



## Módulo 3 – Comunicação Serial e Conv. A/D

### ORIENTAÇÕES:

O módulo 3 trata de comunicação serial e conversão de sinais analógicos em digitais. Este módulo vale 20 pontos. Toda semana será selecionado um exercício para receber visto. Serão 3 vistos ao total. Cada visto vale 3 pontos. No final do módulo, um problema será proposto para sumarizar o conteúdo do módulo. O problema vale 11 pontos.

### OBJETIVOS DESTA MÓDULO:

- Compreender protocolos de comunicação serial síncrona e assíncrona.
- Diferenciar bit-banging do uso do periférico dedicado.
- Entender o processo de amostragem e conversão de um sinal analógico em digital.
- Configurar as interfaces seriais e o conversor A/D.

## Comunicação Serial

Comunicação serial é o processo de envio de dados de maneira sequencial, um bit após o outro. Esse tipo de comunicação permite economizar (preciosos) pinos de entrada e saída ao realizar transferências de grandes quantidades de dados de maneira serial. Ao invés de 16 pinos para enviar uma palavra de 16-bits é necessário apenas 1.

Em sistemas embarcados, dispositivos com funcionalidade específica geralmente implementam algum tipo de comunicação serial para enviar ou receber dados e instruções. Acelerômetros, cartões de memória, GPS, sensores de temperatura, umidade e pressão, todos possuem internamente um microcontrolador que trata o dado bruto e o envia por uma linha serial. Diante disso, é de suma importância compreender as diferentes formas de comunicação para interfacear o microcontrolador com outros dispositivos.

Podemos enquadrar a comunicação serial em duas classes:

- **Síncrona**, na qual é disponibilizado um relógio que indica o instante de validade do bit. Exemplos: SPI, I2C
- **Assíncrona**, quando não há a disponibilidade deste relógio. Exemplos: UART, USB.

## Comunicação Serial Síncrona: I<sup>2</sup>C

O protocolo I<sup>2</sup>C (*Inter-Integrated Circuit Communication*) permite que vários dispositivos se conectem a um mesmo barramento e se comuniquem entre si. Esse barramento, também conhecido como TWI (*Two Wire Interface*) utiliza apenas dois fios para realizar a comunicação bidirecional entre diversos dispositivos. Na terminologia utilizada, o dispositivo que está gerando uma mensagem é chamado de transmissor, enquanto o dispositivo que está recebendo a mensagem é chamado de receptor. O dispositivo que controla o barramento é chamado de mestre (master), e os dispositivos que respondem ao mestre são chamados de escravos (slave). Os dispositivos mestre e escravo assumem o papel de transmissor ou receptor em diferentes etapas da comunicação. O fio do barramento que carrega dados é chamado de SDA (Serial DATA). O outro fio estabelece o ritmo de comunicação e é chamado de SCL (Serial CLock). Apenas o mestre controla o clock, mesmo quando estiver recebendo uma mensagem.

