

## **MULTIPLEXAÇÃO DE DISPLAYS LEDS**

Por : Eng. Antonio Paulo Hawk

### **PARTE 1 - DISPLAYS DE LEDS DE 7 SEGMENTOS**

Uma coisa que todo mundo quer fazer, assim que toma contato com alguma linguagem de programação, é ligar um display de 7 segmentos ( ou mais de um ) em um microcontrolador.

Um display de 7 segmentos nada mais é do que 7 ( na verdade 8 por causa do ponto decimal ! ) Leds que podem ter os anodos ( ANODO COMUM ) ou os catodos ( CATODO COMUM ) ligados juntos, e as outras pontas desses Leds estão disponíveis independentemente umas das outras.

Assim, podemos ligar qualquer um desses Leds, bastando circular uma corrente neles, que geralmente vai de 5 mA a 20 mA . Neste caso, essa corrente vai circular durante todo o tempo em que o Led correspondente no display estiver acionado.

Primeiro, vamos ver como que definimos a corrente que vai atravessar o Led.

### **RESISTOR LIMITADOR – COMO DIMENSIONAR**

A maneira mais fácil é utilizar um resistor limitador, então vamos começar usando um display do tipo catodo comum, na cor vermelha.

Note que eu deixei claro uma coisa: cor vermelha! Por que isso é importante?

Existe no mercado displays nas cores vermelha verde, azul e branca!

Cada uma dessas cores implica que a tensão de condução do Led assume um determinado valor, e é muito importante para o cálculo da corrente que vai circular no Led, a qual será limitada por um resistor!

O valor exato da tensão é encontrado no datasheet do Led, mas para uso do dia a dia usaremos uma tabela com valores mais comumente adotados. Estes valores são aproximados para uma corrente de 20 mA, pois a tensão varia conforme a corrente.

Veja esta tabela de tensões versus cor:

COR	TENSÃO DIRETA
Vermelha	1,7 Volts
Verde	2,1 Volts
Azul	3,5 Volts
Branca	3,4 Volts

Desta maneira, para calcular o valor do resistor a ser posto em série com o Led, basta fazer a seguinte conta:

$$R = V/I = (V_{out} - V_{led}) / 0,02$$

Você deve estar pensando que a tensão de saída de um microcontrolador alimentado com 5 V é também 5 V, como aprendeu nos cursos, não é ?

**Não é não !!!! Depende da corrente que passa por ela !**

Se você usar um Led com baixa tensão, como 1,7 Volts, não muda muito o resultado. Mas se você utilizar um Led azul, com 3,5 V de tensão de condução, vai perceber que seu resultado pode estar errado em até mais de 30% ! Por isso que é importante saber EXATAMENTE qual a tensão que teremos na saída!

Para determinar exatamente o valor de  $V_{out}$ , você tem de consultar o datasheet de seu microcontrolador..... Vou aqui utilizar o ATMEGA328P como exemplo, pois é hoje o microcontrolador mais utilizado pelos iniciantes devido ao seu farto uso nos Arduínos.

Consultando o gráfico na página 363, você verifica que quando alimentado com 5 V, para uma corrente de 20 mA a tensão de saída chega a 4,45 Volts, portanto vamos usar este valor.

## ATmega48PA/88PA/168PA/328P

Figure 29-69. ATmega88PA: I/O Pin Output Voltage vs. Source Current ( $V_{CC} = 3\text{ V}$ )

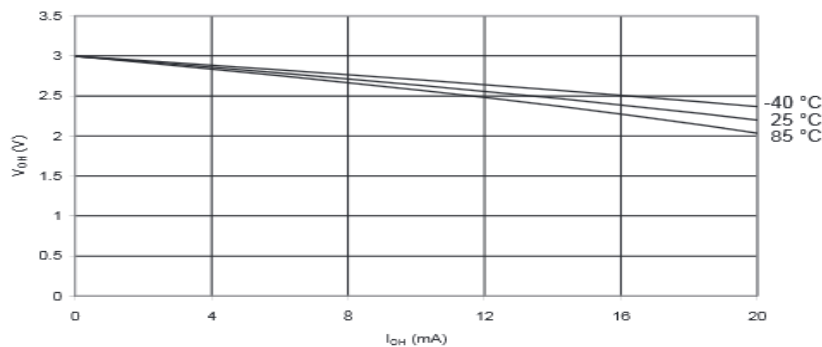
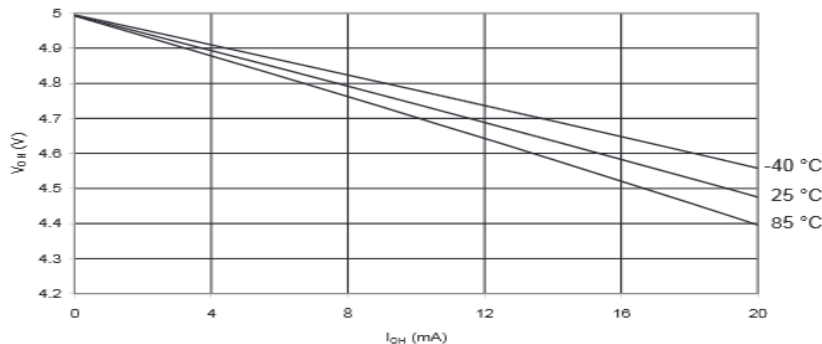


Figure 29-70. ATmega88PA: I/O Pin Output Voltage vs. Source Current ( $V_{CC} = 5\text{ V}$ )



Já se a corrente fosse de 10 mA, a tensão de saída será de 4,75 Volts.

Como exemplo prático, vamos supor que vamos usar um Led Vermelho, portanto usaremos um resistor de  $(4,45 - 1,7) / 0,02 = 137$  Ohms; neste caso usaremos o valor comercial de 150 Ohms.

Agora, temos aqui outro problema:

Imagine que você ligou os sete segmentos em seu microprocessador. Quando você quiser acender o número 8, a corrente que será fornecida pelo microcontrolador ao display será de  $8 \times 20\text{mA} = 160$  mA, que é uma corrente considerada alta para um microcontrolador.

Você terá de consultar novamente o datasheet e verificar se o seu modelo utilizado suporta essa corrente.

Veja o que temos na página 313 do datasheet:



## 28. Electrical Characteristics

### 28.1 Absolute Maximum Ratings\*

Operating Temperature .....	-55°C to +125°C
Storage Temperature .....	-65°C to +150°C
Voltage on any Pin except RESET with respect to Ground .....	-0.5V to $V_{CC}+0.5V$
Voltage on $\overline{\text{RESET}}$ with respect to Ground .....	-0.5V to +13.0V
Maximum Operating Voltage .....	6.0V
DC Current per I/O Pin .....	40.0 mA
DC Current $V_{CC}$ and GND Pins .....	200.0 mA

Aqui, vemos que o microcontrolador suporta tanto a corrente em cada pino (40 mA) como a corrente total ( 200 mA ). Podemos prosseguir sem nenhum problema.

Antes de montar nosso circuito, vou explicar uma outra maneira de se utilizar os Leds, que se chama Modo Pulsado.

Repare que até agora estamos falando de manter uma corrente constante, todo o tempo em que o Led estiver aceso. Isso não é obrigatório, podemos sim variar a corrente nele!

E desta maneira podemos fazer uma corrente maior passar por ele, desde que seja por pouco tempo.

Vamos ver um datasheet de um display Led para vermos quais informações o fabricante fornece:



### *Segment Digit LED Display*

## 1.2 Common Cathode 0.36 Inch (9.14mm)

PRODUCT DESCRIPTION
(1) 0.36 Inch (9.14mm) Digit Height
(2) Low current operation
(3) Excellent color and font characteristics
(4) Colors: White, blue, red, yellow and green
(5) Gray or black color background
(6) Common Cathode
(7) RoHs Compliant Part



### Absolute Maximum Rating (Ta = 25°C)

PARAMETER	RED	AMBER	GREEN	BLUE	WHITE	UNITS
DC Forward Current Per Segment	30	30	25	30	20	mA
Peak Current Per Segment <sup>(1)</sup>	70	50	50	25	25	mA
Avg. Forward Current (Pulse Operation) Per Segment	30	30	25	25	25	mA
Derating Linear From 25°C Per Segment	0.3					mA/°C
Reverse Voltage <sup>(2)</sup>	3					V
Operating Temperature	-25 to +85					°C
Storage Temperature	-30 to +85					°C

(1) Pulse conditions of 1/10 duty and 0.1msec width, for long operating life, max. of 20mA recommended

(2) Reverse biasing of the dot matrix is not recommend, will cause damage to the leds

Vamos considerar o caso do display vermelho ( RED ) . Olhe que interessante:

- Corrente máxima em modo contínuo: 30 mA
- Pico de corrente: 70 mA ( mais abaixo explicarei )
- Corrente média modo pulsado: 30 mA
- Condição de trabalho modo pulsado: duty de 1/10 e tempo do pulso de 0,1 milissegundos
- Caso você queira que seu display dure bastante, nunca passe a corrente de 30 mA....

No modo Pulsado, nosso display suporta um pico de corrente de 70 mA , DESDE QUE a duração dele seja de apenas 0,1 milissegundo, e que depois disso fique desligado por no mínimo 1 milissegundo ANTES DE SER LIGADO NOVAMENTE ( repare o duty de 1/10 , isto é, 1 período ligado e 10 períodos desligado ).

Agora, vamos identificar os segmentos:

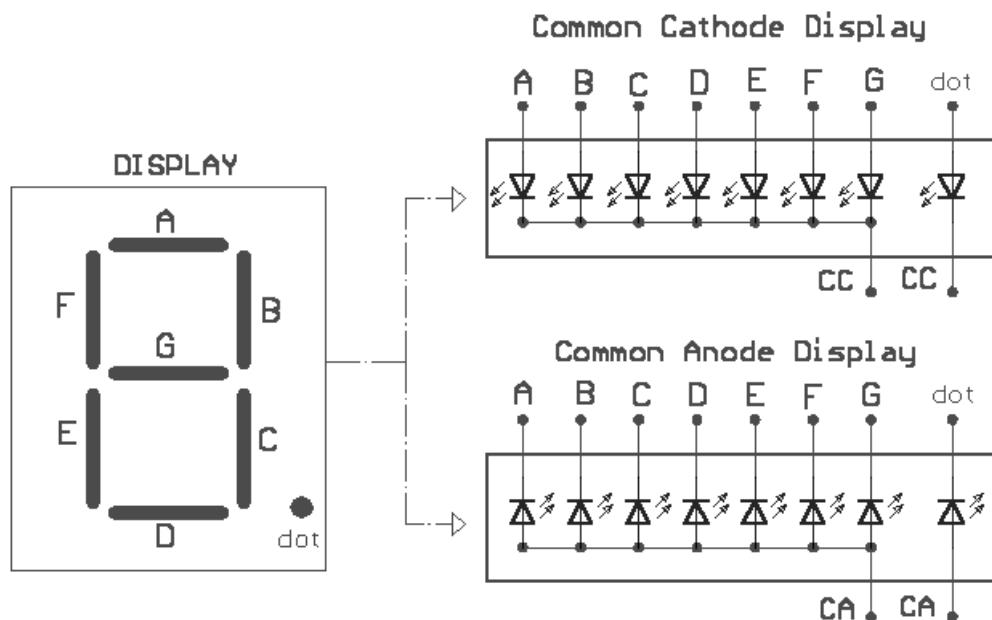


Fig.7-- Common Anode/Cathode DISPLAY Sam 6/02

A figura acima ilustra eletricamente as conexões internas para os casos de catodo comum e de anodo comum. O que é importante é ver as letras que identificam os segmentos: A , B , C, D, E, F, G .

*Agora, lembre-se de que o que virá abaixo vale para o display utilizado no exemplo, que é de CATODO COMUN, e, portanto, ele vai acender quando a saída for colocada no estado alto ( nível 1 ) .*

Para mostrar o número 1 , acendemos os segmentos B e C .

Para o número 7, acendemos A, B e C.

Para o número 8, acendemos todos os segmentos.

Simple, não é?

Em nosso programa, temos apenas de criar uma tabela que nos mostra quais segmentos temos de acender para formarmos o número que quisermos.

Para facilitar, repare que ligamos os segmentos em uma mesma porta do microcontrolador, e da seguinte maneira:

Bit 0 – A

Bit 1 – B e assim por diante até o Bit 6 – G.

Ou seja, utilizando binário, temos do MSB ao LSB: 0GFEDCBA

Portanto, basta utilizar uma única operação de escrita nos 8 bits do port utilizado, e podemos fazer acender qualquer número que desejarmos.

Assim, criaremos uma tabela na memória, informando quais os bits que precisam ser ligados para acender o número que quisermos! Onde o bit tiver o valor binário 1 significa que o segmento correspondente vai estar aceso ! É uma idéia bem simples.

DIGITO	Seg A	Seg B	Seg C	Seg D	Seg E	Seg F	Seg G	Binário
0	1	1	1	1	1	1	0	00111111
1	0	1	1	0	0	0	0	00000110
2	1	1	0	1	1	0	1	01011011
3	1	1	1	1	0	0	1	01001111
4	0	1	1	0	0	1	1	01100110
5	1	0	1	1	0	1	1	01101101
6	1	0	1	1	1	1	1	01111101
7	1	1	1	0	0	0	0	00000111
8	1	1	1	1	1	1	1	01111111
9	1	1	1	0	0	1	1	01100111

Portanto, para acendermos o numero 3 no display, basta colocarmos na saída do Port o numero 01001111b , que é o hexadecimal 4Fh . Agora, é só montarmos a nossa tabela.

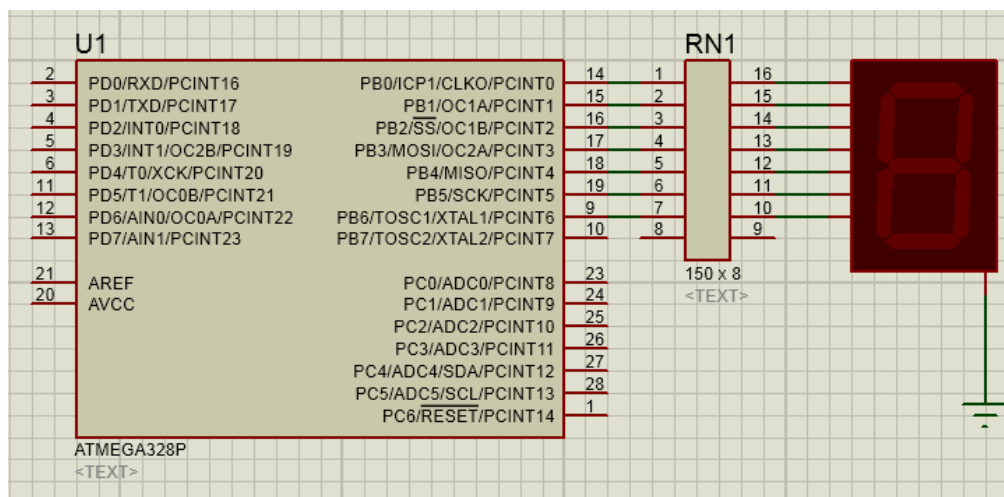
Apenas como curiosidade, se utilizássemos um display do tipo ANODO COMUM, como que ficaria a nossa tabela?

Um display do tipo anodo comum teria o terminal comum ligado ao + 5V, portanto o segmento iria acender apenas quando o nível da porta fosse colocado em nível baixo (0).

Na prática, pode utilizar a tabela acima, apenas INVERTENDO O NÍVEL DOS BITS! Ou seja, para acender o numero 3, na saída colocaremos o numero 10110000b , que é B0h em hexadecimal.

## APRESENTANDO O CIRCUITO

Vamos considerar o esquema abaixo com um display tipo Catodo Comum:



Não utilizaremos o ponto decimal para facilitar. Desta maneira, estamos utilizando 7 saídas para acionar os 7 segmentos, e ligamos o catodo comum direto ao terra ( GND ).

O componente marcado como RN1 é um tipo de CI , que contém apenas 8 resistores dentro. No mercado chama-se Resistor Array. Mas você pode utilizar resistores independentes se quiser!

Para fazer a programação, utilizarei a linguagem Basic, utilizando a versão gratuita do compilador BASCOM, muito utilizado por engenheiros e hobbystas em milhares de projetos.

Você pode baixar a versão free neste link:

[http://www.mcselec.com/index.php?option=com\\_docman&task=doc\\_download&gid=139&Itemid=54](http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=139&Itemid=54)

Com ela você compilar qualquer programa, a única limitação da versão free é que você não pode ultrapassar 4K de objeto a ser gravado. Mas apenas projetos grandes vão ultrapassar este tamanho, e não é o caso de nenhum projeto apresentado aqui.

Seque abaixo o programa, com os comentários na cor verde:

```
' Programa DISPLAY1
' Faz um contador e mostra a saída em um
' display de 7 segmentos tipo catodo comum
,

$regfile = "m328pdef.dat"
'usa um ATMEGA328P

$crystal = 8000000
'usando clock interno

$hwstack = 40
$swstack = 16
$framesize = 32

Config Portb = Output
' vamos usar o PortB como saída digital

Dim I As Byte
'criamos uma variável I do tipo BYTE

Dim Saida As Byte
'criamos uma variável SAIDA do tipo Byte

Saida = 0
' inicialmente vamos garantir o display apagado

Portb = Saida
'apaga o display

Do
' vamos fazer um loop todo o tempo
```

```

For I = 0 To 9

    Saida = Lookup(i , Tabela)
    ' procura o valor correspondente ao display na tabela
    ' e armazena na variável SAIDA

    Portb = Saida
    ' coloca os segmentos correspondentes ao número em I na saída

    Wait 1
    ' espera 1 segundo antes de trocar

Next I

Loop
'volta ao início do programa

End

```

```

Tabela:
' aqui está a tabela com a correspondencia dos segmentos a acender

Data &H3F , &H06 , &H5B , &H4F , &H66
Data &H6D , &H7D , &H07 , &H7F , &H67

```

Este programa, depois de compilado, tem o tamanho “enorme” de 370 bytes apenas.

Basta compilar, gerar o .hex e gravar no seu microcontrolador Atmega328P.

Quando energizar, o programa vai começar a mostrar os dígitos de 0 a 9, trocando a cada 1 segundo.

O que o programa faz é criar um contador, que vai de 0 até 9, e consulta uma tabela de equivalência, que vai retornar o valor necessário para acender o display e formar o número que desejamos mostrar.

Devido a isto ser muito simples, vamos passar para o próximo tópico, que é a Multiplexação.

## **MULTIPLEXAÇÃO EM DISPLAYS DE 7 SEGMENTOS**

Já vimos que tivemos de utilizar 7 pinos de saída do nosso microcontrolador para acender um único dígito.



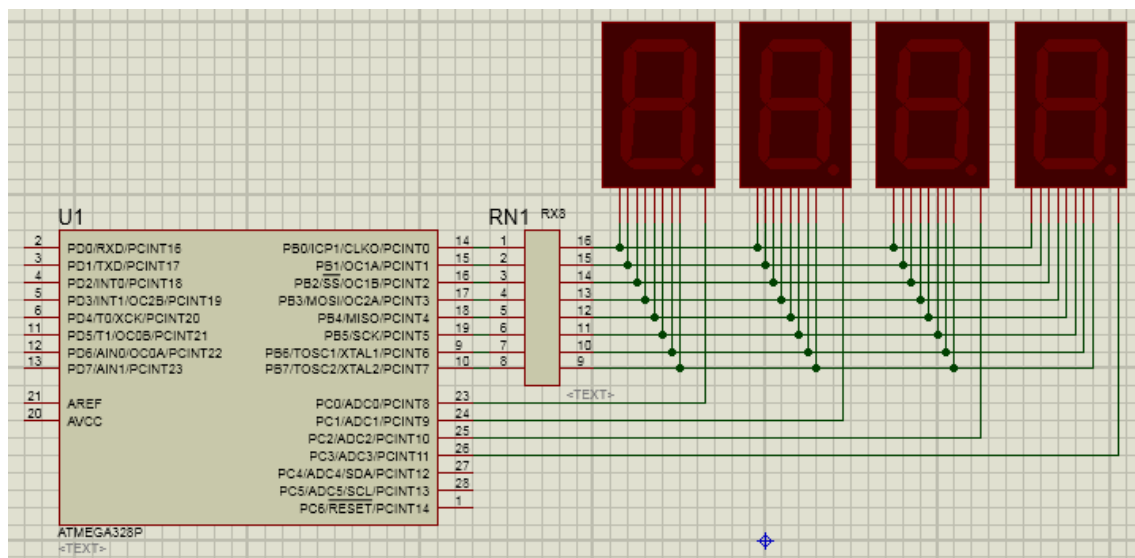
Mas, e se o display tiver, por exemplo, 4 dígitos ? Se fizermos o mesmo processo, e ainda tivermos de utilizar o ponto decimal em todos eles, iremos precisar de 32 pinos de saída! E imagine a corrente que vai passar, seria da ordem de  $4 \times 160 \text{ ma} = 640 \text{ mA}$ , totalmente impossível para qualquer microcontrolador.

Para contornarmos este problema, usamos uma técnica em que apenas um display está aceso em qualquer tempo! Acendemos primeiro o display numero 1, apagamos ele, e em seguida acendemos o display numero dois, apagamos ele ... e assim sucessivamente até acender o quarto display, quando então apagamos ele e voltamos a acender o primeiro novamente.

Se fizermos isto bem rápido, os nossos olhos terão a impressão de que todos estão acesos ao mesmo tempo! É o mesmo efeito que existia na TV com tubo, e no cinema também, que nada mais é de mostrar apenas uma parte da imagem de cada vez, e se repetimos isso pelo menos 30 vezes por segundo, nossos olhos não vão perceber que estamos fazendo a troca, e vão ser enganados, tendo a ilusão de que todos estão sempre acesos. Simples, não é?

Para conseguirmos isto, vamos usar apenas 8 pinos do microcontrolador, que irão ser ligados em todos os 4 displays no mesmo segmento ( incluindo o ponto decimal ), e mais 4 pinos, onde controlaremos qual o display que irá acender. Totalizamos assim 12 pinos, que é bem fácil de ter em muitos microcontroladores de hoje em dia.

