controlar quantas amostras do sinal serão lidas. Como a função `analogRead()` só retorna uma amostra do sinal, ela será usada dentro do *loop*, para que se leia as n amostras que se queira. Abaixo está exemplificado um pequeno código para esse tipo de leitura:

```
for(i=0; i<n; i++){
vetor[i] = analogRead(sensor);
}
```

No código, a cada vez que o *loop* é executado a função `analogRead()` armazena uma amostra na posição i do vetor. Quando o *loop* termina, a variável i é incrementada e o processo se repete.

Mas esse tipo de leitura possui alguns problemas. O intervalo entre cada amostra não é constante, o que torna a veracidade do sinal amostrado em relação ao sinal original duvidosa. Com isso, esse tipo de leitura é mais recomendada para sinais de baixa frequência.

3 Usando rotinas de interrupções

3.1 Como escolher a melhor taxa de amostragem

Para definir a taxa de amostragem, a frequência de oscilação do cristal é usada e, a partir dela, o fator *prescaler* divide essa frequência gerando a frequência de amostragem. Os fatores que podem ser utilizados são do resultado da potenciação de 2, indo de 2^1 até 2^7 (128).

Esses fatores são selecionados no registrador ADCSRA (*ADC Control and Status Register A*).

3.2 Configurando o Atmega328 para ativar o modo *free running*

O outro tipo de leitura de um sinal é feito usando rotinas de interrupção. Nesse tipo de leitura o microcontrolador será manipulado para que se ative um modo chamado *free running*. Nesse modo, o microcontrolador irá fazer a leitura do sinal e a conversão A/D automaticamente em intervalos constantes. Quando cada conversão for finalizada, a rotina de interrupção será chamada e irá retirar o valor de cada amostra antes que o microcontrolador faça a leitura de uma nova amostra.

O Atmega328 possui rotinas específicas de interrupção e neste processo de amostragem será usada a ISR (ADC_vect) que é a ISR indicada para a conversão A/D.

Para ativar o modo de *free running*, primeiro precisamos configurar o ADMUX (*ADC Multiplexer selection Register*). Nele, iremos configurar a tensão de referência que será usada na leitura do sinal e qual canal analógico será lido (porta analógica).

Após ter configurado o ADMUX iremos configurar os ADCSRA e ADCSRB (*ADC Control and Status Register A e B*). Nele, iremos basicamente habilitar a conversão A/D, configurar o *prescaler*, que seria a taxa de amostragem e enviar o *bit* para iniciar a conversão.

```
void setup()
pinMode(marker, OUTPUT);
DIDR0 = 0x3F;
ADMUX = 0x43;
ADCSRA = 0xAC;
```

```
ADCSRB = 0x40;
bitWrite(ADCSRA, 6, 1);
sei();
```

Após ter configurado o Atmega328 a ISR sempre será chamada logo após cada conversão for realizada. Nessa rotina o programa irá armazenar os *bytes* do registrador com o valor da conversão no vetor. Primeiro será tirado os *bits* menos significativos e depois os mais significativos.

```
ISR(ADC\_vect)
bitClear(PORTB, 4);
aval = ADCL;
aval += ADCH << 8;
bitSet(PORTB, 4);
```

4 Apêndice

```
int sendStatus = 0; // envia o status
int startDelay = 0;
byte valueBin[1024]; // valores binarios
int counter = 0;

void setup() {
TIMSK0 = 0x00; // desabilita o contador
DIDR0 = 0x3F; // desabilita entradas digitaos
ADMUX = 0xC3; // fazendo a leitura ADC3, justificado a direita, valor de referência
interna de 1.1V
ADCSRA = 0xAC; // conversor AD ligado, interrupção habilitada, prescaler = 128
ADCSRB = 0x40; // canais AD MUX ligados, modo free running
bitWrite(ADCSRA, 6, 1); // Envio do bit 6 para iniciar a conversão (=ADSC) no ADCSRA
sei(); // Habilita a interrupção Set global interrupt flag
Serial.begin(9600);
}

void loop() {
if (sendStatus == 1) {
for (int i=0; i<=1023; i++) { // valores binarios de saida
Serial.print(i);
Serial.print("\t");
Serial.println(valueBin[i]);
}
Serial.println("Done");
sendStatus = 2;
}
}

/*** Rotina de Interrupçao do ADC ***/
ISR(ADC_vect) {
int aval = ADCL; // armazena o byte menos significativo do ADC
aval += ADCH << 8; // armazena o byte mais significativo do ADC
if (sendStatus == 0) {
if (startDelay < 1000) { // descarta as primeiras x amostras
startDelay++;
}
else {
valueBin[counter] = aval; // incrementa o valor binario
counter ++;
if (counter == 1000) { // para se um valor de amostra está cheio
sendStatus = 1;
}
}
}
}
```