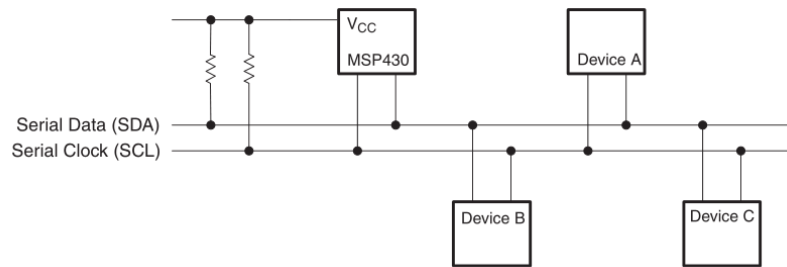


Figura 1 – Barramento I2C

### Sinais de controle: Start e Stop

Quando o barramento está ocioso as linhas SDA e SCL ficam em nível lógico HIGH. Isso é garantido através de resistores de pull-up como os da Figura 1. Para iniciar uma transmissão, o mestre força a descida de SDA (mudança de HIGH para LOW) enquanto o clock está em nível alto. Este sinal é a condição de partida (START), e prepara os dispositivos para a comunicação. Para finalizar a transmissão, o mestre libera SDA (mudança de LOW para HIGH) quando o clock está em nível alto. Este sinal é a condição de parada (STOP) liberando assim o barramento para outras comunicações. Se for necessário reestabelecer a comunicação, deve-se gerar novamente a condição de partida. A Figura 2 apresenta as condições de START e STOP.

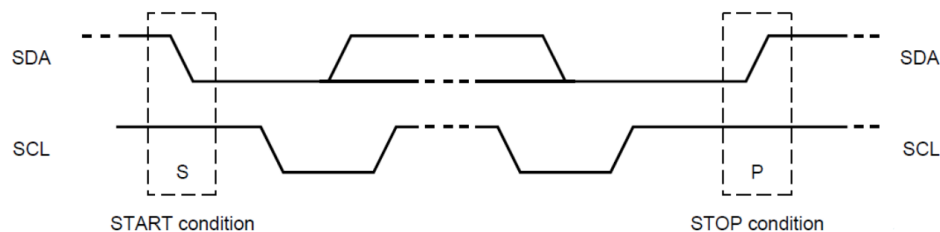


Figura 2 – Comandos de Start e Stop

### Transferência de 1 bit

Na transferência de um bit, como mostrado na Figura 3, o dado em SDA deve permanecer estável entre a subida e a descida do clock (SCL). Mudanças em SDA enquanto SCL está em nível alto são entendidas como sinais de controle (como visto nas condições de START e STOP). Durante a transmissão, o valor de SDA é constante enquanto SCL estiver em nível HIGH e assume o valor do bit a ser transmitido.

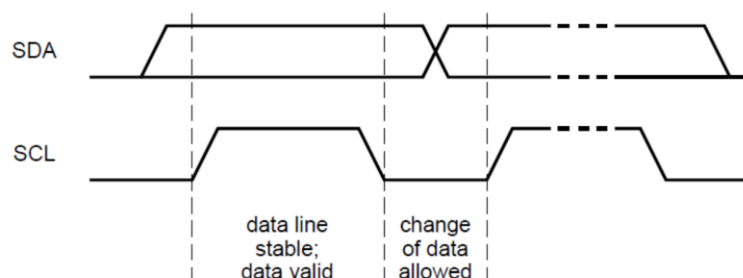


Figura 3 – Transferência de 1 bit

### Transferência de uma palavra (8-bits) + ACK/NACK

A comunicação no barramento é feita em bytes (8 bits). A cada byte transmitido, um sinal de *acknowledge* (ACK) deve ser gerado. Portanto, para cada byte transmitido, 9 batidas de clock precisam ser geradas. O número de bytes enviados entre um sinal de START e STOP é ilimitado. A figura 4 mostra a comunicação entre dois dispositivos. O transmissor envia os 8 primeiros bits e libera a linha no nono bit. O receptor então puxa a linha para LOW indicando que entendeu a mensagem, com um ACK. Se a linha ficar em HIGH na nona batida do relógio significa um “*not-acknowledge*” (NACK) e pode ter diversos significados: pode ser que o escravo não entendeu a palavra enviada; ou que não há nenhum dispositivo na linha escutando; ou que o escravo decidiu enviar um NACK de propósito para indicar que não quer ou não pode mais receber bytes.

Na Figura 4, as ações do mestre-transmissor e escravo-receptor são apresentadas separadas, porém ambos compartilham a mesma linha SDA. Isso deixa claro que é o escravo (e não o mestre) que envia o ACK no momento certo. Note que o clock é controlado pelo mestre, inclusive na batida do ACK. Outra curiosidade interessante, é que no momento do ACK, o escravo deixa de ser receptor para transmitir o bit de ACK/NACK e o mestre deixa de ser transmissor para receber o bit de ACK/NACK. Isso reforça a afirmação anterior que diz que o mestre e escravo assumem papéis de transmissor e receptor em diferentes estágios da comunicação.