A primeira idéia é fazer algo como o esquema abaixo:



Repare que aqui utilizamos um port inteiro para os 8 segmentos, e mais 4 pinos de outro port para seleccionar qual dígito irá acender. Usamos novamente uma rede de resistores, que parece um Circuito integrado de 16 pinos, mas que contém dentro 8 resistores apenas. Facilita muito o desenho do circuito!

Agora, repare bem, quando um determinado dígito estiver selecionado, suponha que vamos acender o numero 8 mais o ponto decimal. São 9 pinos do Port B fornecendo a corrente, e do jeito que está o esquema, existe um pino do outro Port C que recebe toda essa corrente somada ! Mas....

Lembra das limitações de corrente? Nenhum pino suporta mais do que 40 mA ! Portanto, teremos de limitar a corrente em cada segmento para um máximo de  $40/9 = 4,5$  mA, ou correremos o risco de queimar o nosso microcontrolador.

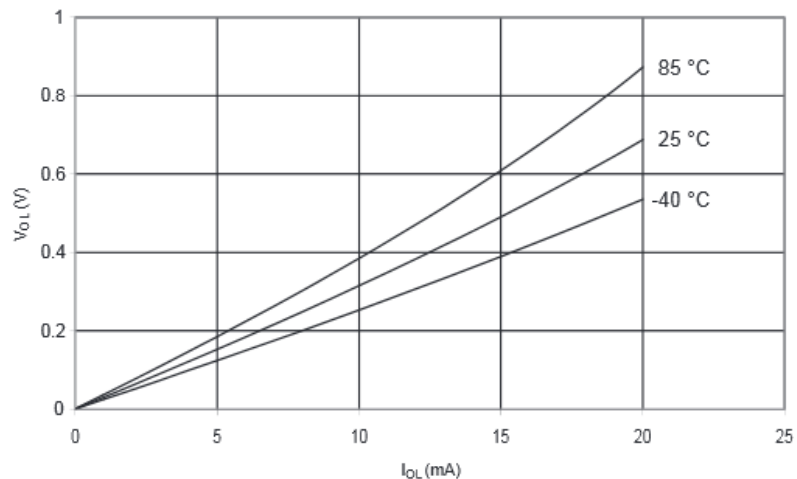
Podemos fazer o que já fizemos acima no esquema anterior, que é usar um resistor maior para fazer essa limitação. Para isso, consultamos outra vez o datasheet que mostra os níveis de tensão versus a corrente de saída, mas agora vamos precisar de outra informação, pois o pino que vai receber toda a corrente vai ser levado para o nível lógico 0, e deveria estar com 0 Volts, mas devido ao excesso de corrente ele vai ter uma tensão um pouco maior.

Vamos ver agora o comportamento da tensão de saída em nível baixo:

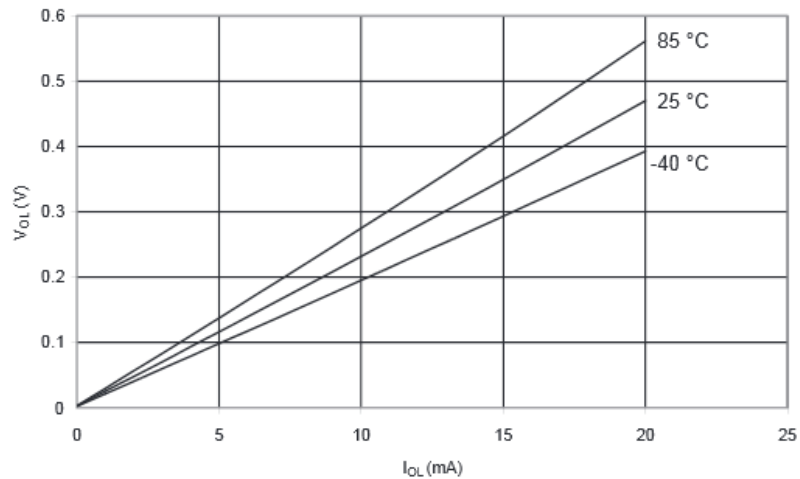
## **ATmega48PA/88PA/168PA/328P**

### **29.4.8 Pin Driver Strength**

**Figure 29-159.** ATmega328P: I/O Pin Output Voltage vs. Sink Current ( $V_{CC} = 3$  V)



**Figure 29-160.** ATmega328P: I/O Pin Output Voltage vs. Sink Current ( $V_{CC} = 5$  V)



E agora ??? Não existe a informação que queremos para 40 mA. Qual o motivo? O fabricante não quer que usemos essa corrente, pois isso vai diminuir muito a vida útil do componente!

Vamos fazer um exemplo teórico apenas. Extrapolando o gráfico, veremos que para 40 mA teremos uma tensão em torno de 0,9 Volts.

Agora, para calcular o resistor, temos de levar em consideração a queda de tensão do pino que fornece a corrente e do pino que recebe a corrente!

Vamos pesquisar a queda para um pino que em nível alto fornece 4,5 mA . Este gráfico já foi apresentado antes, e fazendo a consulta nele, teremos uma tensão de 4,9 Volts.

A tensão sobre o resistor seria de  $4,9 - 0,9 - 1,7 = 2,3$  /  $0,0045 = 511$  ohms. Usaremos o valor comercial mais próximo acima, que é de 560 ohms.

Na teoria, nosso circuito vai funcionar, dentro dos limites, mas temos um problema: como abaixamos muito a corrente de cada segmento, ele vai ficar com um brilho bem fraco.

Existem modelos especiais de displays de alto brilho, que mesmo com baixa corrente permite um brilho bem razoável, mas estes modelos são caros.

***Ainda nem levamos em conta um dos efeitos da Multiplexação: o brilho aparente, para quatro displays, será dividido por 4, o que tornará ainda mais fraco o brilho de cada dígito.***

Qual a solução para aumentarmos essa corrente através dos Leds do Display?

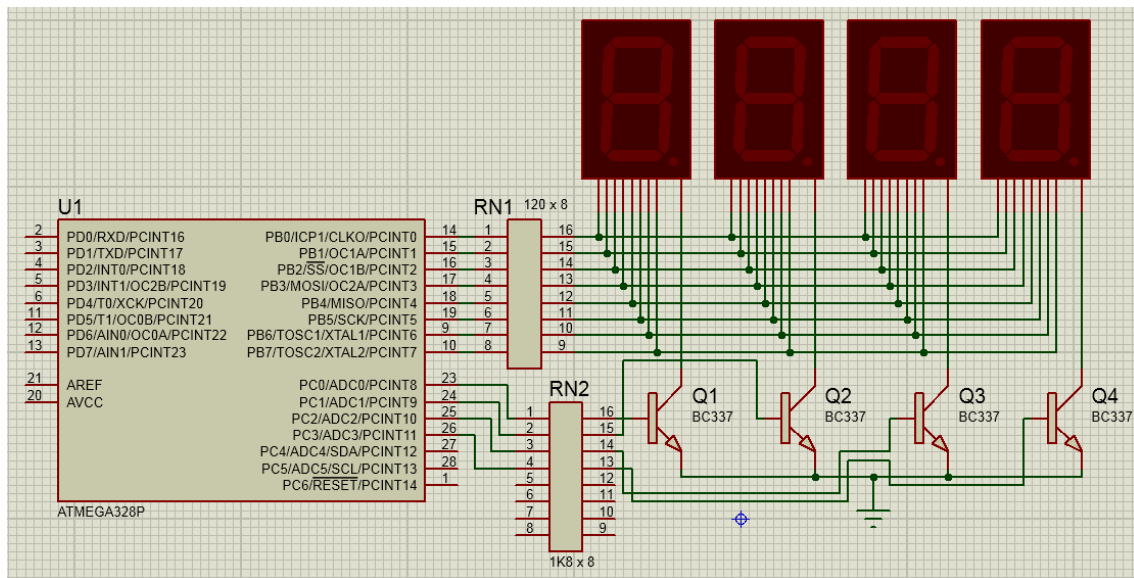
Em vez de ligarmos direto o terminal comum do display em um microcontrolador, vamos utilizar um transistor para receber essa corrente, pois ele suporta muito mais corrente!

Desta maneira, poderemos ainda fornecer o limite de corrente em cada um dos 8 pinos, o que vai permitir um brilho bem maior !

Agora, nossa única limitação será o total da corrente que o microcontrolador consegue fornecer no total. Se quisermos fornecer ainda mais corrente para o display, será necessário utilizar um CI tipo Source de corrente, como o UDN2981A, entre o microcontrolador e o Display. E nesse caso, lembre-se de levar em conta a tensão VceSat em vez da tensão de saída no pino do microcontrolador quando você for calcular o resistor limitador.

*No nosso programa, a diferença será a de que antes, para se acender o display, teríamos de levar esse pino por onde passa toda a corrente ao nível 0 , e agora, teremos de levar ao nível 1, para que o transistor possa conduzir.*

Veja o exemplo abaixo:



Repare que usamos um transistor comum BC337, ligado ao nosso microcontrolador através de um resistor de 1K8 para limitar a corrente de base. Os resistores dos segmentos tiveram seus valores alterados para 120 Ohms.

Como que fizemos esses cálculos?

Se cada pino vai fornecer a corrente de 20 mA, a tensão sobre o resistor é dada pela tensão no pino de saída do microcontrolador, menos a tensão Vce saturação do transistor. Vamos ao datasheet do BC337-25 :

- Ganho entre 160 e 400
- Corrente máxima de coletor 800 mA
- Tensão de saturação Vce Sat a 100 mA = 0,2 V com corrente de base = 1 mA

Agora, aqui começamos a ter de pensar...é melhor sempre olhar os valores a partir dos gráficos , pois representam muitas variações interessantes !

Podemos sempre escolher melhor os nossos parâmetros, pois basta localizarmos nos gráficos qual a situação que mais se assemelha ao uso que estamos querendo fazer, e pegar valores bem mais corretos.

Primeiro, a folha do datasheet que nos interessa muito:

## BC337, BC337-25, BC337-40

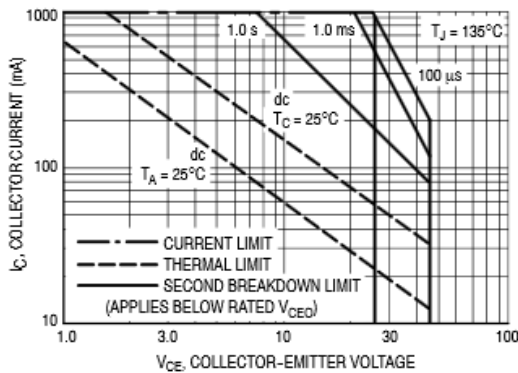


Figure 2. Active Region - Safe Operating Area

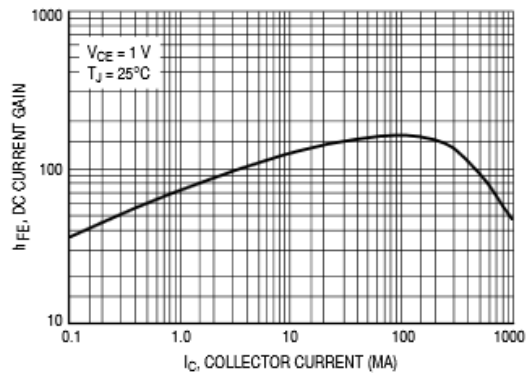


Figure 3. DC Current Gain

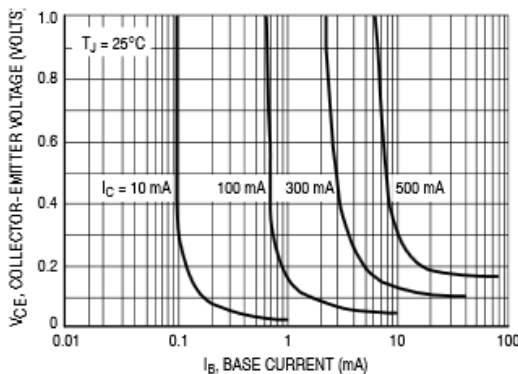


Figure 4. Saturation Region

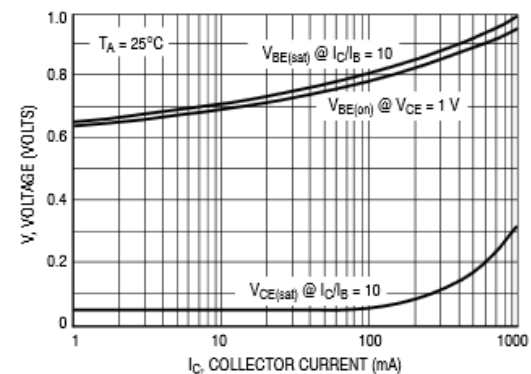


Figure 5. "On" Voltages

Normalmente, quando queremos ligar um relé, ou acender um Led, queremos levar o transistor à saturação. Assim, basta fornecer na base uma corrente maior do que 1 mA e garantimos que a tensão entre o coletor e o emissor será bem baixa, em torno de 0,2 Volts.

Mas o que acontece com um transistor quando está muito saturado? Ele vai demorar muito mais tempo para cortar a corrente de coletor quando a corrente de base é levada a zero!

Traduzindo, quando quisermos cortar o transistor, ele vai continuar conduzindo por um tempo, e esse tempo a mais depende de quanto maior é a relação entre a corrente de coletor e a corrente de base!

Na prática, acontece o seguinte:

Se quisermos fazer o Multiplex em alta velocidade, tipo cada display acender pelo menos 30 vezes por segundo ( o que vai dar uma frequência de Multiplex de 120 Hertz para os quatro dígitos ), o tempo em que cada display deveria ficar aceso acaba aumentando, e muitas vezes o dígito anterior ainda vai ter um pouco de brilho quando o próximo dígito acender! E isso faz com que todos os segmentos pareçam acesos ao mesmo tempo, ou com que aqueles que deveriam estar sempre apagados apareçam com um brilho pequeno, mas suficiente para atrapalhar a leitura de nosso display!

Como evitar este efeito? Simples, com duas implementações:

- Evitar saturar muito o transistor
- Esperar um pequeno tempo entre apagar um display e acender o outro.

Vamos cuidar do transistor. Olhando novamente o gráfico da figura 4, vemos que se fornecemos uma corrente de base de 0,7 mA , já iremos saturar o transistor, e que vai apresentar um VCE de 0,3 Volts. Já da figura 5 tiramos a informação de que VBE será de 0,8 V nessa situação com 100 mA de corrente do coletor.

*Lembramos aqui que a corrente que vai atravessar esse transistor vai variar de 20 mA até 160 mA , portanto temos de considerar que a VBE pode ser um pouco maior, VCE pode ser um pouco maior, e a corrente de base teria de ser um pouco maior também !*

Vamos assim definir que a corrente de base mínima tem de ser pelo menos 2 mA, e que a VBE máxima pode ser de 0,8 V, e que VCE máxima será de 0,3 V.

$$R = (5 - 0,8)/0,002 = 2,1 \text{ Kohm}$$

Usaremos o valor imediatamente abaixo para garantir, que será de 1,8 Kohm.

E o resistor usado nos segmentos do Led?

$$R = (4,45 - 0,3 - 1,7)/0,02 = 122 \text{ ohms} . \text{ Vamos adotar o valor comercial de } 120 \text{ ohms}.$$

Esta foi a maneira que cheguei aos valores do esquema mostrado acima.

Uma análise mais profunda vai nos mostrar que quando apenas um segmento de um display estiver aceso, a corrente que vai passar pelo coletor será de apenas 20 mA, e como a corrente de base não muda, o transistor vai estar muito saturado, e vai demorar mais tempo para desligar esse segmento, o que vai atrapalhar bastante a visualização, pois vai existir uma situação com dois transistores acionados , um conduzindo muita corrente ( normal ) e um outro que vai conduzir bem pouco porque ainda não conseguiu cortar totalmente, e assim dois displays vão estar com segmentos acesos, um mais forte que o outro, tornando impossível entender os resultados mostrados.

**Assim, teremos de fazer de qualquer maneira a solução pelo programa, que é demorar um pouco entre desligar um dígito e acender o outro.**

Essa demora pode ser relativamente pequena. Uma experiência que fiz mostrou que nas condições de circuito acima, se eu esperar 30 microsegundos após desligar o segmento antes de acender o outro, já faz o display ficar perfeito.

Vamos aos exemplos, chega de teoria!

Usaremos um display de quatro dígitos. Poderemos mostrar qualquer numero entre 0000 e 9,999 .

## LÓGICA DO PROGRAMA DE MULTIPLEXAÇÃO

Vamos criar uma matriz tipo Byte para armazenar quatro bytes. Cada byte armazenará o número que queremos que seja mostrado no respectivo display. Assim, os valores desses bytes vão estar entre 0 e 9.

O nosso programa principal será o responsável por colocar os quatro números a serem mostrados dentro dessa matriz, bem como indicar qual o display que temos de acender o ponto decimal.

Fazendo desta maneira, o processo de fazer o Multiplex vai demorar apenas 33 microsegundos, isso já com o delay, usando o clock interno de 8 MHz no Atmega328P.

Como esse processo ocorre 120 vezes em 1 segundo, o tempo total perdido será de aproximadamente 4 milissegundos, ou seja, perdemos apenas 4% do tempo do processador para fazer todo o processo !

Criaremos uma rotina que vai ser acionada por uma interrupção a cada 8,33 milissegundos, que será responsável por tratar todo o processo do Multiplex. Para isso, vamos usar um dos Timers que temos no microcontrolador. Pode-se usar qualquer um deles! Eu usei o Timer0 de 8 bits no primeiro exemplo, e nos outros dois eu usei o Timer1 de 16 bits.

Lembra que logo acima eu expliquei sobre a frequência do Multiplex? Para uma boa visualização, cada segmento tem de acender pelo menos 30 vezes a cada segundo. Então, como usamos 4 displays, teremos de multiplicar isso por 4, obtendo 120 Hz.

Na prática, quanto maior essa frequência, mais “perfeito” fica o efeito, mas o brilho diminui conforme aumentamos a frequência.... Por que isso acontece?

Vamos usar aqui 120 Hz como exemplo. Se cada display tem de acender 30 vezes por segundo, a grosso modo seu brilho será reduzido a 1/30 do brilho que teria se ficasse aceso o tempo todo ! Se aumentarmos o Multiplex para 200 Hz, cada display acenderá por apenas 1/50 do normal....

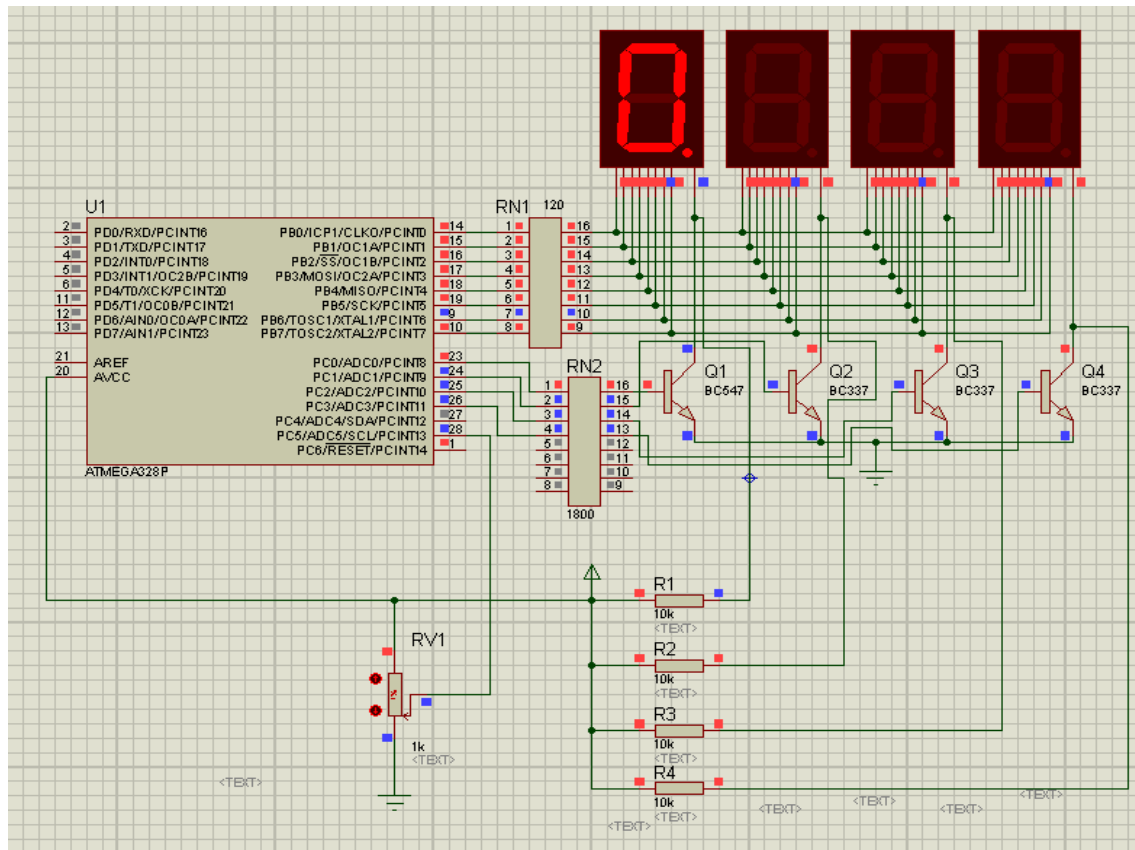
O nosso olho ajuda a não percebermos tanta variação, devido ao efeito da persistência da retina. Mas você percebe bastante a diminuição quando muda de 120 para 200 Hertz, pois os brilhos já estão bem fracos.

Então, vamos voltar ao nosso programa.

