PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

ESCOLA POLITÉCNICA
NÚCLEO DE ENGENHARIAIS

SISTEMAS MICROPROCESSADOS

LUIS HENRIQUE DA ROCHA TROSCIANCZUK
ENGENHARIA DE COMPUTAÇÃO

TOCADOR DE MÚSICA PARA ATMEGA328P PROGRAMADO EM C PARA AVR

Curitiba
Novembro de 2020

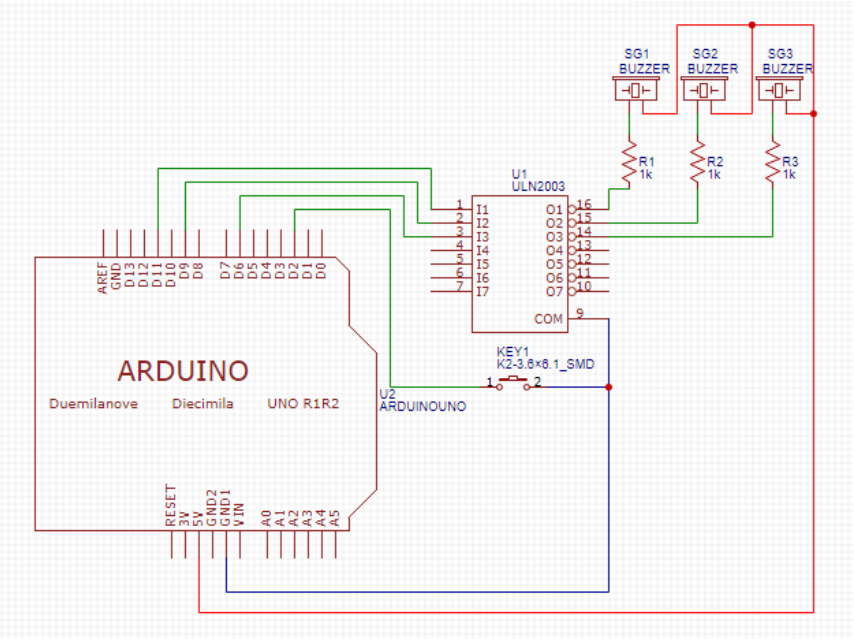
1. INTRODUÇÃO

O objetivo do projeto é conseguir fazer o ATMEGA328P de um Arduino Uno conseguir fazer o acionamento de 3 *buzzers* passivos, tocando simultaneamente e independentemente, a fim de conseguir executar além de melodias de músicas, também as harmonias que as acompanham.

OBJETIVOS

1. Ativar os 3 *timers* disponíveis no ATMEGA328P;
2. Fazer *set* dos registradores de cada *timer* que fazem o acionamento de PWM nas portas digitais;
3. Calcular a frequência correta dos PWM para cada nota específica;
4. Calcular o tempo correto de cada nota;
5. Trocar as frequências das notas em tempo de execução.

2. CIRCUITO

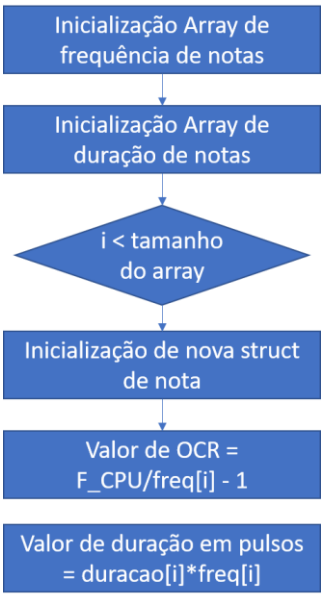


Nesse circuito utilizamos as saídas do ULN2003 para fazer o acionamento dos *buzzers*, e as entradas sendo acionadas pelas portas D6, D9 e D11 que

equivalem no Atmega a PD6, PB1 e PB3. Essas portas foram selecionadas para o acionamento dos *buzzers* pois são as portas de acionamento das portas COMA dos *timers* 0, 1 e 2 respectivamente, que será mais bem explicado na seção seguinte. A porta PD2 é configurada com uma entrada com *pull-up*, quando é feito o pressionamento do botão o Atmega inicia a música.