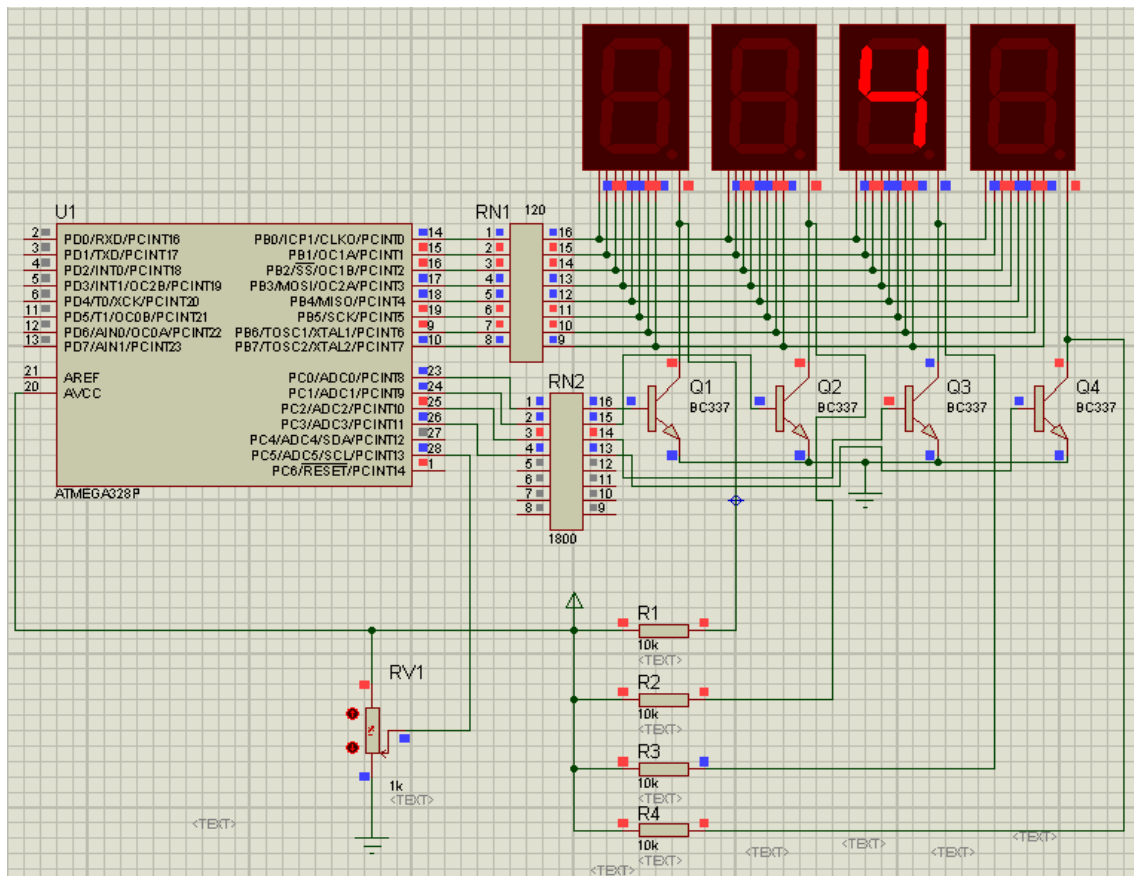
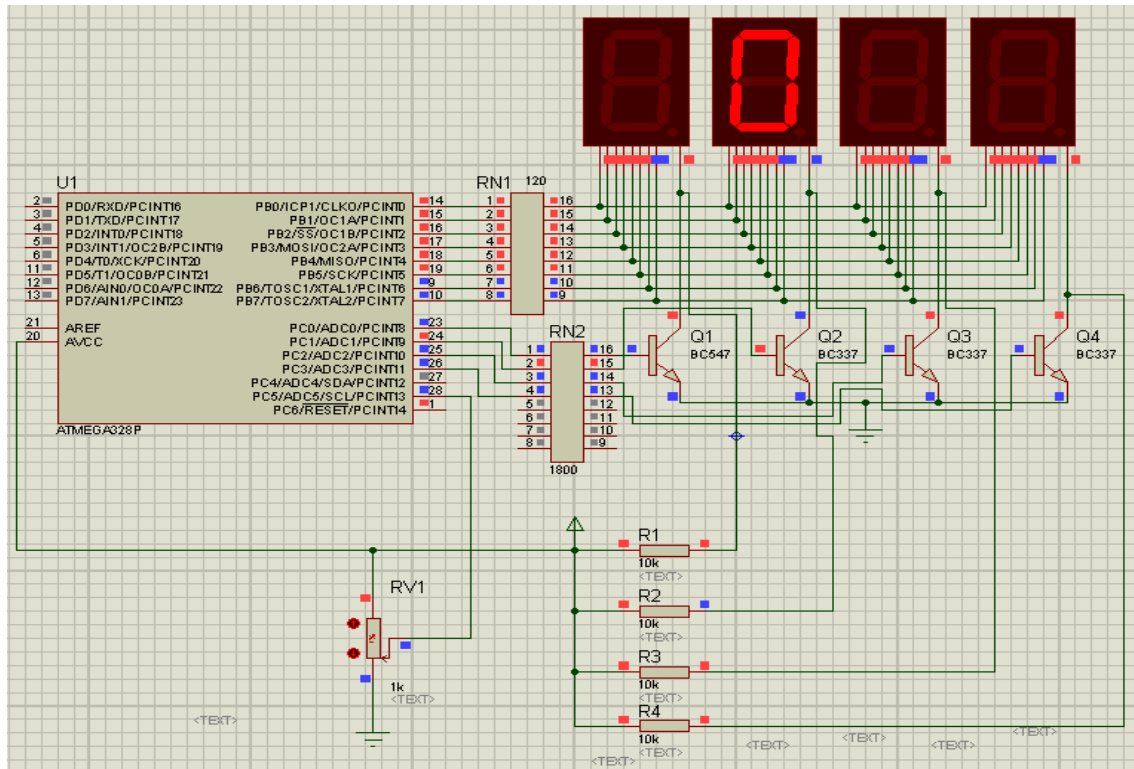
A cada interrupção, apagaremos imediatamente o display que estava aceso, e incrementamos um contador, que nos indica o próximo display a ser aceso. Sabendo qual é o dígito, pegamos o valor numérico que tem de ser mostrado nele, consultamos uma tabela que vai nos dizer quais os segmentos que teremos de acender, verificamos também se temos de acender o ponto decimal desse display, e aguardamos um pequeno tempo de 15 microsegundos, antes de ligar o display selecionado. Assim, eliminamos o problema que eu comentei bem lá em cima, sobre o transistor sair da saturação.

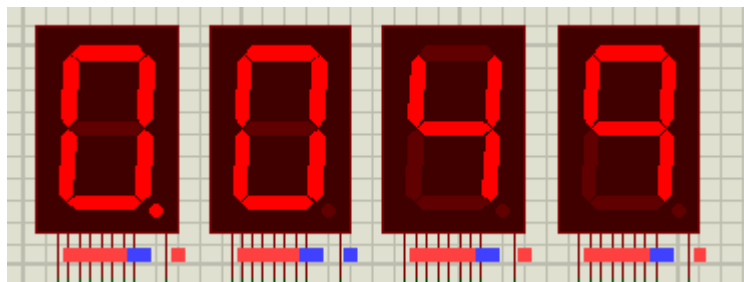
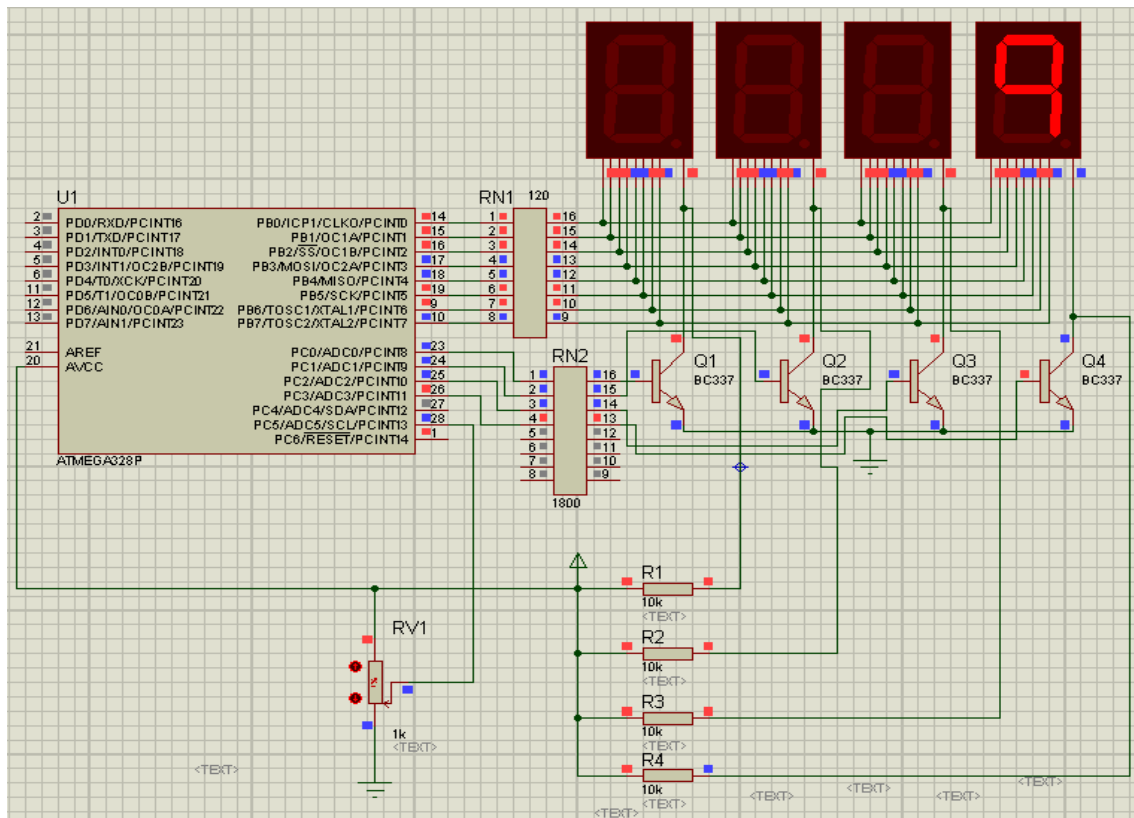
Quando chegarmos ao ultimo dos 4 dígitos, fazemos tudo de novo a partir do primeiro.

Segue a sequência abaixo, onde cada display é aceso sequencialmente (programa foi compilado com frequência de Multiplex de 4 hertz para as capturas), e por fim o uma captura feita em velocidade real. Repare que as cores vermelhas nos terminais dos componentes significam que existe nível 1, e as azuis que a tensão é nível 0. Assim fica fácil verificar todos os níveis em cada passo do programa. Apenas um aviso sobre os resistores R1 até R4 : Eles não são necessários no circuito real ! Mas são necessários para o correto funcionamento do simulador Isis do Proteus.









Simples, não é ?

Esse processo possui um pequeno inconveniente. Imagine que o seu programa principal calculou o numero a ser mostrado no display. Agora, imagine que já atualizou 3 dos 4 dígitos, e nesse momento acontece uma interrupção do timer, e vai justamente mostrar um dos novos dígitos já atualizados. Na prática, você perceberia que de vez em quando acontece algo esquisito, alguns “erros” piscam muito rápido no numero mostrado.

Para evitar isto, eu sugiro sempre o seguinte procedimento:

- Deixe para armazenar os números na matriz apenas quando todos estiverem prontos.
- **Antes de atualizar a matriz, pare o Timer !!!!** Com isso você garante que não acontecerá a interrupção. Quando terminar de armazenar o último dígito, libere novamente o Timer!

Creio que agora você já sabe tudo para fazer o seu próprio projeto com displays multiplexados.

Embora este pequeno tutorial foi baseado no Atmega328P, pode ser aplicado em qualquer microprocessador. O uso da Linguagem Basic do Bascom torna bem simples a compreensão e a migração para outras linguagens.

Um ultimo comentário sobre o delay necessário no programa. O delay pode ser maior ou menos, conforme a velocidade de processamento de seu programa. Nos exemplos aqui da apostila, usamos um AVR com 8 MHz. Se usarmos a velocidade interna de 1 MHz, temos de eliminar o comando de delay, pois já vai demorar bastante para executar a rotina de interrupção, em torno de 150 microsegundos. E se o clock for de 16 MHz, temos de aumentar o comando de delay para 24 microsegundos.

E se utilizar um Pic tipo 16f ou 18F ? É só lembrar que o processamento dele é 4 vezes menor que o de um AVR no mesmo clock.

## **PROGRAMAS EM BASCOM**

Todos os programas apresentados aqui compilam com sobras na versão Demo do Bascom. Disponibilizarei também os arquivos para a simulação no ISIS para cada um deles.

Apresentarei três programas, sendo o primeiro um mostrador que mostra uma contagem que vai de 0000 até 9999 e volta para 0000.

O segundo é uma versão mais elaborada do primeiro, pois ele oculta os zeros à esquerda (não significativos) da contagem.

O terceiro e último mostra como obter um voltímetro de 0 a 5 volts Dc.

### **PROGRAMA 1 – MOSTRADOR CONTADOR DE 0000 ATÉ 9999**

Segue o primeiro programa, que depois de compilado tem o tamanho de 826 bytes.

```
'-----  
' PROGRAMA MULTIPLEX1 - Implementa o Multiplex com 4 dígitos, e  
' um contador para ficar alterando os numeros a serem mostrados.  
' Mostra semelhante a um multímetro, mostrando sempre 4 dígitos acesos  
' por exemplo, quando é 0 mostrará 0000 .  
' Este programa utiliza o Timer0 de 8 bits para a base de tempo  
' e implementa um refresh de 120 Hertz aproximado.  
'-----
```

```
$crystal = 8000000
$regfile = "m328pdef.dat"
$hwstack = 40
$swstack = 16
$framesize = 32
```

```
Dim Digitos(4) As Byte
Dim Disp_index As Byte
Dim Disp_aux As Byte
Dim Disp_flag As Bit
Dim Dp_position As Byte
```

```
Config Portb = Output
' PortB vai acender os segmentos, usaremos todos como saída.
```

```
Config Portc.0 = Output
Config Portc.1 = Output
Config Portc.2 = Output
Config Portc.3 = Output
' portc comanda qual display será aceso. usamos os bits PortC.0 até PortC.3
' assim configuramos apenas os 4 bits como saída.
```

```
Digitos(1) = 11
Digitos(2) = 11
Digitos(3) = 11
Digitos(4) = 11
Disp_index = 0
Dp_position = 0
```

```
Config Timer0 = Timer , Prescale = 1024
Timer0 = 191
' vamos gerar uma interrupção a cada 8,32 milissegundo
' que corresponde a uma frequencia de Multiplex de 120,2 Hertz
' ou seja, cada display vai acender 30 vezes por segundo !
```

```
On Timer0 Timer0_sub
Enable Timer0
Enable Interrupts
```

```
Wait 1
' todos os segmentos vao ficar acesos por 1 segundo
' util para ver se algum foi danificado!
Digitos(1) = 0
Digitos(2) = 0
Digitos(3) = 0
Digitos(4) = 0
' começaremos do numero 0000
```

```
Do
    Waitms 100
    Incr Digitos(4)
    If Digitos(4) > 9 Then
        Digitos(4) = 0
```

```

    Incr Digitos(3)
End If
If Digitos(3) > 9 Then
    Digitos(3) = 0
    Incr Digitos(2)
End If
If Digitos(2) > 9 Then
    Digitos(2) = 0
    Incr Digitos(1)
End If
If Digitos(1) > 9 Then
    Digitos(1) = 0
End If
Dp_position = 4
Loop

```

```
End
```

```

'----- ROTINAS DE INTERRUPÇÃO -----
Timer0_sub:
' Rotina chamada pelo Timer0 a cada
Timer0 = 191
'recarrega o timer novamente para

Portc.0 = 0
Portc.1 = 0
Portc.2 = 0
Portc.3 = 0
' apaga todos os dígitos

If Disp_index > 3 Then Disp_index = 0
' vamos ver se já fizemos o ultimo digito, pois então teremos de
' começar pelo primeiro novamente

Incr Disp_index
' aqui já apontamos para o próximo digito

Disp_aux = Digitos(disposition)
' pegamos o numero que queremos mostrar
Portb = Lookup(disposition , Table_0f)

' verifica se temos de acender o ponto decimal também
If Disp_index = Dp_position Then
    Disp_aux = Lookup(12 , Table_0f)
    ' procura na tabela a posição do segmento
    Portb = Portb Or Disp_aux
    ' acende apenas o ponto decimal
End If

' agora vamos acender os segmentos correspondentes ao numero
Disp_aux = Lookup(disposition , Table_mux)
Waitus 15
Portc = Portc Or Disp_aux
' faz acender o dígito correto
Return

```

```
' TABELA DE SEGMENTOS A ACENDER
```

```
Table_of:
```

```
    Data &B00111111 , &B00000110 , &B01011011 , &B01001111 '0,1,2,3
    Data &B01100110 , &B01101101 , &B01111101 , &B00000111 '4,5,6,7
    Data &B01111111 , &B01100111 , &B00000000 , &B11111111 '8,9,NADA,TUDO
    Data &B10000000 'PONTO
```

```
' TABELA INDICADORA DO DISPLAY A ACENDER
```

```
Table_mux:
```

```
    Data &B00000000 , &B00000001 , &B00000010 , &B00000100 , &B00001000
```

## PROGRAMA 2 – CONTADOR SEM OS ZEROS À ESQUERDA

Segue o programa aperfeiçoado, que agora compilado tem 1248 bytes.

```
'-----
' PROGRAMA MULTIPLEX2 - Mostra uma maneira de apagar os dígitos
' que ficam à esquerda do numero mostrado, assim não veremos um
' monte de 0 sem nenhum significado !
' Agora usaremos o Timer1 de 16 bits só para variar .....
' experimente alterar o valor da constante Taxa e veja que legal !
'-----

$crystal = 8000000
'vamos usar o clock interno que é de 8 Mhz.
$regfile = "m328Pdef.dat"
' vamos compilar para o ATMEGA328P
$hwstack = 40
$swstack = 16
$framesize = 32

Dim Digitos(4) As Byte
' nosso display terá 4 dígitos
Dim Disp_index As Byte
Dim Disp_aux As Byte
Dim Disp_flag As Bit
Dim Dp_position As Byte
Dim Numero As Word
Dim Str_valor As String * 5
Dim I As Byte
Dim Ndigit As Byte
Dim Digit_temp(6) As Byte

'----- valores para ajustar a velocidade do Mux -----
Const Taxa = 65471
' se quiser ver o flicker, mude para 65341, e se quiser ver
' a mudança uma a uma, use 62932
' valor normal do programa = 65471
```

```

Config Portb = Output
' PortB vai acender os segmentos, usaremos todos como saída.

Config Portc.0 = Output
Config Portc.1 = Output
Config Portc.2 = Output
Config Portc.3 = Output

' portc comanda qual display será aceso. usamos os bits PortC.0 até PortC.3
' assim configuramos apenas os 4 bits como saída.


Digitos(1) = 11
Digitos(2) = 11
Digitos(3) = 11
Digitos(4) = 11
' com o valor 11, vamos acender todos os segmentos
Disp_index = 0
Dp_position = 0


Config Timer1 = Timer , Prescale = 1024
Timer1 = Taxa
' vamos gerar uma interrupção a cada 8,32 milissegundo
' que corresponde a uma frequencia de Multiplex de 120,2 Hertz
' ou seja, cada display vai acender 30 vezes por segundo !

On Timer1 Timer1_sub
Enable Timer1
Enable Interrupts

Wait 1
' todos os segmentos vao ficar acesos por 1 segundo
' util para ver se algum foi danificado!
Digitos(1) = 0
Digitos(2) = 0
Digitos(3) = 0
Digitos(4) = 0

Numero = 0
' começaremos do numero 0000

Do
    Waitms 100
    Incr Numero
    If Numero = 10000 Then Numero = 0
    Gosub Acerta

Loop

'----- sub-rotina acerta -----
' esta rotina faz com que os zeros à esquerda do número sejam apagados
' a apresentação do número fica muito melhor.

```

```

'
Acerta:
Str_valor = Str(numero)
' vamos converter o numero desejado em uma string de caracteres

Str2digits Str_valor , Digit_temp(1)
'esta função transforma os dígitos dentro da string em números, onde cada um
'dos dígitos será armazenado em uma posição de uma matriz numérica.
' é muito poderosa e facilita bastante nosso trabalho

Ndigit = Digit_temp(1)
'agora Sabemos Quantos Dígitos Possui O Número
'e só precisamos recolocar na ordem que o nosso programa vai apresentar
'pois na conversão a matriz foi criada com os os numeros de maior grandeza
'no inicio da matriz e daí em diante conforme abaixa a grandeza
'sugiro examinar o help do Bascom para entender o funcionamento.

Stop Timer1
' Vamos parar a contagem do timer1, para evitar que haja alguma interrupção
' enquanto preparamos todos os novos digitospois poderia haver alguma
' mudança no display, então paramos a contagem
' e não teremos a interrupção do timer1 !

For I = 1 To 4
    Digitos(i) = 10
Next I
' já colocamos o caractere que vai apagar todos os 4 dígitos
' agora é só reordenar conforme a quantidade de dígitos presente
' no nosso número a ser mostrado.

If Ndigit = 4 Then
    Digitos(1) = Digit_temp(5)
    Digitos(2) = Digit_temp(4)
    Digitos(3) = Digit_temp(3)
    Digitos(4) = Digit_temp(2)
Elseif Ndigit = 3 Then
    Digitos(2) = Digit_temp(4)
    Digitos(3) = Digit_temp(3)
    Digitos(4) = Digit_temp(2)
Elseif Ndigit = 2 Then
    Digitos(3) = Digit_temp(3)
    Digitos(4) = Digit_temp(2)
Else
    Digitos(4) = Digit_temp(2)
End If

Start Timer1
' pronto, vamos liberar o timer1 novamente pois tudo já está pronto !
' este trecho todo acaba atrasando apenas 18 microsegundos, que não chega
'a ser 3% do tempo normal de cada interrupção, portanto imperceptível.
Return

End

'----- ROTINAS DE INTERRUPÇÃO -----
Timer1_sub:
' Rotina chamada pelo Timer1 a cada 8,3 milisegundos
Timer1 = Taxa

```



```

' recarrega o timer novamente

Portc.0 = 0
Portc.1 = 0
Portc.2 = 0
Portc.3 = 0
' apaga todos os dígitos

If Disp_index > 3 Then Disp_index = 0
' vamos ver se já fizemos o ultimo digito, pois então teremos de
' começar pelo primeiro novamente

Incr Disp_index
' aqui já apontamos para o próximo digito

Disp_aux = Digitos(displ_index)
' pegamos o numero que queremos mostrar

Portb = Lookup(displ_aux , Table_0f)
' pegamos a equivalencia dos segmentos na tabela

' Agora verificamos se temos de acender o ponto decimal também
  If Disp_index = Dp_position Then
    Disp_aux = Lookup(12 , Table_0f)
    ' procura na tabela a posição do segmento
    Portb = Portb Or Disp_aux
    ' acende apenas o ponto decimal
  End If

' agora vamos acender os segmentos correspondentes ao numero
Disp_aux = Lookup(displ_index , Table_mux)
Waitus 15
Portc = Portc Or Disp_aux
' faz acender o dígito correto

Return

' -----
' TABELA DE SEGMENTOS A ACENDER

Table_0f:
  Data &B00111111 , &B00000110 , &B01011011 , &B01001111 '0,1,2,3
  Data &B01100110 , &B01101101 , &B01111101 , &B00000111 '4,5,6,7
  Data &B01111111 , &B01100111 , &B00000000 , &B11111111 '8,9,NADA,TUDO
  Data &B10000000 'PONTO

' -----
' TABELA INDICADORA DO DISPLAY A ACENDER

Table_mux:
  Data &B00000000 , &B00000001 , &B00000010 , &B00000100 , &B00001000

```

## PROGRAMA 3 – VOLTÍMETRO DIGITAL ATÉ 5 VOLTS

Segue o programa do voltímetro digital, compilado com 2332 bytes.

```
'-----  
' PROGRAMA MULTIPLEX3 - Implementa um voltímetro que pode medir  
' tensão entre 0 e 5 volts, e mostra com até 3 casa depois da vírgula  
'  
'  
'-----  
  
$crystal = 8000000  
'vamos usar o clock interno que é de 8 Mhz.  
$regfile = "m328Pdef.dat"  
' vamos compilar para o ATMEGA328P  
$hwstack = 40  
$swstack = 16  
$framesize = 32  
  
Dim Disp_index As Byte  
Dim Disp_aux As Byte  
Dim Disp_flag As Bit  
Dim Digitos(4) As Byte  
' nosso display terá 4 dígitos  
Dim Dp_position As Byte  
Dim Numero As Word  
Dim Str_valor As String * 5  
Dim I As Byte  
Dim Digit_temp(6) As Byte  
Dim Resultado As Single  
' RESULTADO tem de apresentar numeros não inteiros  
  
'----- valores para ajustar a velocidade do Mux -----  
Const Taxa = 65471  
' se quiser ver o flicker, mude para 65341, e se quiser ver  
' a mudança uma a uma, use 62932  
' valor normal do programa = 65471  
  
Config Portb = Output  
' PortB vai acender os segmentos, usaremos todos como saída.  
  
Config Portc.0 = Output  
Config Portc.1 = Output  
Config Portc.2 = Output  
Config Portc.3 = Output  
  
' portc comanda qual display será aceso. usamos os bits PortC.0 até PortC.3  
' assim configuramos apenas os 4 bits como saída.  
  
  
Digitos(1) = 11  
Digitos(2) = 11
```

```

Digitos(3) = 11
Digitos(4) = 11
' com o valor 11, vamos acender todos os segmentos
Disp_index = 0
Dp_position = 0

Config Timer1 = Timer , Prescale = 1024
Timer1 = Taxa
' vamos gerar uma interrupção a cada 8,32 milissegundo
' que corresponde a uma frequencia de Multiplex de 120,2 Hertz
' ou seja, cada display vai acender 30 vezes por segundo !

On Timer1 Timer1_sub
Enable Timer1
' aqui definimos a subrotina que irá ser chamada a cada
' interrupção, e já deixamos o timer1 correr

Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc
' aqui configuramos o conversor a/d para fazer uma leitura
' apenas quando pedirmos, vai usar como referencia 5V

Enable Interrupts
' aqui habilitamos as interrupções definidas no programa

Wait 1
' todos os segmentos vao ficar acesos por 1 segundo
' util para ver se algum foi danificado!
Digitos(1) = 0
Digitos(2) = 0
Digitos(3) = 0
Digitos(4) = 0

Dp_position = 1
' ponto decimal sempre no primeiro dígito

Do
    Waitms 100
    Numero = 0
    For I = 1 To 32
        Numero = Numero + Getadc(5)
    Next I
    Shift Numero , Right , 5
    'fizemos a soma de 32 leituras consecutivas, e calculamos
    'a média para ter uma melhor estabilidade

    Numero = Numero * 5
    Resultado = Numero / 1023
    ' aqui fizemos o chamado fator de escala, pois o fundo de escala
    ' é 5 volts e nesse caso obteremos leitura de 1023, então
    ' basta multiplicar por 5 e dividir por 1023, e já teremos o
    ' valor em volts.

    Str_valor = Fusing(resultado , "#.###")

```

```

' convertemos o resultado em uma string já formatada como queremos
' e com arredondamento na última casa decimal ! Mais uma
' poderosa instrução do Bascom

Str_valor = Trim(str_valor)
' eliminamos qualquer espaço em branco

Str2digits Str_valor , Digit_temp(1)
' fazemos o mesmo truque em tirar todos os dígitos da string e
' armazenar em uma matriz como número, assim fica muito simples
' pegar cada dígito e apresentar.

Stop Timer1
' paramos a contagem do timer1 para evitar mostrar resultados
' sem nexos

If Digit_temp(1) = 3 Then
' o valor a mostrar é zero
  Digitos(1) = 0
  Digitos(2) = 0
  Digitos(3) = 0
  Digitos(4) = 0
Else
' temos de ordenar os dígitos, agora com uma novidade, pois
' digit_temp(5) tem o ponto decimal, então pularemos
  Digitos(1) = Digit_temp(6)
  Digitos(2) = Digit_temp(4)
  Digitos(3) = Digit_temp(3)
  Digitos(4) = Digit_temp(2)
End If
Start Timer1
' continua a contagem do Timer1
Loop

End

'----- ROTINAS DE INTERRUPTÃO -----
Timer1_sub:
' Rotina chamada pelo Timer1 a cada 8,3 milisegundos
Timer1 = Taxa
' recarrega o timer novamente

Portc.0 = 0
Portc.1 = 0
Portc.2 = 0
Portc.3 = 0
' apaga todos os dígitos

If Disp_index > 3 Then Disp_index = 0
' vamos ver se já fizemos o último dígito, pois então teremos de
' começar pelo primeiro novamente

Incr Disp_index
' aqui já apontamos para o próximo dígito

Disp_aux = Digitos(Disp_index)
' pegamos o número que queremos mostrar

```

```

Portb = Lookup(displ_aux , Table_0f)
' pegamos a equivalencia dos segmentos na tabela

' Agora verificamos se temos de acender o ponto decimal também

If Displ_index = Dp_position Then
Displ_aux = Lookup(12 , Table_0f)
' procura na tabela a posição do segmento

Portb = Portb Or Displ_aux
' acende apenas o ponto decimal
End If

' agora vamos acender os segmentos correspondentes ao numero

Displ_aux = Lookup(displ_index , Table_mux)
Waitus 15
Portc = Portc Or Displ_aux
' faz acender o dígito correto
Return

' -----
' TABELA DE SEGMENTOS A ACENDER

Table_0f:
Data &B00111111 , &B00000110 , &B01011011 , &B01001111 '0,1,2,3
Data &B01100110 , &B01101101 , &B01111101 , &B00000111 '4,5,6,7
Data &B01111111 , &B01100111 , &B00000000 , &B11111111 '8,9,NADA,TUDO
Data &B10000000 'PONTO

' -----
' TABELA INDICADORA DO DISPLAY A ACENDER

Table_mux:
Data &B00000000 , &B00000001 , &B00000010 , &B00000100 , &B00001000

```

Em anexo, você encontra os arquivos fontes dos programas, e os arquivos da simulação no Ísis do Proteus.

A idéia é você alterar a taxa do Multiplex, recompilar o programa, e ver na tela do simulador o funcionamento em velocidade reduzida. Você pode alterar a frequência do Timer para verificar o funcionamento em baixa velocidade, para ver o programa fazer acender os displays um a um.

Agora, você deve estar pensando: se eu precisar de um display com bastante brilho, como posso fazer?

Vamos agora conhecer um CI fantástico, com baixo custo e excelente desempenho e que pode entregar muita corrente (até queimar...) aos nossos displays de 7 segmentos ! Ele é o famoso MAX7219.

## **DISPLAYS DE 7 SEGMENTOS COM O MAX7219**

Você já percebeu que se quisermos ter um bom brilho, temos de entregar mais corrente aos segmentos. E para isto teremos de acrescentar um CI do tipo Current Drive ou vulgarmente Source Drive. Ele ficaria ligado entre as 8 saídas do microcontrolador e os 8 segmentos.

Até aqui, tudo bem. Mas, vamos complicar um pouco mais. Imagine que agora você quer fazer um controle automático de brilho, isto é, você quer poder mudar o brilho do display, talvez para uso com bateria, ou em locais escuros. Como fazer isto?

Podemos mudar os tempos envolvidos na Multiplexação, como por exemplo, aumentando bastante a frequência. Mas agora o efeito da saturação é o nosso novo limitador, pois logo vai parecer que todos os segmentos começam a acender.... e complica demais.

Para evitar isto, surgiu o MAX7219. Ele é um CI controlador de até 8 displays Leds do tipo catodo comum, e pode fornecer até 40 mA por segmento. Pode mudar o brilho com 16 níveis através de PWM feito no próprio CI. Pode utilizar de 1 até 8 displays, e pode ser reconfigurado por software. Você não precisa de tabelas, pode enviar para o CI o valor de cada dígito com 4 bits ( Code B ) e o CI se encarrega dos segmentos. Podemos também “cascatear” vários CI’s, podendo mostrar 16 ou mais dígitos. E o melhor de tudo, você precisa de apenas 3 pinos do microcontrolador para fazer tudo isto, pois a interface é serial, e de altíssima velocidade !

Este CI também é muito utilizado para se fazer matriz de Leds, pois ele pode controlar 64 Leds individuais, dispostos em uma matriz de 8 linhas por 8 colunas. Mas isto é um uso um pouco mais avançado e ficará para uma próxima oportunidade.

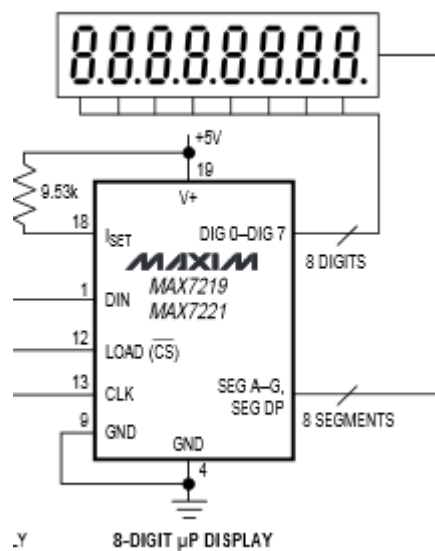
Como funciona a “conversa” com o MAX7219? Bem simples, sempre enviaremos 16 bits para ele de uma vez (ou seja, dois bytes consecutivos). Apenas 12 bits têm validade, sendo que 8 bits são referentes a dados, e 4 bits referentes a endereço de um registro.

Dentro do CI, podemos acessar até 16 registros individuais, sendo que 8 deles guardam os números a serem mostrados no display, e os outros 8 restantes são para diversas configurações.

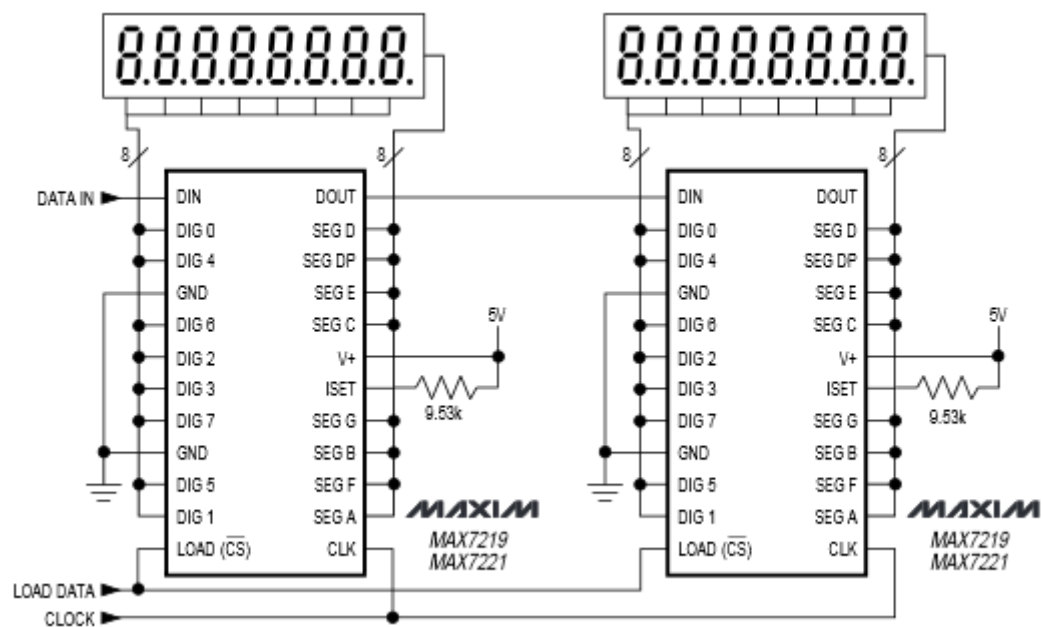
Assim, para podermos usar o CI, precisamos primeiro configurar o modo de trabalho que queremos, a quantidade de displays que estão conectados nele, e o brilho inicial, que depois pode ser mudado como quisermos.

Chega de conversa e vamos logo aos diagramas. Vamos apresentar os diagramas de conexão, as tabelas que nos interessam, e depois vamos explicar como utilizar este útil CI.

Exemplo de uso típico :



Como juntar dois CI's para termos até 16 dígitos :

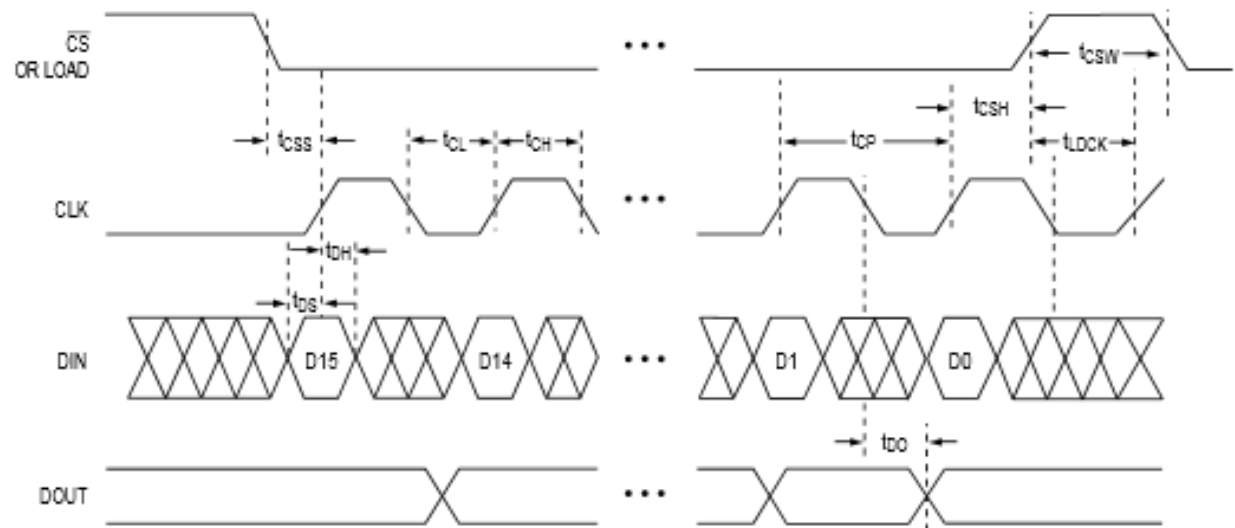


FORMATO DE COMUNICAÇÃO:

Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

## DIAGRAMA DE TEMPOS:



**Table 2. Register Address Map**

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	X0
Digit 0	X	0	0	0	1	X1
Digit 1	X	0	0	1	0	X2
Digit 2	X	0	0	1	1	X3
Digit 3	X	0	1	0	0	X4
Digit 4	X	0	1	0	1	X5
Digit 5	X	0	1	1	0	X6
Digit 6	X	0	1	1	1	X7
Digit 7	X	1	0	0	0	X8
Decode Mode	X	1	0	0	1	X9
Intensity	X	1	0	1	0	XA
Scan Limit	X	1	0	1	1	XB
Shutdown	X	1	1	0	0	XC
Display Test	X	1	1	1	1	XF



**Table 3. Shutdown Register Format (Address (Hex) = XC)**

MODE	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown Mode	XC	X	X	X	X	X	X	X	0
Normal Operation	XC	X	X	X	X	X	X	X	1

**Table 4. Decode-Mode Register Examples (Address (Hex) = X9)**

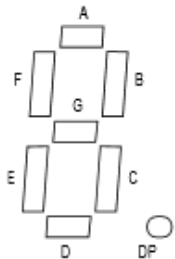
DECODE MODE	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
No decode for digits 7-0	0	0	0	0	0	0	0	0	00
Code B decode for digit 0 No decode for digits 7-1	0	0	0	0	0	0	0	1	01
Code B decode for digits 3-0 No decode for digits 7-4	0	0	0	0	1	1	1	1	0F
Code B decode for digits 7-0	1	1	1	1	1	1	1	1	FF

**Table 5. Code B Font**

7-SEGMENT CHARACTER	REGISTER DATA						ON SEGMENTS = 1							
	D7*	D6-D4	D3	D2	D1	D0	DP*	A	B	C	D	E	F	G
0		X	0	0	0	0		1	1	1	1	1	1	0
1		X	0	0	0	1		0	1	1	0	0	0	0
2		X	0	0	1	0		1	1	0	1	1	0	1
3		X	0	0	1	1		1	1	1	1	0	0	1
4		X	0	1	0	0		0	1	1	0	0	1	1
5		X	0	1	0	1		1	0	1	1	0	1	1
6		X	0	1	1	0		1	0	1	1	1	1	1
7		X	0	1	1	1		1	1	1	0	0	0	0
8		X	1	0	0	0		1	1	1	1	1	1	1
9		X	1	0	0	1		1	1	1	1	0	1	1
—		X	1	0	1	0		0	0	0	0	0	0	1
E		X	1	0	1	1		1	0	0	1	1	1	1
H		X	1	1	0	0		0	1	1	0	1	1	1
L		X	1	1	0	1		0	0	0	1	1	1	0
P		X	1	1	1	0		1	1	0	0	1	1	1
blank		X	1	1	1	1		0	0	0	0	0	0	0

\*The decimal point is set by bit D7 = 1

**Table 6. No-Decode Mode Data Bits and Corresponding Segment Lines**

 <p style="text-align: center;">STANDARD 7-SEGMENT LED</p>								
REGISTER DATA								
	D7	D6	D5	D4	D3	D2	D1	D0
Corresponding Segment Line	DP	A	B	C	D	E	F	G

**Table 7. Intensity Register Format (Address (Hex) = XA)**

DUTY CYCLE		D7	D6	D5	D4	D3	D2	D1	D0	HEX CODE
MAX7219	MAX7221									
1/32 (min on)	1/16 (min on)	X	X	X	X	0	0	0	0	X0
3/32	2/16	X	X	X	X	0	0	0	1	X1
5/32	3/16	X	X	X	X	0	0	1	0	X2
7/32	4/16	X	X	X	X	0	0	1	1	X3
9/32	5/16	X	X	X	X	0	1	0	0	X4
11/32	6/16	X	X	X	X	0	1	0	1	X5
13/32	7/16	X	X	X	X	0	1	1	0	X6
15/32	8/16	X	X	X	X	0	1	1	1	X7
17/32	9/16	X	X	X	X	1	0	0	0	X8
19/32	10/16	X	X	X	X	1	0	0	1	X9
21/32	11/16	X	X	X	X	1	0	1	0	XA
23/32	12/16	X	X	X	X	1	0	1	1	XB
25/32	13/16	X	X	X	X	1	1	0	0	XC
27/32	14/16	X	X	X	X	1	1	0	1	XD
29/32	15/16	X	X	X	X	1	1	1	0	XE
31/32	15/16 (max on)	X	X	X	X	1	1	1	1	XF

**Table 8. Scan-Limit Register Format (Address (Hex) = XB)**

SCAN LIMIT	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
Display digit 0 only*	X	X	X	X	X	0	0	0	X0
Display digits 0 & 1*	X	X	X	X	X	0	0	1	X1
Display digits 0 1 2*	X	X	X	X	X	0	1	0	X2
Display digits 0 1 2 3	X	X	X	X	X	0	1	1	X3
Display digits 0 1 2 3 4	X	X	X	X	X	1	0	0	X4
Display digits 0 1 2 3 4 5	X	X	X	X	X	1	0	1	X5
Display digits 0 1 2 3 4 5 6	X	X	X	X	X	1	1	0	X6
Display digits 0 1 2 3 4 5 6 7	X	X	X	X	X	1	1	1	X7

\*See *Scan-Limit Register* section for application.