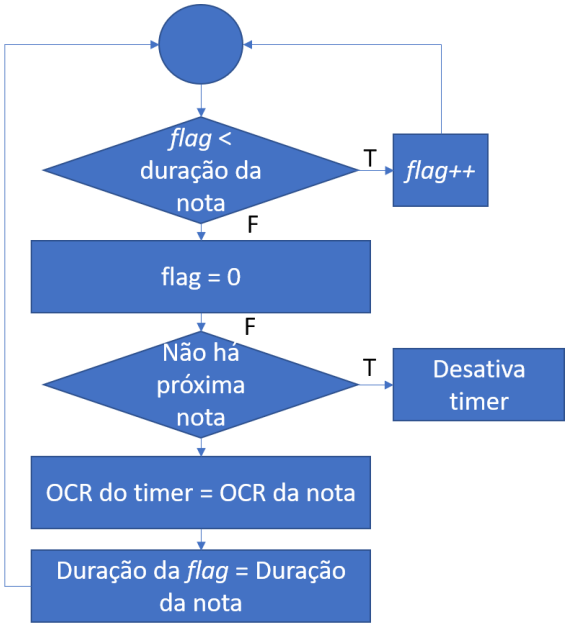
3. EXPLICAÇÃO DO ALGORITMO

A inicialização de uma música funciona da seguinte forma:



*Nos *timers* 0 e 2 a fórmula do valor de OCR é F_CPU/256/frequência. É utilizado o prescaler de 256, pois há menos resolução com o registrador OCR de apenas 8 bits.

Foi optado por essa forma de cálculo de duração por pulsos pois como está sendo utilizado o modo CTC, é utilizada uma flag contadora que é incrementada a cada chamada do *interrupt*, e ao atingir esse número o registrador OCRA recebe o valor da nova nota e é zerada a flag contadora. Também foi optado por ativar o modo *toggle* da saída COM dos *timer* para haver a geração de um sinal PWM nessas saídas de *duty cycle* de 50% e frequência igual ao valor do *clock* do ATMEGA sobre o valor do OCRA do *timer* sem a necessidade da ativação explícita da saída que desejamos em seu registrador *PORT*. O diagrama de blocos do algoritmo do *interrupt* é o seguinte:



Note que os *interrupts* de comparação A dos 3 *timers* tem esse mesmo formato, portanto para haver diferenciação entre qual *buzzer* vai tocar que notas, as séries de notas são armazenadas em listas ligadas de structs do tipo Nota cujo primeiro ponteiro é armazenado em uma struct do tipo Música, que têm esse formato:

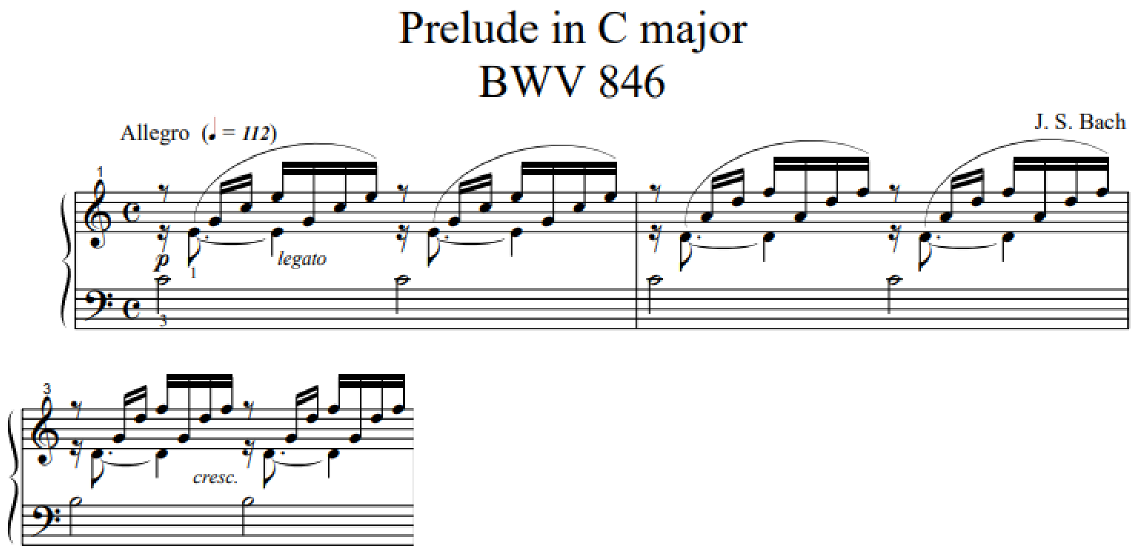
struct Nota:	struct Música:
Frequência Duração Valor de OCR Duração em Pulsos Ponteiro da próxima nota	Tempo em BPM Primeira nota aguda Primeira nota media Primeira nota grave