**Table 9. Maximum Segment Current for 1-, 2-, or 3-Digit Displays**

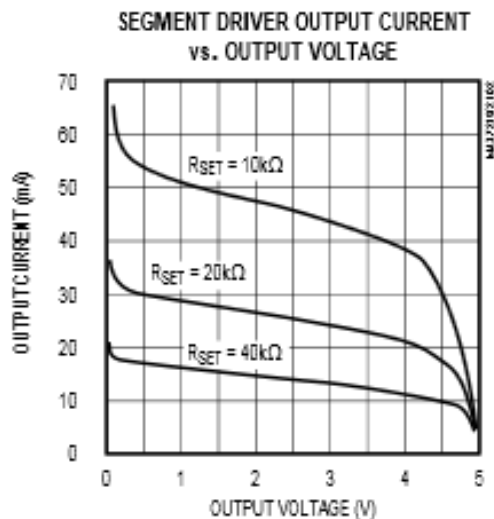
NUMBER OF DIGITS DISPLAYED	MAXIMUM SEGMENT CURRENT (mA)
1	10
2	20
3	30

**Table 10. Display-Test Register Format (Address (Hex) = XF)**

MODE	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Normal Operation	X	X	X	X	X	X	X	0
Display Test Mode	X	X	X	X	X	X	X	1

**Note:** The MAX7219/MAX7221 remain in display-test mode (all LEDs on) until the display-test register is reconfigured for normal operation.

## DIAGRAMA DE CORRENTE VERSUS RESISTOR DE SET



Pronto, tudo o que precisamos saber está colocado nas páginas anteriores.

Quando enviarmos os dados seriais, enviaremos primeiro o bit D15, depois o D14, e assim até o ultimo bit, que é o D0. Ou seja, sempre será no formato do MSB até o LSB.

Usamos três sinais para transmitirmos os dados. Temos dois sinais usados em comunicação serial, que são o CLOCK e o DATA. E o último é o CS, ou Chip Select, que tem de ser colocado em nível 0 quando desejamos que o chip aceite conversar conosco.

Assim, temos de fazer a seguinte sequência: Primeiro, colocamos CS em nível 1, a seguir CLOCK em nível 0, e colocamos DATA no nível que queremos. Agora podemos começar a conversa:

- CS=0
- CLOCK = 1 (pronto, o que está em DATA já foi para o MAX7219)
- CLOCK=0
- DATA=X ( X pode ser 0 ou 1 )
- CLOCK=1 (o que estava em DATA foi enviado ao MAX7219)
- CLOCK=0

Faremos esse processo 16 vezes no total, para enviarmos os 16 bits ao CI, e quando terminarmos, logo após colocar CLOCK=0, faremos CS=1 e pronto, acabamos!

Veja esse processo na figura onde está escrito DIAGRAMA DE TEMPOS.

Uma coisa que temos de escolher antes de montar é o valor do resistor RSET que iremos utilizar. Quanto mais baixo, maior a corrente disponível em cada segmento.

Para isto, vamos direto para a TABELA 11. Suponha que vamos usar um display Vermelho, e que ele permite 30 mA em cada segmento. Consultando o datasheet, você verifica que a tensão no Led para corrente de 30 mA é de 2 Volts. Assim, procure a coluna para 2 Volts, onde ela cruza com a corrente de 30 Ma, nele obtemos o valor de 17.1 kOhms . Usaremos o valor logo acima, que é de 18K.

Agora, já temos o hardware definido. Vamos usar 4 displays de 7 segmentos na cor vermelha, tipo catodo comum, com resistor Rset de 18K.

Nosso programa vai enviar os números no formato binário de 4 bits, representando de 0 a 9, e inicialmente queremos brilho máximo.

Para usarmos o MAX7219, temos primeiro de configurar 4 registros. Veja na TABELA 2, e acompanhe os nomes deles.

Vou utilizar um código de cores: Vermelho indica os bits não-significativos (podem ter qualquer valor), verde indica os bits que selecionam o registro desejado, e em azul os bits de dados que são importantes em cada registro.

Registro DECODE MODE : Veja na tabela 4, vamos configurar para que possamos usar o B-CODE para todos os dígitos. Assim, se quisermos acender o numero 9 no display, o valor binário a enviar será 1001b . Portanto, temos de enviar no formato binário os 16 bits abaixo, indo do MSB ao LSB: **111110011111111b**

Registro SCAN LIMIT : Veja na TABELA 8, como vamos utilizar 4 displays, temos de enviar a seguinte informação binária: **111110111111011b**

Registro INTENSITY : Veja na TABELA 7, como queremos inicialmente o brilho máximo , vamos enviar a seguinte informação binária: **111110101111111b**

Registro SHUTDOWN : Veja na TABELA 3, como queremos que o display entre em operação, vamos enviar a seguinte informação binária: **111111001111111b**

Pronto, com o envio desses 4 conjuntos de dados, o display já irá acender, e estará prontinho para operar normalmente.

Agora, precisamos enviar os dados. Para isto, como temos 4 displays ligados no CI, teremos de enviar 4 conjuntos de dados, cada conjunto irá fazer acender um display em uma determinada posição. Vamos chamar os nossos 4 displays de disp3 .... disp2 ... disp1 ... disp0, ou seja , Disp3 é o mais significativo (à esquerda) e Disp0 é o menos significativo (à direita).

Temos de acessar os registros com os nomes DIGIT 3, DIGIT 2, DIGIT 1 e DIGIT 0.

Vamos supor que temos de mostrar o numero 1234 . Veja agora a TABELA 2 para achar o registro, e a seguir a tabela 5.

Sequência binária para DISP 3 : 1111010001110001b

Sequência binária para DISP 2 : 1111001101110010b

Sequência binária para DISP 1 : 1111010101110011b

Sequência binária para DISP 0 : 1111010001110100b

O ponto decimal fica numa posição esquisita, mas vamos mostrar um exemplo: queremos acender o numero 5.678 . Veja a nova sequência:

Sequência binária para DISP 3 : 1111010011110101b (veja o ponto aceso, bit d7=1)

Sequência binária para DISP 2 : 1111001101110110b

Sequência binária para DISP 1 : 1111010101110111b

Sequência binária para DISP 0 : 1111010001111000b

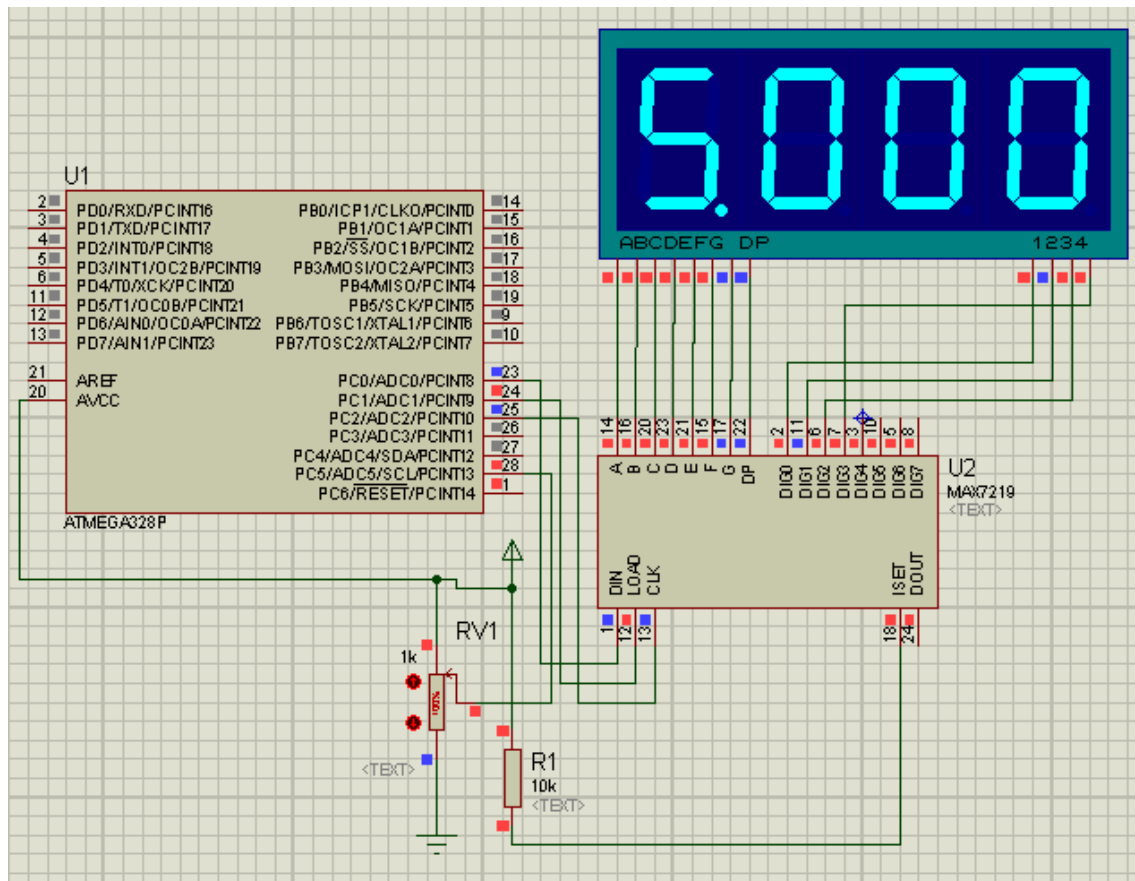
Até que agora não parece tão complicado, não é ?

Sugiro que você estude mais o MAX7219, pois ele faz um excelente controlador de matriz de Leds, e pode-se utilizar em muitas aplicações diferentes.

Eu mesmo fiz um Analisador de Espectro de Áudio, onde o mostrador eram duas matrizes de 8x8 Leds, colocadas uma ao lado da outra, utilizando dois CI's MAX7219.

Vamos a seguir mostrar o esquema de ligações utilizado no programa, repare que em vez de colocar 4 displays de 7 segmentos , usei um modelo diferente, facilmente encontrado no mercado, tendo os quatro dígitos em uma só peça.

E isto também é necessário para uma melhor simulação no Isis, pois este modelo de display suporta uma grande variação nas temporizações.



Agora, vamos ao programa em Bascom. Iremos modificar o programa do Voltímetro, que fizemos lá encima, para usar o MAX7219.

```

'-----
' PROGRAMA MULTIPLEX4 - Implementa um voltímetro que pode medir
' tensão entre 0 e 5 volts, e mostra com até 3 casas depois da vírgula
' Usaremos agora o MAX7219
' os displays estão numerados assim D4-D3-D2-D1
' Este programa utiliza aritmética de ponto flutuante.
'-----

```

```

$crystal = 8000000
'vamos usar o clock interno que é de 8 Mhz.
$regfile = "m328Pdef.dat"
' vamos compilar para o ATMEGA328P
$hwstack = 40
$swstack = 32
$framesize = 52

```

```

Dim Digitos(4) As Byte
Dim Digit_temp(6) As Byte
' nosso display terá 4 dígitos
Dim Dp_position As Byte
Dim Numero As Word
Dim Str_valor As String * 5
Dim I As Byte

```

```
Const Reg_decode_mode = &B1111100111111111
Const Reg_scan_limit = &B1111101111111011
Const Reg_intensity = &B1111101011111111
Const Reg_shutdown = &B1111110011111111
```

Dim Resultado As Single

' RESULTADO tem de apresentar numeros não inteiros

```
Config Portc.0 = Output           'data
Config Portc.1 = Output           'load
Config Portc.2 = Output           'clock
```

```
Max_data Alias Portc.0
Max_load Alias Portc.1
Max_clock Alias Portc.2
```

```
Digitos(1) = 0
Digitos(2) = 0
Digitos(3) = 0
Digitos(4) = 0
Max_load = 1
```

```
Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc
' aqui configuramos o conversor a/d para fazer uma leitura
' apenas quando pedirmos, vai usar como referencia 5V
```

```
Enable Interrupts
' aqui habilitamos as interrupções definidas no programa
```

```
Dp_position = 4
' ponto decimal sempre no primeiro dígito
```

'vamos agora inicializar o MAX7219

```
Numero = Reg_shutdown
Gosub Sai_max
Numero = Reg_decode_mode
Gosub Sai_max
Numero = Reg_intensity
Gosub Sai_max
Numero = Reg_scan_limit
Gosub Sai_max
```

'já configurado !

Do

```
Waitms 100
Numero = 0
For I = 1 To 32
```

```

    Numero = Numero + Getadc(5)
Next I
Shift Numero , Right , 5
'fizemos a soma de 32 leituras consecutivas, e calculamos
'a média para ter uma melhor estabilidade

Numero = Numero * 5
Resultado = Numero / 1023
' aqui fizemos o chamado fator de escala, pois o fundo de escala
' é 5 volts e nesse caso obteremos leitura de 1023, então
' basta multiplicar por 5 e dividir por 1023, e já teremos o
' valor em volts.

Str_valor = Fusing(resultado , "#.###")
' convertemos o resultado em uma string já formatada como queremos
' e com arredondamento na última casa decimal ! Mais uma
' poderosa instrução do Bascom

Str_valor = Trim(str_valor)
' eliminamos qualquer espaço em branco

Str2digits Str_valor , Digit_temp(1)
' fazemos o mesmo truque em tirar todos os dígitos da string e
' armazenar em uma matriz como número, assim fica muito simples
' pegar cada dígito e apresentar.

If Digit_temp(1) = 3 Then
    ' o valor a mostrar é zero
    Digitos(1) = 0
    Digitos(2) = 0
    Digitos(3) = 0
    Digitos(4) = 0
Else
    ' temos de ordenar os dígitos, agora com uma novidade, pois
    ' digit_temp(5) tem o ponto decimal, então pularemos
    Digitos(4) = Digit_temp(6)
    Digitos(3) = Digit_temp(4)
    Digitos(2) = Digit_temp(3)
    Digitos(1) = Digit_temp(2)
End If
For I = 4 To 1 Step -1
    Numero = Digitos(i)
    If Dp_position = I Then
        Numero = Numero Or &B1111000010000000
    End If
    Select Case I
        Case 1
            Numero = Numero Or &B1111010000000000
        Case 2
            Numero = Numero Or &B1111001100000000
        Case 3
            Numero = Numero Or &B1111001000000000
        Case 4
            Numero = Numero Or &B1111000100000000
    End Select
    Gosub Sai_max
Next I

```



Loop

```
Sai_max:
    Max_data = 0
    Max_clock = 0
    Max_load = 0
    Shiftout Max_data , Max_clock , Numero , 1
    Max_load = 1
Return
```

End

Este programa compilado tem 2.268 bytes.

### **Finalmente, para encerrar, um pouco de matemática em programação.**

Muitas vezes, não temos a enorme facilidade de contarmos com matemática de ponto flutuante em nossos projetos. Simplesmente porque isso faz o programa ficar muito maior, pois nossos microprocessadores são feitos para trabalhar com dados internos de 8 bits apenas. Ponto flutuante exige um monte de bytes para armazenamento, e um monte de operações de cálculos. E também demora muito mais tempo de processamento para serem executadas.

Podemos fazer o voltímetro usando apenas matemática com 16 bits, que é a que temos nos AVR's e nos Pics comuns? (Embora sejam de 8 bits, não dá trabalho fazer em partes de 8 bits, e podemos trabalhar facilmente com números de 16 bits).

Sim, podemos!!!! Sempre temos de usar um pouco de engenhosidade! Afinal, estudamos para isso, para pensar!

Vamos pensar em nosso display. Vamos apresentar 4 dígitos nele. Veja a tensão de 3.574 Volts. Se não tivesse o ponto decimal, seria mostrado o numero inteiro 3574 !

O maior valor que nosso voltímetro pode mostrar é 5.000 Volts, ou o numero inteiro 5000.

Afinal, qual o maior numero que podemos trabalhar com 16 bits? Resposta: 65535 !

Se fizermos o fundo de escala de nosso multímetro ser 5000, então resolvemos o problema! Basta acendermos a vírgula!

Podemos fazer as contas todas usando números de 16 bits ( chama-se WORD a esse tipo de variável ) , sem números decimais, e no final apresentar o numero do jeito que está !

Sabemos que o conversor A/D dos AVR's e dos Pics sempre apresenta os resultados da conversão em 10 bits. Isso significa um número entre 0 e 1023.

Se você se lembrar, nosso ultimo programa faz 32 medidas de tensão em seguida, somando todas elas, e em seguida divide por 32 para achar uma média, e temos o resultado novamente entre 0 e 1023.

A minha idéia é fazer mais medições e ir somando elas, para chegar bem pertinho da soma dar 50000. Então, vou fazer a soma de 49 medidas, que vai passar um pouquinho, resultando 50127. Temos então um pequeno erro, de cerca de  $((50127/50000) - 1) \times 100 = 0,254 \%$

Temos de subtrair isso da contagem, para obtermos o valor que queremos, que é 50000. Para isso, vamos lembrar na matemática básica do PERCENTUAL.

$$0,254\% = 0,254/100 = 0,2\% + 0,05\% + 0,004\% !$$

Agora, matemática básica do ginásio: se 2% de 100 é igual a 2, equivale a dividir 100 por 50, correto ? E se quisermos 0,2% de 100, equivale a dividir por 500 . Assim, se quisermos saber quanto é 0,2% de um número, basta dividir ele por 500!

A mesma coisa se aplica aos outros percentuais:

0,05% equivale a dividir o numero por 2000

0,004% equivale a dividir o número por 25000

Então, temos de fazer a seguinte conta:

Se a nossa soma deu 50.127, calculamos os percentuais e subtraímos do total!

Olha só:

$$50127 - 50127/500 - 50127/2000 - 50127/25000 \text{ fica assim :}$$

Lembre-se que a divisão comum sempre resulta apenas a parte inteira do numero:

$$50127 - 100 - 25 - 2 = 50000 \quad !!!!!$$

Pronto, conseguimos “ajustar a nossa medida”!

Esse resultado tem 5 dígitos, vamos dividir por 10 para voltarmos aos 4 dígitos , o que resulta em 5000 !

E quanto erro teremos nesse procedimento maluco? Oras, o conversor A/D sempre conta até 1023, portanto ele tem uma resolução de  $5/1023 = 4,89 \text{ mV}$  .

Na prática, é muito difícil filtrar os ruídos abaixo de 20 mV, portanto estamos trabalhando com uma resolução teórica de 5 mV, mas na prática considere 20 mV, ou teremos de projetar um belo layout, com muitos filtros.

Segue abaixo o programa, implementando exatamente a técnica descrita acima. Tente entender, pois é muito usada quando trabalhamos em Assembler, ou quando temos pouca memória para o programa.

```

'-----
' PROGRAMA MULTIPLEX 5 - Implementa um voltímetro que pode medir
' tensão entre 0 e 5 volts, e mostra com até 3 casas depois da vírgula
' Usaremos agora o MAX7219
' os displays estão numerados assim D4-D3-D2-D1
' Este programa utiliza aritmética de 16 bits apenas !
'-----

```

```

$crystal = 8000000
'vamos usar o clock interno que é de 8 Mhz.
$regfile = "m328Pdef.dat"
' vamos compilar para o ATMEGA328P
$hwstack = 40
$swstack = 32
$framesize = 52

```

```

Dim Digitos(4) As Byte
' nosso display terá 4 dígitos
Dim Dp_position As Byte
Dim Numero As Word
Dim I As Byte

```

```

Const Reg_decode_mode = &B1111100111111111
Const Reg_scan_limit = &B1111101111111011
Const Reg_intensity = &B1111101011111111
Const Reg_shutdown = &B1111110011111111

```

```

Dim Resultado As Word
' RESULTADO tem de apresentar numeros não inteiros
Dim Resultado1 As Word
' MAIS UMA VARIÁVEL PARA OS CÁLCULOS

```

```

Config Portc.0 = Output           'data
Config Portc.1 = Output           'load
Config Portc.2 = Output           'clock

```

```

Max_data Alias Portc.0
Max_load Alias Portc.1
Max_clock Alias Portc.2

```

```

Digitos(1) = 0
Digitos(2) = 0
Digitos(3) = 0
Digitos(4) = 0
Max_load = 1

```

```

Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc
' aqui configuramos o conversor a/d para fazer uma leitura
' apenas quando pedirmos, vai usar como referencia 5V

```

```

Enable Interrupts
' aqui habilitamos as interrupções definidas no programa

```

```

Dp_position = 4
' ponto decimal sempre no primeiro dígito

'vamos agora inicializar o MAX7219

Numero = Reg_shutdown
Gosub Sai_max
Numero = Reg_decode_mode
Gosub Sai_max
Numero = Reg_intensity
Gosub Sai_max
Numero = Reg_scan_limit
Gosub Sai_max

'já configurado !

Do
    Waitms 100
    Numero = 0
    For I = 1 To 49
        Numero = Numero + Getadc(5)
    Next I

' vamos calcular a media, somando 49 ezes a leitura. Porque 49 ?
' se multiplicarmos 49 x 1023, teremos 50.127, que ultrapassa em pouco 50.000
' SE DIVIDIRMOS 50127/50.000 = 1,00254, que é 2,54% maior do que queremos,
' agora, temos um valor que ultrapassa 50.000 em 0,254% . Vamos portanto
subtrair
' esses 0,254% !

    Resultado1 = Numero
    Resultado1 = Resultado1 / 500                                ' 0,2%
    Numero = Numero - Resultado1
    Resultado1 = Numero
    Resultado1 = Numero / 2000                                    '0,05 %
    Numero = Numero - Resultado1
    Resultado1 = Numero
    Resultado1 = Resultado1 / 25000                              ' 0,004 %
    Numero = Numero - Resultado1

' FINALMENTE SUBTRAIMOS 2,54%
' VAMOS AGORA DIVIDIR POR 10 PARA CHEGARMOS NA ESCALA QUE QUEREMOS.
Numero = Numero / 10

    Resultado1 = Numero / 1000
    Digitos(4) = Low(resultado1)
    Resultado = Digitos(4) * 1000
    Numero = Numero - Resultado

    Resultado1 = Numero / 100
    Digitos(3) = Low(resultado1)
    Resultado = Digitos(3) * 100
    Numero = Numero - Resultado

    Resultado1 = Numero / 10
    Digitos(2) = Low(resultado1)

```

```

Resultado = Digitos(2) * 10
Numero = Numero - Resultado

Digitos(1) = Low(numero)

' prontinho... não precisamos utilizar matemática de ponto flutuante, que
' sempre faz o programa ficar bem maior. Veremos no final a redução do
programa.

For I = 4 To 1 Step -1
    Numero = Digitos(i)
    If Dp_position = I Then
        Numero = Numero Or &B1111000010000000
    End If
    Select Case I
        Case 1
            Numero = Numero Or &B1111010000000000
        Case 2
            Numero = Numero Or &B1111001100000000
        Case 3
            Numero = Numero Or &B1111001000000000
        Case 4
            Numero = Numero Or &B1111000100000000
    End Select
    Gosub Sai_max
Next I

Loop

Sai_max:
    Max_data = 0
    Max_clock = 0
    Max_load = 0
    Shiftout Max_data , Max_clock , Numero , 1
    Max_load = 1
Return

End

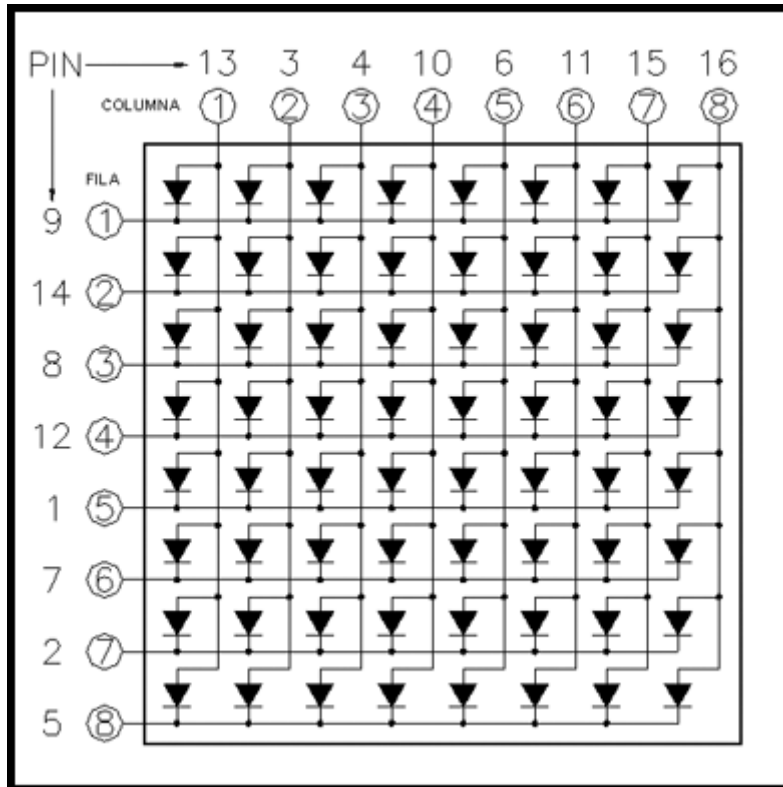
```

Este programa compilado tem 1.474 bytes. Se comparar com o anterior, que utiliza ponto flutuante, vemos que a redução no tamanho é enorme, cerca de 35% menor !!!

Repare que fizemos a separação dos 4 dígitos da maneira mais simples possível, sem utilizar nenhuma função de alto nível do Bascom. Fiz isto apenas para que você aprenda esta maneira simples de se separar os dígitos individuais de um número.

## PARTE 2 – MATRIZES DE LED's

As matrizes de Leds são componentes que geralmente contém 64 Leds no total, organizados em 8 linhas de 8 colunas, conforme a figura abaixo, que mostra uma matriz do tipo CATODO COMUM :



Repare que os catodos dos Leds são interligados 8 a 8. No exemplo acima, para acender o primeiro Led da esquerda, parte superior, temos de colocar uma tensão positiva na primeira coluna pino 13, e colocar nível 0 na primeira linha, que é o pino 1.

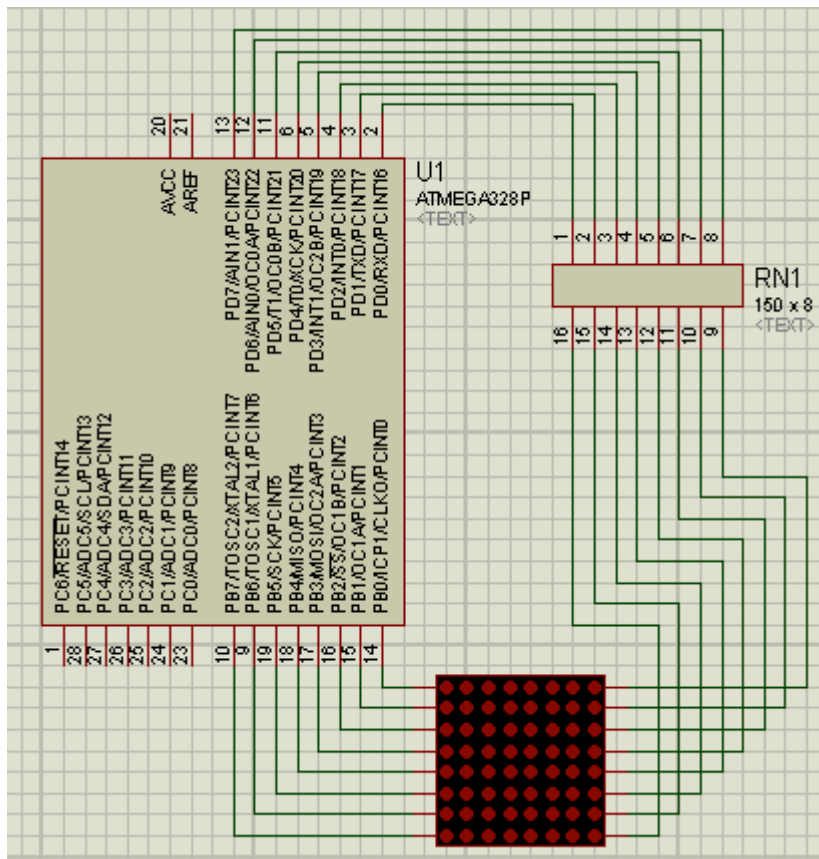
Repare a dificuldade que é acender vários Leds.... só é possível acender ao mesmo tempo os Leds pertencentes a uma mesma linha !

Se quisermos acender vários, temos de recorrer à Multiplexação, onde iremos acionar os Leds da primeira linha, em seguida os da segunda linha, e assim até a última linha. Se fizermos isto com uma boa velocidade, nosso olho vai fazer a famosa “integração”, e veremos como se todos estivessem acesos ao mesmo instante.

Como fazemos isto?

A primeira maneira é a mais simples, usamos uma porta inteira de um microcontrolador para fazer as linhas, e outra porta inteira para fazer as colunas. Perderemos 16 pinos do nosso microcontrolador para fazer isto.

Veja o esquema abaixo:



Aqui, usamos o Port B para comutar as colunas, e o Port D para comutar as linhas.

Repare que temos os mesmos problemas de corrente que tivemos acima, usando os displays de 7 segmentos diretamente. A corrente em cada coluna é limitada à corrente máxima da de um pino, portanto a corrente de cada Led das linhas tem de ser no máximo a da coluna / 8. Isto limita bastante o brilho, quando usamos o princípio do Multiplex.

Vamos ilustrar aqui, como seria um programa para acender um enorme X na matriz.

Imagine que iremos utilizar 8 bytes de memória, totalizando  $8 \times 8 = 64$  bits, portanto é uma representação perfeita da matriz.

Vamos apresentar os 8 bytes da seguinte maneira : o primeiro byte representa os Leds da primeira linha; o próximo byte representa a segunda linha, e assim por diante.

Dentro de cada byte, o MSB representa o Led mais à esquerda, e o LSB representa o Led mais à direita.

Agora, vamos supor que o bit em 1 representa o Led aceso, e 0 representa o Led apagado.

Veja como ficam os nossos 8 bytes para acender o X:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
Byte1	1	0	0	0	0	0	1	0
Byte2	0	1	0	0	0	1	0	0
Byte3	0	0	1	0	1	0	0	0
Byte4	0	0	0	1	0	0	0	0
Byte5	0	0	1	0	1	0	0	0
Byte6	0	1	0	0	0	1	0	0
Byte7	1	0	0	0	0	0	1	0
Byte8	0	0	0	0	0	0	0	0

Reparou que tem uma coisa esquisita.. não usamos todas as 8 linhas e as 8 colunas, mas sim 7 linhas e 7 colunas ?

Se você pretende escrever um texto na matriz, é recomendado sempre que o tamanho da matriz seja de números IMPARES, ou seja, 7x7 , 7x5 . Senão, se fosse usado um número par, nunca teria uma posição central, aquela bem no meio da matriz, que no nosso caso é exatamente o meio do X, que está na Col4 e Byte4.

Nesta representação que adotei, não importa a letra a ser mostrada, sempre teremos o Byte8 igual a 0 e a Col8 igual a 0.

Nosso programa fará o Multiplex como se fossem 8 dígitos , fazendo acender uma coluna a cada vez, como se fosse um display do tipo CATODO COMUM, igual ao nosso exemplo lá no começo ....

Assim, selecionamos a coluna 1 fazendo sair no PortB o valor 0111111b.  
A seguir, colocamos no PortD o valor 10000010b , e isto fará acender os dois Leds.

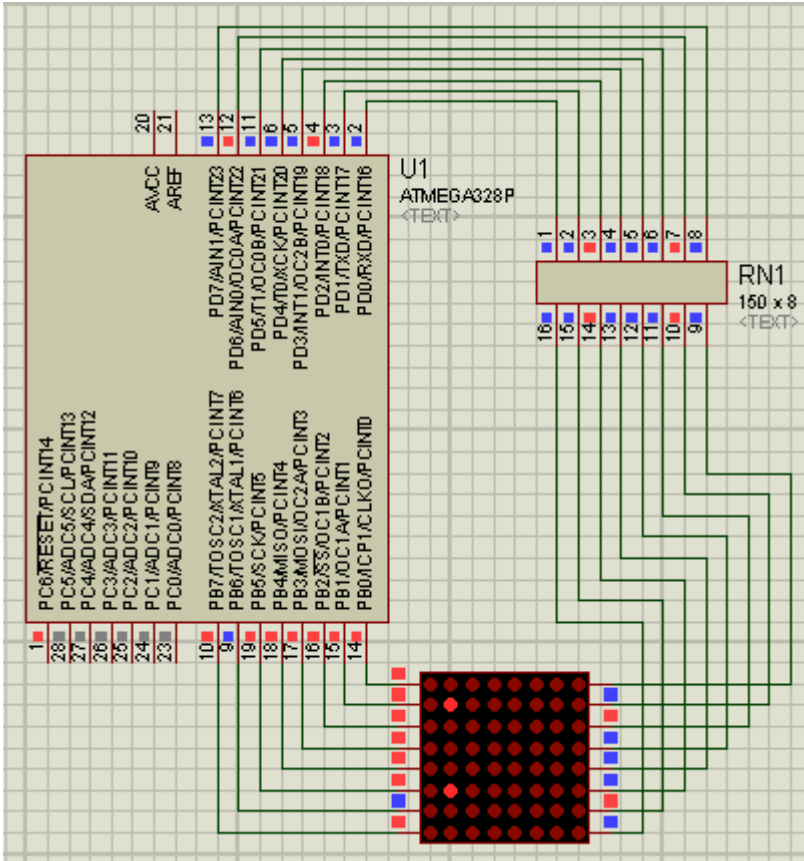
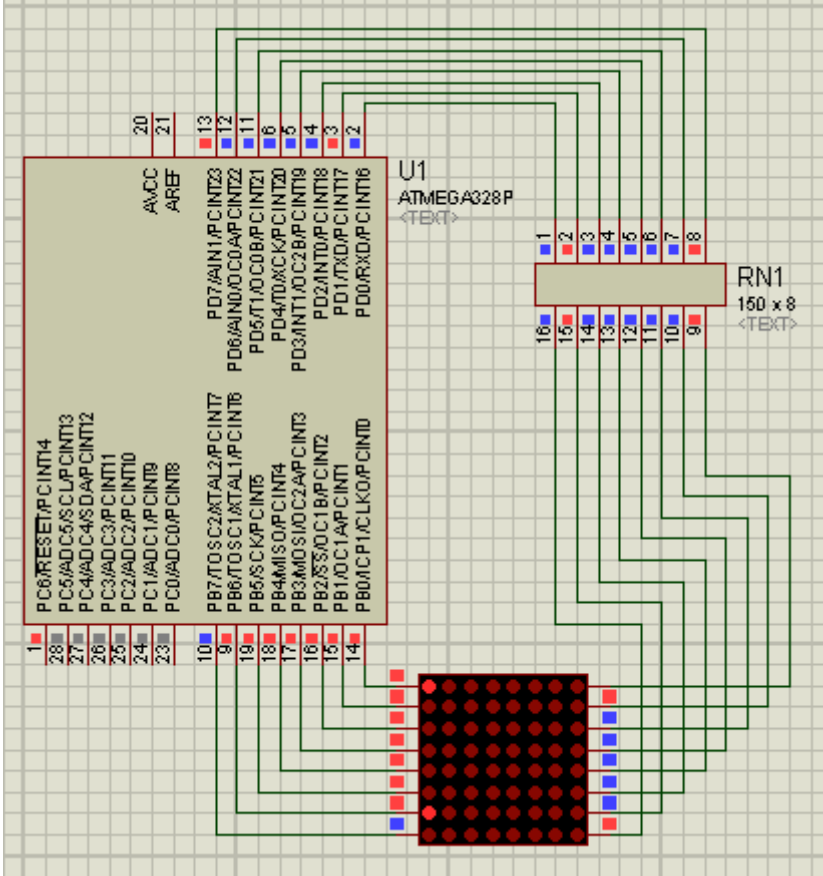
Na próxima vez, faremos PortB = 1011111b e PortD = 01000100b

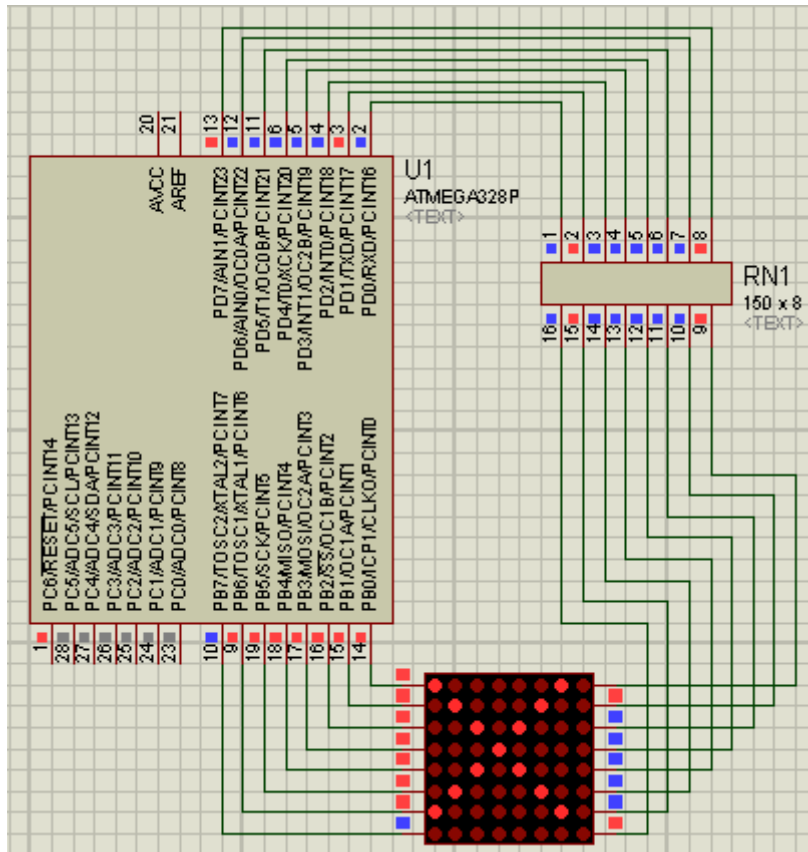
E assim por diante. Repararam que é exatamente o mesmo princípio que usamos lá em cima para fazer o Multiplex do display com 4 dígitos ? Só que aqui são 8 dígitos !

A situação é muito parecida com a que já vimos antes, porém tratamos agora as 8 colunas da mesma maneira com que tratamos os 4 dígitos. Tivemos de aumentar a frequência do Multiplex para 240 Hz, para manter a mesma taxa de acender cada coluna pelo menos 30 vezes por segundo.

Veja dois instantes seguidos da Multiplexação, e o efeito total captado pelo nosso olho:







Vamos ao programa em Bascom.

## PROGRAMA 5 – MATRIX DE 8X8 BÁSICA

```
'-----
' PROGRAMA MULTIPLEX MATRIX1 - Implementa o Multiplex Basico para
' fazer acender um X na nossa matrix.
' Usaremos uma tabela de 8 bytes para cada letra ou numero a ser
' mostrado, mas por simplicidade colocaremos apenas o valor
' correspondente à letra X
' Este programa utiliza o Timer0 de 8 bits para a base de tempo
' e implementa um refresh de 240 Hertz aproximado.
'-----
```

```
$crystal = 8000000
$regfile = "m328def.dat"
$hwstack = 40
$swstack = 16
$framesize = 32
```

```
Dim Disp_index As Byte
Dim Disp_aux As Byte
```

```
Config Portb = Output
' PortB vai acender as COLUNAS
```

```
Config Portd = Output
'PortD vai acender as LINHAS
```

```
Config Timer0 = Timer , Prescale = 256
Timer0 = 126
' vamos gerar uma interrupção a cada 4,16 milissegundo
' que corresponde a uma frequencia de Multiplex de 240,4 Hertz
' ou seja, cada coluna vai acender 30 vezes por segundo !
```

```
On Timer0 Timer0_sub
Enable Timer0
Enable Interrupts
```

```
Do
    Waitms 100
```

```
Loop
```

```
End
```

```
'----- ROTINAS DE INTERRUPÇÃO -----
```

```
Timer0_sub:
' Rotina chamada pelo Timer0 a cada
Timer0 = 126
'recarrega o timer novamente para

Portb = 255
' apaga todos os dígitos

If Disp_index > 8 Then Disp_index = 0
' vamos ver se já fizemos o ultimo dígito, pois então teremos de
' começar pelo primeiro novamente

Incr Disp_index
' aqui já apontamos para o próximo dígito

Disp_aux = Disp_index - 1
' pegamos os Leds que queremos mostrar ( linha ) nesta coluna
Portd = Lookup(displ_aux , Table_x)

' agora vamos selecionar a coluna
Disp_aux = Lookup(displ_index , Table_mux)
Waitms 25
Portb = Disp_aux
' Acendemos a coluna
Return
```

```
' TABELA DE Leds A ACENDER APENAS PARA A LETRA X
```

```
Table_x:
    Data &B10000010 , &B01000100 , &B00101000 , &B00010000
```

Data &B00101000 , &B01000100 , &B10000010 , &B00000000

#### ' TABELA INDICADORA Da Coluna A ACENDER

Table\_mux:

Data &B11111111 , &B01111111 , &B10111111 , &B11011111 , &B11101111

Data &B11110111 , &B11111011 , &B11111101 , &B11111110 , &B11111111

Programa compilado com tamanho de 542 Bytes.

Agora, vamos ao próximo passo: Da maneira que desenhamos o circuito, nunca vamos conseguir uma boa corrente para os Leds. Uma das soluções é utilizar o mesmo princípio dos transistores que utilizamos acima, para poder aumentar a corrente. Já seria um excelente passo. Mas existe outra maneira, que é a utilização de um CI Driver de corrente, que vai fazer exatamente a mesma função dos transistores, e a de um CI Source de corrente, que vai poder fornecer muito mais corrente do que os pinos do microcontrolador. A vantagem é que não precisamos mudar nada em nosso programa!

Imagine que você está usando uma matriz de Leds que suporte a corrente de 40 mA em cada Led. Se vamos acender todos os 8 Leds de uma linha, teremos um total de corrente de 320 mA , que será fornecida pelo CI Source, e consumida pelo CI Driver.

Desta maneira vamos ter um excelente brilho!

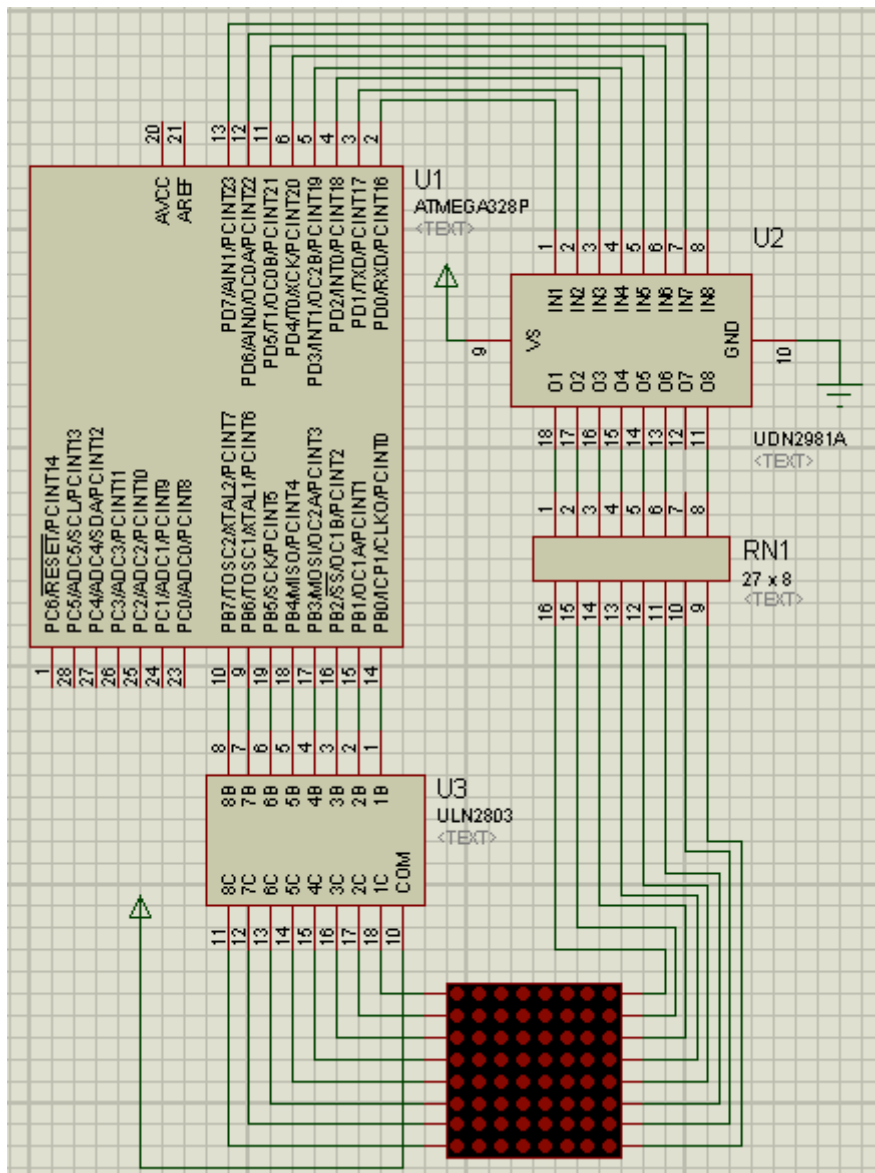
Mas, temos de mais uma vez calcular o valor do resistor limitador de corrente do Led, só que agora temos de calcular em função das tensões nas saídas dos dois Drivers. Olhando o datasheet, temos que para o ULN2803A a menor tensão VceSat é de 0,9 volts, e para o UDN2981A é de 1,6 Volts. Assim, se nosso Leds vermelho tiver tensão de condução para 30 mA de 1,8 Volts, temos uma tensão total de  $0,9 + 1,6 + 1,8 = 4,3$  volts, e nosso resistor limitador será de  $(5-4,3)/0,03 = 23,3$  ohms, e usaremos o valor comercial de 27 ohms.