A struct música armazena o tempo em BPM para facilitar a conversão direta de partitura em *arrays* de frequência e duração por valor da nota. Para esse programa foi convencionado o valor de 256 para um compasso de 4 batidas, portanto a duração em segundos de uma nota pode ser calculada através de:

$$Duração = \frac{60 \cdot valor\ da\ nota}{32 \cdot BPM}$$

O valor de 32 no denominador representa o valor de uma única batida, portanto é possível fazer a conversão de músicas com qualquer fórmula de compasso. No cabeçalho do arquivo foi definido o valor das notas desde a semifusa ($\frac{1}{64}$ de uma

batida) até uma semibreve, ou 4 tempos, além dos valores de frequência de quase 9 oitavas inteiras, apesar do programa não conseguir executar as notas até o B3, o que será discutido em seguida. Para ilustração abaixo é a transcrição das notas dos 3 primeiros compassos do Prelúdio em Dó Maior de J.S. Bach:




```
song initializePreludeCMajor(){

    const uint16_t trebleNotes[] = {PAUSE, E7, G7, C8, E8, G7, C8, E8,
        PAUSE, E7, G7, C8, E8, G7, C8, E8, PAUSE, D7, A7, D8, F8, A7, D8,
        F8, PAUSE, D7, A7, D8, F8, A7, D8, F8, PAUSE, D7, A7, D8, F8, A7, D8,
        F8, PAUSE, D7, A7, D8, F8, A7, D8, F8};
    const uint16_t trebleDuration[] = {SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH,
    SIXTEENTH, SIXTEENTH, SIXTEENTH, SIXTEENTH};

    const uint16_t bassNotes[] = {C7, PAUSE, C7, PAUSE, C7, PAUSE, C7, PAUSE,
    B6, PAUSE, B6, PAUSE};
    const uint16_t bassDuration[] = {DOTTED(QUARTER), EIGTH, DOTTED(QUARTER), EIGTH,
    DOTTED(QUARTER), EIGTH, DOTTED(QUARTER), EIGTH,DOTTED(QUARTER), EIGTH, DOTTED(QUARTER), EIGTH};
```

Em notas repetidas, foi optado o uso de notas pontuadas seguidas por uma pausa para haver distinção entre as duas, pois quando há a execução de duas notas iguais seguidas o *buzzer* toca como uma única nota contínua.

4. LIMITAÇÕES DO PROGRAMA

A biblioteca Tone.cpp do Arduino usa um algoritmo de avaliação e processamento dos dados da frequência, alterando o prescaler de cada *timer* de forma que a nota tocada pelo *buzzer* seja a mais fiel possível à frequência desejada, foi optado não fazer esse tipo de processamento neste trabalho pois aumentaria muito a complexidade do código, pois também afetaria o cálculo da duração das

notas por pulso, que já é preprocessada para facilitar a transcrição direta de partituras.

Outro problema que essa limitação de utilizar um *prescaler* fixo trás é para os *timers* de 8 bits (0 e 2), pois eles não consegue tocar notas mais graves que um B3 (cuja frequência é 262) pois necessitaria de um OCR que pudesse ser maior que 255, por exemplo a nota vizinha, A#3, necessitaria de um OCR de $\frac{\frac{16 \cdot 10^6}{256}}{233} - 1 = 267$ que não é comportado por um registrador de 8 bits.

5. RESULTADOS

Ao ser executada, a música é reproduzida de forma satisfatória, sendo possível reconhecê-la. Porém, no fim é possível notar que começa a aparecer um *drift* no tempo entre um canal de áudio de outro, porém acredito que deve ser algum problema de arredondamento entre a frequência e o OCR.

6. CONCLUSÃO

Os *timers* são ferramentas complexas e muito versáteis, aprender a utilizar todas as suas propriedades é um recurso que dá muito poder ao desenvolvedor de sistemas embarcados de conseguir fazer diversos tipos de controles e utilizar diversos atuadores.

7. ANEXO

Código disponível em: https://github.com/luis951/AVRMusicPlayer