Apenas um detalhe: em um caso como este, o certo seria alimentar o UDN2981A com mais tensão, para garantir que não haja uma grande variação de brilho por causa da variação das tensões de VceSat dos dois CI's, que podem variar em torno de 0,2 Volts para cada um. Imagine que no pior caso vamos ter as seguintes tensões: 1,1 e 1,8 , mais 1,8 do Led, teremos um total de 4,7 Volts, que com  $R=27$  Ohms vai dar uma corrente de  $0,3/27 = 9$  mA !!! Percebeu o problema? Menos de um terço da corrente !

Agora, imagine se alimentarmos o UDN2981A com 12 Volts, refazendo os cálculos todos, terá um resistor de  $12-4,3= 7,7/0,03 = 256$  Ohms, e usaremos o valor imediatamente acima que é de 270 Ohms.

Se tivermos a mesma variação de tensão VceSat, vamos ter uma corrente de  $(12-4,7)/270 = 27$  mA, que vai manter o brilho quase que inalterado.

Segue como ficaria o nosso circuito (mantendo 5V de alimentação) :



Agora, evoluindo novamente nosso circuito, imagine que vamos usar DUAS MATRIZES em vez de uma. Como fazer a ligação, se não temos mais uma porta com 8 bits livres ?

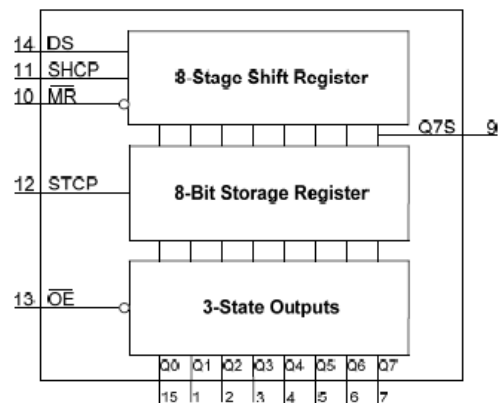
Aqui entra um velho amigo, o 74HCT595. Ele é um CI que integra um Shift-Register e um Latch de saída, permitindo que enviemos os dados para ele usando apenas 3 pinos de I/O, usando o formato serial ! E, o melhor de tudo, pode ligar vários em cascata, um atrás do outro!

Usaremos novamente um único CI UDN2981A, o qual vai fornecer corrente de sobra para todas as matrizes (afinal, apenas uma vai acender de cada vez), e iremos adicionar um ULN1803A em conjunto com um 74HCT595 para cada matriz adicional.

## USANDO 74HCT595 PARA INTERLIGAR VÁRIAS MATRIZES

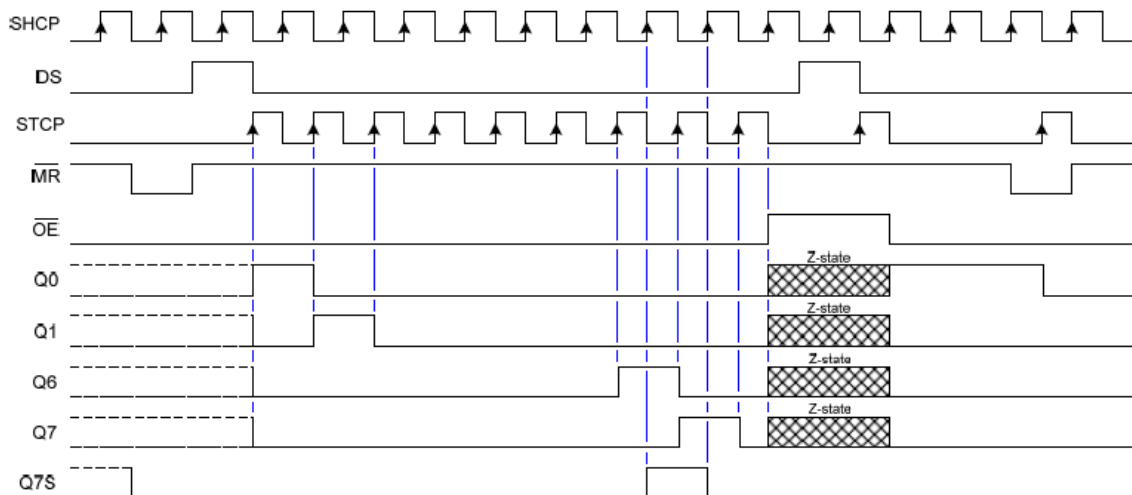
A seguir, veremos como utilizar este maravilhoso CI de baixo custo.

Pin Number	Pin Name	Description
1	Q1	Parallel Data Output 1
2	Q2	Parallel Data Output 2
3	Q3	Parallel Data Output 3
4	Q4	Parallel Data Output 4
5	Q5	Parallel Data Output 5
6	Q6	Parallel Data Output 6
7	Q7	Parallel Data Output 7
8	GND	Ground
9	Q7S	Serial Data Output
10	MR	Master Reset Input
11	SHCP	Shift Register Clock Input
12	STCP	Storage Register Clock Input
13	OE	Output Enable Input
14	DS	Serial Data Input
15	Q0	Parallel Data Output 0
16	Vcc	Supply Voltage



Control				Input	Output		Function
SHCP	STCP	OE	MR	DS	Q7S	Qn	
X	X	L	L	-	L	NC	Low-level asserted on MR clears shift register Storage register is unchanged
X	↑	L	L	-	L	L	Empty shift register transferred to storage register
X	X	H	L	-	L	Z	Shift register remains clear; All Q outputs in Z state
↑	X	L	H	-	Q6S	NC	HIGH is shifted into first stage of Shift Register Contents of each register shifted to next register The content of Q6S has been shifted to Q7S and now appears on device pin Q7S
X	↑	L	H	-	NC	QnS	Contents of shift register copied to storage register With output now in active state, the storage register contents appear on Q outputs
↑	↑	L	H	-	Q6S	QnS	Contents of shift register copied to output register then shift register shifted.

H=HIGH voltage state  
 L=LOW voltage state  
 ↑=LOW to HIGH transition  
 X= don't care – high or low (not floating)  
 NC= No change  
 Z= high-impedance state



Reparem acima as tabelas de funcionamento. Inicialmente, não vamos utilizar o sinal /MR (Master Reset), ligando ele ao positivo.

Temos agora apenas quatro sinais, que são o DS (Dado Serial), o STCP (Storage Clock Pulse), o SHCP (Shift Clock Pulse), e o /OE (Output Enable)

Qual a mecânica de funcionamento?

**DS** – Dado serial a ser colocado dentro do shift register

**/OE** – Levamos ao nível 1 , que coloca a saída em tri-state, fazendo apagar o display.

**SHCP** – Normalmente 0, quando ocorre uma transição para 1 ( subida ), faz com que o nível presente em DS seja transferido à saída 0 (interna) do shift register; o nível que estava presente na saída 0 (interna) é transferido para a saída 1 (interna) e assim sucessivamente até sair na saída 7 (interna) . Na verdade fizemos um deslocamento das 8 saídas do shift register, sendo que entramos com uma nova informação, que é o nível que estava em DS. Agora, fazemos a transição de 1 para 0 , o que prepara para um novo dado.

Basta colocar um novo nível desejado em DS, e repetir o pulso de SHCP de 0 para 1 e voltar novamente para 0, e mais um dado será armazenado.

Se repetirmos esse procedimento 8 vezes no total, teremos colocado todos os 8 dados desejados dentro do shift register.

Mas até o momento estas mudanças todas não aparecem nas saídas Q0-Q7 do CI, porque isso só acontece quando mandarmos “armazenar os dados”!

**STCP** – Normalmente 0, quando vai para 1 permite que os níveis armazenados no shift register sejam transferidos para as saídas Q0-Q7. A seguir, voltamos o nível para 0 e os dados que estão nas saídas vão se manter, independente do que fizemos nos sinais DS ou SHCP . A saída foi “armazenada”!

**/OE** – Colocamos novamente em 0, o que habilita as saídas, fazendo acender a coluna.

Assim, nossa rotina fará o seguinte: pega os 8 bits que sevem ser transferidos para a Matriz de Leds, e coloca serialmente , um a um, até todos os 8 estiverem transferidos, aí selecionamos a linha ou coluna a ser acionada, e fazemos o armazenamento, fazendo com que a linha ou a coluna tenha os seus 8 Leds acesos conforme o nível do sinal.

Simples, basta apenas fazermos isto tudo rapidamente.

A nossa nova rotina de Multiplex vai fazer o seguinte:

- Pega todos os 8 Leds a serem apresentados na primeira coluna, e coloca na saída de 8 bits do Microcontrolador. Esta saída está ligada diretamente a um UDN2981A que irá entregar uma corrente bem maior a cada Led.

- Habilita a passagem de corrente dessa primeira coluna (saída dos 74HCT595) que estão ligadas a um ULN2803A, que suporta a corrente de todos os Leds acesos.

- Quando houver outra interrupção, vai pegar os 8 Leds a serem apresentados na segunda coluna, e irá habilitar a segunda coluna para acender os 8 Leds. E assim

s sucessivamente, até terminar todas as colunas existentes (por exemplo, se forem 3 matrizes, teremos 24 colunas).

Qual vai ser a frequência do Multiplex? Como temos de mostrar cada coluna pelo menos 30 vezes por segundo, nossa frequência será 30 vezes o número de colunas!

Se quisermos acender 3 matrizes, num total de 24 colunas, teremos de selecionar a frequência de 720 Hertz.

Para encadearmos vários 74HCT595, basta ligar a saída /Q7 do primeiro à entrada DS do segundo. Os outros sinais são ligados em paralelo. Nesse caso, em vez de enviarmos apenas 8 bits seriais, enviamos tantos bits conformem forem as colunas. No caso de 3 matrizes, sempre enviaremos 24 bits antes de armazenar para ser mostrado.

Um truque legal é que em vez de enviarmos sempre os 24 bits seriais, se utilizarmos os 74HCT595 para habilitar as colunas, basta sempre apenas fazer **um** pulso de clock do shift, e isto já fará com que uma saída previamente habilitada se propague, e já poderemos armazenar novamente. Ganhamos bastante tempo com esse procedimento.

Você vai entender isto melhor quando verificar o circuito.

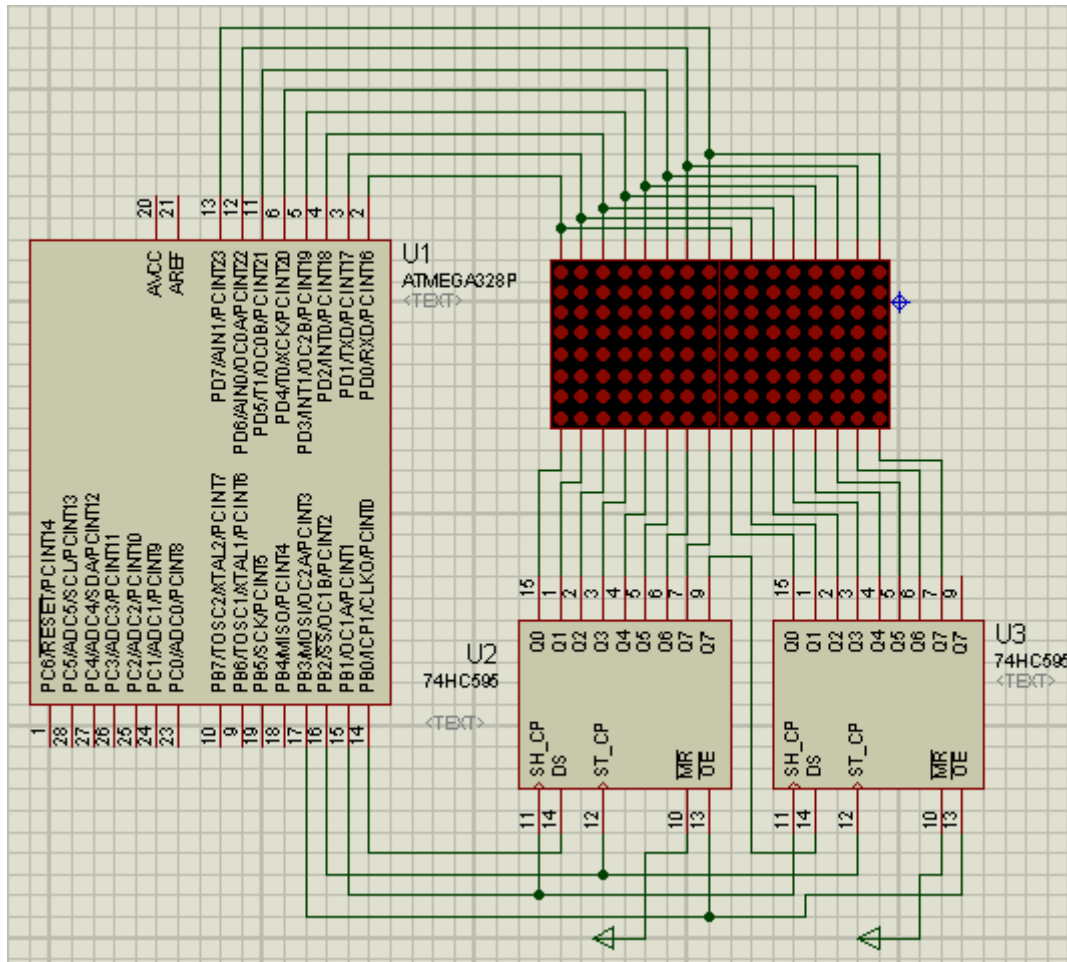
No caso de um circuito com muitas matrizes, temos de lembrar de duas características do CI ULN2803A, que são o tempo mínimo para entrar em condução (20 nano segundos), e o tempo máximo para sair de condução (130 nano segundos). Lembra-se da discussão sobre corte e saturação dos transistores, que está lá em cima, quase no começo? É o mesmo efeito aqui, porém como este CI é próprio para isto, os tempos envolvidos são bem menores. Mas dependendo da frequência do Multiplex, estes tempos podem ter de ser levados em conta!

Bom, chega de teoria, vamos a um exemplo de um circuito que utiliza duas matrizes de Leds.

Um uso corriqueiro para este tipo de circuito são os painéis eletrônicos de Leds, onde os textos ficam passando pelos displays, indo da direita para a esquerda. É um efeito muito bonito, e que você pode implementar tanto no programa principal como na rotina de Multiplex.

Mais para frente, veremos uma solução bem melhor para isto, que é o uso do MAX7219 para o controle de uma matriz completa. Assim, basta encadearmos vários MAX7219 para fazermos displays enormes, com a facilidade de controle de brilho e uma programação muito mais simples, pois esses CI's já fazem o Multiplex interno para nós!





Para simplificar a simulação, este circuito não apresenta nem o UDN2981A que teria de estar ligado na saída do Port D do microcontrolador, nem os resistores que teriam de estar entre as saídas do UDN2981A e as matrizes, e nem os dois ULN2803A que deveriam estar presentes em cada um dos 74HCT595. Devido ao fato de a presença do ULN2803A faz a inversão da saída, teremos de modificar o programa quando ele for utilizado. Esta modificação será também apresentada.

Da maneira mostrada no esquema, nossas duas matrizes equivalem a uma única matriz, como o formato de 8 linhas por 16 colunas. Para representar esta coluna, usaremos 16 bytes, sendo que o primeiro byte contem os 8 Leds mais à esquerda, respectivamente o LSB em cima e o MSB em baixo; daí sucessivamente até o último byte, que representa os 8 Leds mais à direita.

Vamos dimensionar uma matriz de 16 bytes para isso, onde os bytes serão acessados desde coluna(1) até coluna(16).

A nossa rotina de Multiplex será bem simples, ela irá a cada vez inibir a saída fazendo  $/OE = 1$ , e então pega os Leds que tem de acender na próxima coluna, coloca na saída do Port D, faz  $DS=0$ , gera um clock no shift para avançar para a próxima coluna a mostrar, armazena fazendo  $STCP=1$ , e então faz novamente  $/OE=0$ , o que irá acender a coluna desejada.

A cada vez ela seleciona a próxima coluna, tendo o cuidado de voltar ao início quando estivermos na última posição.

Simple e efetivo, não é?

A seguir, o programa que ilustra essa técnica, e que irá apresentar o mesmo X do programa anterior na primeira matriz, e o inverso do X na segunda matriz.

```
'-----  
' PROGRAMA MULTIPLEX MATRIX2 - Implementa o Multiplex Basico para  
' fazer acender um X normal na primeira matriz e o inverso desse X na  
' segunda matriz. Usamos agora o 74HCT595 para controlar as matrizes  
' Usaremos uma Matriz de 16 bytes para armazenar os Leds das duas matrizes.  
'  
' Este programa utiliza o Timer0 de 8 bits para a base de tempo  
' e implementa um refresh de 480 Hertz aproximado.  
'-----
```

```
$crystal = 8000000  
$regfile = "m328def.dat"  
$hwstack = 40  
$swstack = 16  
$framesize = 32
```

```
Dim Disp_index As Byte  
Dim Disp_aux As Byte  
Dim Coluna(16) As Byte  
Dim Nada As Word  
Dim Flag As Bit
```

```
Config Portb = Output  
' PortB vai acender as COLUNAS
```

```
Config Portd = Output  
'PortD vai controlar os 74HCT595
```

```
Ds Alias Portb.0  
Shcp Alias Portb.1  
Stcp Alias Portb.2  
Oe Alias Portb.3
```

```
' Vamos agora carregar nossos 16 bytes com o desenho  
' do X e do inverso do X que estão na tabela TABLE_X  
For Disp_index = 0 To 15  
    Disp_aux = Lookup(dispen_index , Table_x)  
    Incr Disp_index  
    Coluna(dispen_index) = Disp_aux  
    Decr Disp_index  
Next Disp_index
```

```
Config Timer0 = Timer , Prescale = 256  
Timer0 = 191  
' vamos gerar uma interrupção a cada 2,08 milissegundo
```

```
' que corresponde a uma frequencia de Multiplex de 480 Hertz  
' ou seja, cada coluna vai acender 30 vezes por segundo !
```

```
On Timer0 Timer0_sub  
Enable Timer0  
Enable Interrupts
```

```
Nada = 65535  
' é apenas um padrão de 16 bits 1 , que vamos usar para não  
' selecionar nenhuma coluna para iniciar o processo !
```

```
Oe = 1  
'Faz as matrizes apagarem
```

```
Shiftout Ds , Shcp , Nada , 1  
' Envia os 16 bits 1 aos shift registers
```

```
Stcp = 1  
NOP  
Stcp = 0  
' armazenou e colocou todos os 16bits 1 na saída dos latches
```

```
Shcp = 0  
Ds = 0  
Disp_index = 0
```

```
Do  
    Waitms 100
```

```
Loop
```

```
End
```

```
'----- ROTINAS DE INTERRUPÇÃO -----
```

```
Timer0_sub:  
    ' Rotina chamada pelo Timer0  
    Timer0 = 191  
    'recarrega o timer novamente para  
  
    ' apaga todos os dígitos  
    Oe = 1
```

```
    ' vamos ver se já fizemos o ultimo dígito, pois então teremos de  
    ' começar pelo primeiro novamente, carregando 16 bits 1 !
```

```
If Disp_index > 16 Then  
    Flag = 1  
    Shiftout Ds , Shcp , Nada , 1  
    Stcp = 1  
    NOP  
    Stcp = 0  
    Disp_index = 0  
    Ds = 0
```

```
End If
```

```
    ' agora vamos carregar um nível 1 para a próxima coluna a mostrar  
    ' uso um truque aqui. Se eu acabei de inicializar tudo com os 16 bits 1,
```

```

' eu deixo DS=0 para que um nivel zero seja introduzido no shift
' register, assim vai fazer acender uma coluna. A seguir, faço DS=1
' e todos os outros shifts vão apenas propagar uma só coluna com
' nivel 0.
Shcp = 1
NOP
Shcp = 0
Stcp = 1
NOP
Stcp = 0
Ds = 1

Incr Disp_index
' aqui já apontamos para a coluna

' Se não zeramos os shifts, temos de ter um delay
If Flag = 0 Then
    Waitus 30
Else
    Flag = 0
End If

' pegamos os Leds que queremos mostrar nesta coluna
' dentro da nossa matriz coluna()
Portd = Coluna(dispc_index)

' agora vamos acender uma só coluna!
Oe = 0
' Acendemos a coluna
Return

' TABELA DE Leds A ACENDER

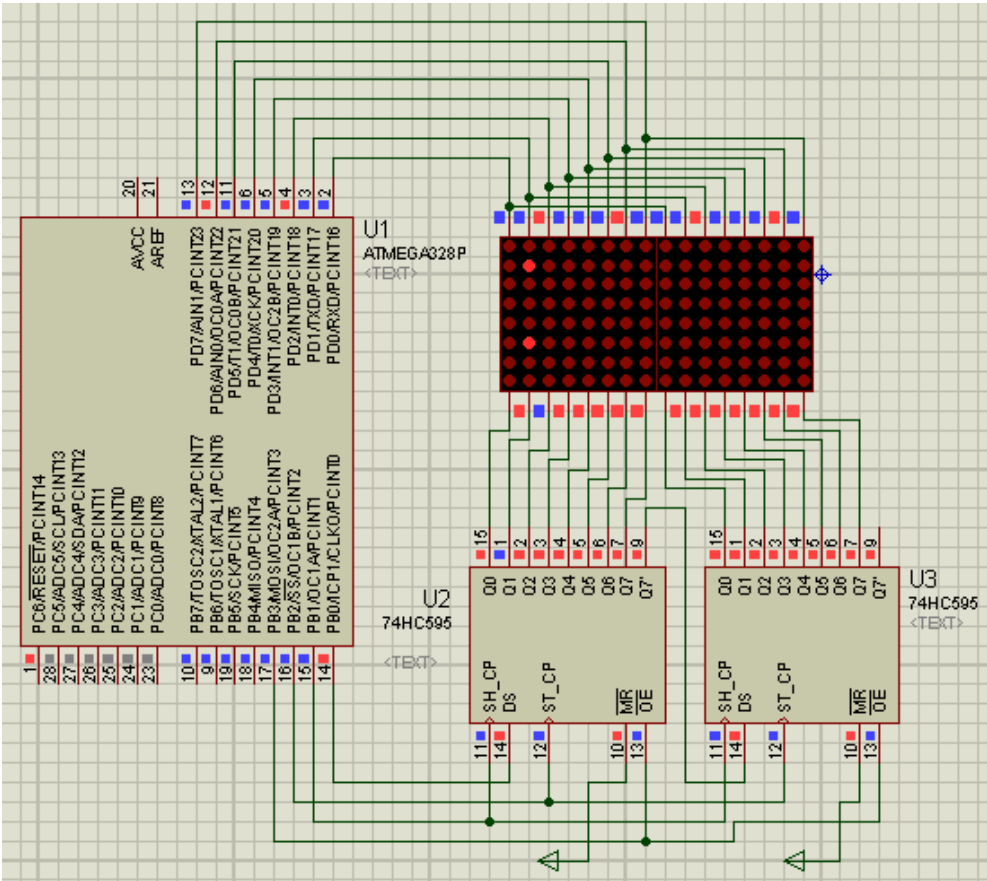
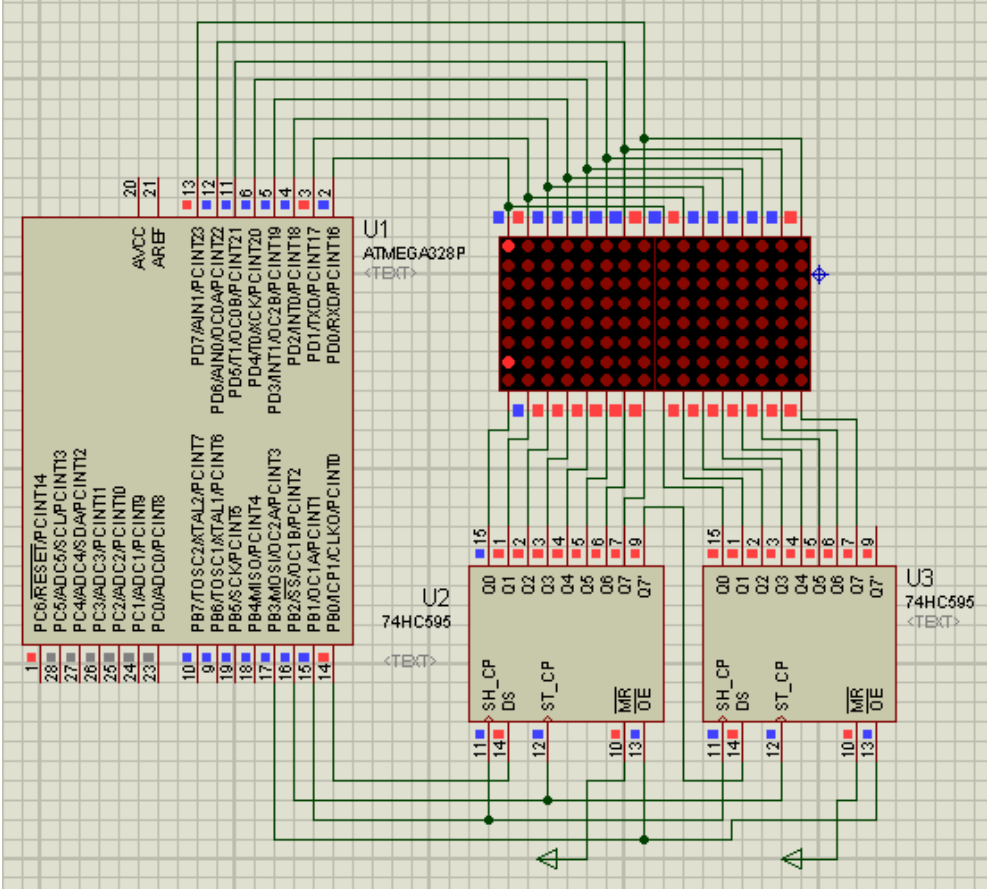
Table_x:
' primeiro o X normal
Data &B10000010 , &B01000100 , &B00101000 , &B00010000
Data &B00101000 , &B01000100 , &B10000010 , &B00000000
' agora segue o X invertido
Data &B01111101 , &B10111101 , &B11010111 , &B11101111
Data &B11010111 , &B10111101 , &B01111101 , &B11111111

```

Tamanho do programa compilado é de 810 bytes.

Seguem algumas telas ilustrando os sinais a cada instante para ver a montagem dos Leds.

As telas mostram respectivamente a primeira coluna, a segunda coluna, a décima-quinta coluna, e por fim, o efeito que fica devido ao Multiplex, onde vemos todas as colunas acesas ao mesmo tempo.





Como prometi antes, segue o mesmo programa modificado para o caso de usar os drivers UDN e ULN para aumentar bastante o brilho de nossas matrizes.

```
'-----  
' PROGRAMA MULTIPLEX MATRIX3 - Implementa o Multiplex Basico para  
' fazer acender um X normal na primeira matriz e o inverso desse X na  
' segunda matriz. Usamos agora o 74HCT595 para controlar as matrizes  
' Usaremos uma Matriz de 16 bytes para armazenar os Leds das duas matrizes.  
'  
' Este programa utiliza o Timer0 de 8 bits para a base de tempo  
' e implementa um refresh de 480 Hertz aproximado.  
' Versão modificada para o uso de drivers UDN e ULN  
'-----
```

```
$crystal = 8000000  
$regfile = "m328def.dat"  
$hwstack = 40  
$swstack = 16  
$framesize = 32
```

```
Dim Disp_index As Byte  
Dim Disp_aux As Byte  
Dim Coluna(16) As Byte  
Dim Nada As Word  
Dim Flag As Bit
```

```
Config Portb = Output  
' PortB vai acender as COLUNAS
```

```
Config Portd = Output  
'PortD vai controlar os 74HCT595
```

```
Ds Alias Portb.0  
Shcp Alias Portb.1  
Stcp Alias Portb.2  
Oe Alias Portb.3
```

```
' Vamos agora carregar nossos 16 bytes com o desenho  
' do X e do inverso do X que estão na tabela TABLE_X  
For Disp_index = 0 To 15  
    Disp_aux = Lookup(disp_index , Table_x)  
    Incr Disp_index  
    Coluna(disp_index) = Disp_aux  
    Decr Disp_index  
Next Disp_index
```

```
Config Timer0 = Timer , Prescale = 256  
Timer0 = 191  
' vamos gerar uma interrupção a cada 2,08 milissegundo  
' que corresponde a uma frequencia de Multiplex de 480 Hertz  
' ou seja, cada coluna vai acender 30 vezes por segundo !
```

```
On Timer0 Timer0_sub  
Enable Timer0
```

Enable Interrupts

Nada = 0

' é apenas um padrão de 16 bits 0 , que vamos usar para não  
' selecionar nenhuma coluna para iniciar o processo !

Oe = 0

Shiftout Ds , Shcp , Nada , 1

' Envia os 16 bits 0 aos shift registers

Stcp = 1

NOP

Stcp = 0

' armazenou e colocou todos os 16bits 1 na saída dos latches

Shcp = 0

Ds = 1

Disp\_index = 0

Do

Waitms 100

Loop

End

'----- ROTINAS DE INTERRUPÇÃO -----

Timer0\_sub:

' Rotina chamada pelo Timer0

Timer0 = 191

'recarrega o timer novamente

' apaga todos os dígitos

Portd = 0

' vamos ver se já fizemos o ultimo digito, pois então teremos de

' começar pelo primeiro novamente, carregando 16 bits 1 !

If Disp\_index > 16 Then

Flag = 1

Shiftout Ds , Shcp , Nada , 1

Stcp = 1

NOP

Stcp = 0

Disp\_index = 0

Ds = 1

End If

' agora vamos carregar um nivel 1 para a próxima coluna a mostrar

' uso um truque aqui. Se eu acabei de inicializar tudo com os 16 bits 1,

' eu deixo DS=0 para que um nivel zero seja introduzido no shift

' register, assim vai fazer acender uma coluna. A seguir, faço DS=1

' e todos os outros shifts vão apenas propagar uma só coluna com

' nivel 0.

Shcp = 1

NOP



```

Shcp = 0
Stcp = 1
NOP
Stcp = 0
Ds = 0

Incr Disp_index
' aqui já apontamos para a coluna

' Se não zeramos os shifts, temos de ter um delay
If Flag = 0 Then
    Waitus 30
Else
    Flag = 0
End If
' Acendemos a coluna

' pegamos os Leds que queremos mostrar nesta coluna
' dentro da nossa matriz coluna()
Portd = Coluna(dispc_index)
Return

' TABELA DE Leds A ACENDER

Table_x:
' primeiro o X normal
Data &B10000010 , &B01000100 , &B00101000 , &B00010000
Data &B00101000 , &B01000100 , &B10000010 , &B00000000
' agora segue o X invertido
Data &B01111101 , &B10111011 , &B11010111 , &B11101111
Data &B11010111 , &B10111011 , &B01111101 , &B11111111

```

Por ultimo, apresento uma restrição ao uso desta técnica para uma grande quantidade de matrizes.

Neste ultimo programa feito para o uso de drivers ULN e UDN, dentro da rotina de interrupção, toda vez que vamos iniciar para fazer a primeira coluna, fazemos 16 shifts com o valor 0 para poder apresentar zero em todas as saídas dos 74hct595 .

Isto demora cerca de 40 micro segundos, e portanto é bem rápido. Imagine se você resolve usar 10 matrizes, ao invés de 2, mesmo adequando o programa, perderemos mais de 300 micro segundos só para zerar os 10 shifts. Isto já pode se tornar bastante demorado, pois se vamos utilizar 10 matrizes, nossa frequência de Multiplex será de 2400 hertz, com um tempo entre as interrupções de 416 micro segundos.

Repare que quase não temos folga de tempo para algum processamento adicional. Imagine se você quiser utilizar 12 ou mais matrizes, este programa não vai funcionar mais.

Para se evitar isto, ao invés de controlarmos o pino do /OE, podemos ligar esse sinal diretamente ao terra, e controlar o pino /MR , onde com um simples pulso já fazemos o

Reset de todos os 74hct595, que já vão apresentar nível 0 em todas as saídas. Assim, melhoramos bastante o programa, e teremos sempre um tempo muito curto na rotina de interrupção.

Modificado desta forma, nossa rotina de interrupção sempre vai ser menor do que 40 micro segundos, e assim podemos utilizar até 40 matrizes ( na teoria ! ) . Relembro aqui que na prática é necessário um pequeno delay para permitir que a matriz apague totalmente, e em meus testes precisei manter em cerca de 30 micro segundos. Este é o motivo de utilizar este pequeno delay na rotina de interrupção. Se não fosse necessário, nossa rotina não iria demorar nem 10 micro segundos !

Finalmente, vamos ver agora outra maneira de se trabalhar com muitas matrizes, que é utilizando novamente o nosso velho MAX7219.

O uso deste CI simplesmente elimina todos os nossos problemas de tempos com a Multiplexação, elimina os drivers de corrente para aumentar o brilho, elimina o uso dos 74HCT595, e ainda permite controlar o brilho em até 16 níveis.

### CONTROLANDO VÁRIAS MATRIZES COM O MAX7219

Você já viu, acima, como que podemos usar até 8 displays de 7 segmentos com um único MAX7219. Porém, nada impede que usemos esse mesmo CI para controlar uma matriz de Leds no formato 8X8, pois esse CI possui um modo chamado **no-decode**, e nesse modo ele faz correspondência direta nas suas saídas conforme o valor binário apresentado a ele. Por exemplo, se mandarmos apresentar o numero binário 10101010, teremos na saída correspondente ao dígito os Leds acesos, de baixo para cima, exatamente correspondendo um Led aceso para um bit 1 .

Então, podemos usar esse CI, e encadeando vários deles, sendo que a entrada do segundo vai ligado à saída do primeiro, e assim por diante. Qual o limite disto?

Teoricamente, o único limite é o tempo que demora para podermos entregar os dados para todos as 8 colunas de cada um dos displays.

Para você ter uma idéia, se usarmos 10 matrizes, temos de enviar serialmente 16 bits para cada coluna x 8 ( numero de dígitos por matriz ) x 10 ( numero de matrizes ) = 1280 bits . Se enviarmos um bit a cada 5 microsegundos, vamos demorar mais de 6 milisegundos para atualizar as matrizes.

Isto acaba causando um problema quando o número de matrizes é grande. Se compararmos com o processo utilizando 74hct595, usaríamos apenas 8 bits para cada coluna, totalizando 640 bits, e o tempo seria reduzido à metade, pouco mais do que 3 milisegundos.

Vamos a seguir apresentar um esquema de ligação para o uso de duas matrizes, e irei apresentar um programa que permite o uso de 2 até 31 matrizes, sem nenhuma modificação, além de indicar o número de matrizes que vamos usar !

Os dados são enviados a partir do pino Portc.0 do microprocessador, e chega à entrada Din do Max7219 mais à direita. Dele, sai um sinal, através do pino Dout, que vai ligado à entrada Din do Max7219 mais à esquerda.

Com 2 matrizes, temos  $2 \times 8 = 16$  colunas.

E a ultima coluna, que é a que está mais à direita, vai mostrar o numero armazenado na ultima posição, que é a DÍgitos(16).

Os dados são enviados serialmente. Assim, enviamos 16 bits de dados, que fazem o endereçamento da coluna correspondente e também já informam o valor a ser mostrado, e a seguir enviamos mais 16 bits serialmente, para o segundo display. Desta maneira, os 16 bits enviados primeiro vão acabar ficando dentro do MAX7219 à esquerda, e os últimos 16 bits ficam dentro do MAX7219 à direita! A seguir, damos o comando de Armazenar.

O programa para isso é muito simples, escrito também com o Bascom. E o melhor é que você pode adequar o mesmo programa para mostrar até 31 matrizes!

Segue abaixo :

```

' -----
' PROGRAMA Matrix4 - Usando matrizes com os MAX7219
'
' Permite o uso de 2 até 31 matrizes de 8x8 , criando uma unica no formato
' de 248 x 8.
' Os valores a serem mostrados estão armazenados em uma matrix de tamanho
' 8 vezes o numero de matrizes, no nosso caso de tamanho 16.
' É um programa que serve de ponto de partida para um display mostrador
' de mensagens de texto.
' -----

$crystal = 8000000
'vamos usar o clock interno que é de 8 Mhz.
$regfile = "m328Pdef.dat"
' vamos compilar para o ATMEGA328P
$hwstack = 40
$swstack = 32
$framesize = 350

' a seguir, é onde especificamos a quantidade de matrizes, basta mudar
' para quantas voce quiser !
Const N = 2
' N= numero de matrizes ( limitado entre 2 e 31 )

Const N1 =(n) * 8
Const N2 =(n) - 1

Dim X As Byte
Dim Y As Byte
Dim Z As Byte
Dim Base As Byte
Dim Numero As Word
Dim I As Byte
Dim Digitos(n1) As Byte

' Esta matrix Digitos() vai conter todos os valores a serem mostrados
' desde a primeira coluna ( digitos(1) ) até a ultima coluna
' ( digitos(8Xnumero de matrizes) )

Const Reg_decode_mode = &B1111100100000000
Const Reg_scan_limit = &B1111101111111111
Const Reg_intensity = &B1111101011111111
Const Reg_shutdown_off = &B1111110011111111
Const Reg_shutdown_on = &B1111110011111110

Config Portc.0 = Output           'data
Config Portc.1 = Output           'load
Config Portc.2 = Output           'clock

Max_data Alias Portc.0
Max_load Alias Portc.1
Max_clock Alias Portc.2

' vamos colocar alguns valores na matrix
For I = 1 To N1
    Digitos(i) = I

```

Next I

Enable Interrupts

'vamos agora inicializar os MAX7219

Numero = Reg\_shutdown\_off  
Gosub Init\_max

Numero = Reg\_decode\_mode  
Gosub Init\_max

Numero = Reg\_intensity  
Gosub Init\_max

Numero = Reg\_scan\_limit  
Gosub Init\_max  
'já tudo configurado !

'a seguir, loop do programa principal  
' dentro do trecho do DO...LOOP pode ser colocado todo o programa  
' que poderia tratar as mensagens e apresentar no display  
' para fazer um display de mensagens rolantes.  
' o programa deveria pegar as letras, e procurar os bytes correspondentes  
' em uma tabela, e guardar na matriz toda a mensagem. A seguir, bastaria  
' apenas ficar "rodando a matriz", isto é, pegar o primeiro elemento e  
guardar  
' ele em uma variável, e em seguida pegar o segundo elemento e armazenar no  
' lugar do primeiro, e assim em diante, sendo que finalmente o elemento  
' que guardamos logo no início será armazenado no ultimo elemento, e podemos  
' atualizar o display.

Do

Waitms 1000

Gosub Sai\_max

'atualiza

matrizes

Loop

'---- sub-rotina para atualizar matrizes -----

Sai\_max:

For Y = 1 To 8

For X = 0 To N2

Base = X

Shift Base , Left , 3

Numero = Y

Shift Numero , Left , 8

Z = Base + Y

Numero = Numero + Digitos(z)

Shiftout Max\_data , Max\_clock , Numero , 1

Next X

Max\_load = 1

Max\_load = 0

Next Y

Return

'---- sub-rotina para enviar comandos aos Max7291

```

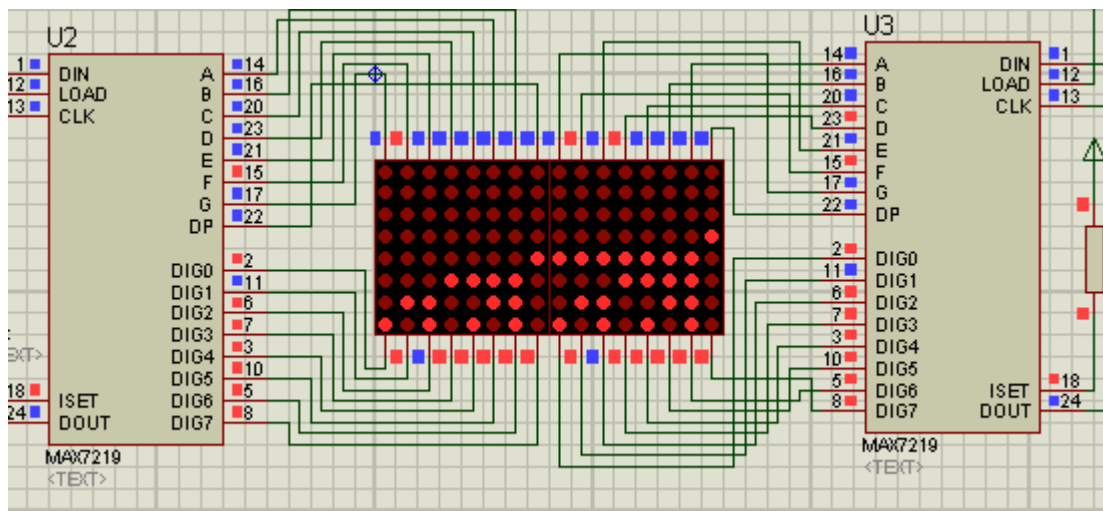
Init_max:
For X = 1 To N
    Shiftout Max_data , Max_clock , Numero , 1
Next X
Max_load = 1
Max_load = 0
Return

End

```

Este programa compilado tem o tamanho de 746 Bytes.

A seguir, o que você veria nessas matrizes :



Repare que o programa preenche a matriz com os números 1,2,3 e sucessivamente até 16 , e você está vendo a saída, mostrando de 1 até 16 da esquerda para a direita, com os Leds correspondentes aos bits 1 acesos.

É um excelente ponto de partida para você fazer um display mostrador de Leds com mensagens de texto que ficam andando da direita para a esquerda!

Basta você armazenar a mensagem, por exemplo, na Eeprom do microcontrolador ou numa Eeprom externa, e no início do programa você pegaria cada letra da mensagem, consulta uma tabela que mostra como seriam cada uma das colunas que fazem esse caracter, e armazenar todas as colunas na matriz.

A partir daí, é só ficar fazendo uma rotação da matriz, da direita para a esquerda, na velocidade que você desejar, isto é, o elemento Dígitos(n) toma o valor do elemento Dígitos(n+1) , para toda a matriz. Se quiser fazer com que o elemento mais à esquerda volte ao final da matriz, você vai ter sua mensagem circulando todo o tempo.

Quanto a tempos de execução, eu fiz uma simulação para 31 matrizes, e demora cerca de 18,5 milisegundos para fazer toda a atualização do display. Se quiser, pode usar um cristal oscilador externo, de 20 MHz, e nesse caso o tempo cairia para 7,4 milisegundos.

Creio que ao chegar até aqui você já sabe tudo o que precisa saber para trabalhar com qualquer tipo de display de Leds.

Boas Montagens !