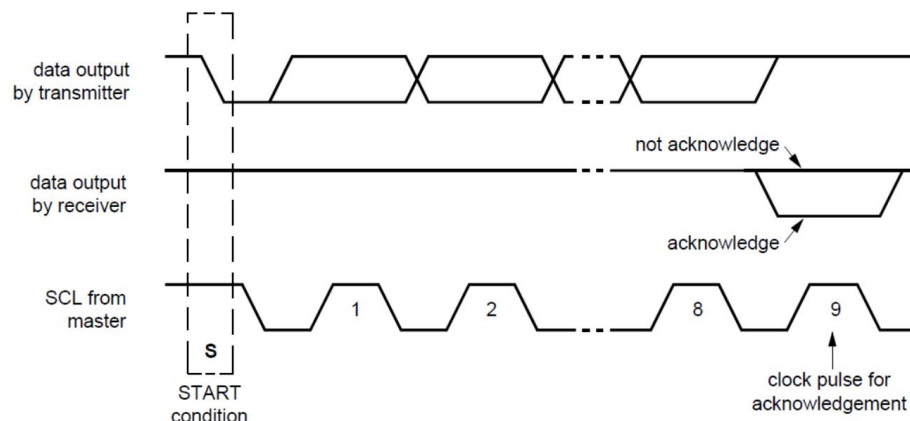


Figura 4 - Transferência de 1 byte

### Endereçamento

O primeiro byte enviado logo após a condição de START é especial e é interpretado como endereço do escravo. São enviados 7 bits de endereço e um bit de leitura/escrita que indica se o mestre vai ler ou escrever no escravo, como mostrado na Figura 5. O endereço é enviado na parte mais significativa do primeiro byte e o bit de R/W é enviado por último, ou seja, ele é o LSB.

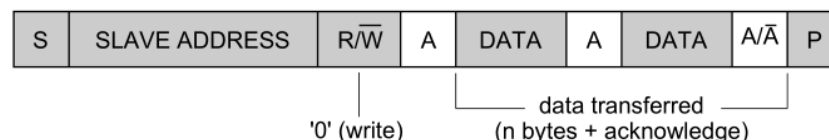


Figura 5 - Endereçamento de um escravo.

Após o envio dessa palavra inicial, as próximas são sempre bytes (8 bits) seguidos do ciclo de ACK/NACK. Por fim, termina-se a comunicação com uma condição de STOP (P).



No MSP430 usamos a interface dedicada USCI, sigla para Universal Serial Communication Interface. Ela é capaz de se comunicar em UART, SPI e I2C. Nesta primeira parte vamos estudar apenas o protocolo I2C. A interface pode ser configurada pelos registros CTL1 e CTL0 mostrados na Figura 6:

UCBxCTL0							
7	6	5	4	3	2	1	0
UCA10	UCSLA10	UCMM	Reserved	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-1

UCBxCTL1							
7	6	5	4	3	2	1	0
UCSSELx		Reserved	UCTR	UCTXNACK	UCTXSTP	UCTXSTT	UCSWRST
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-1

Can be modified only when UCSWRST = 1.

Figura 6 - Registros de configuração

Perceba que alguns bits (hachurados) só podem ser escritos enquanto a interface estiver resetada, ou seja, RST = 1. Para o modo I2C, devemos fazer SYNC = 1, MODE = 3. O bit MST indica se a interface vai se comportar como mestre ou escravo, ou seja, se vai controlar o clock ou só escutar o clock.

#### Configuração como mestre:

Quando a interface for configurada como mestre, devemos escolher o clock nos bits SSEL. A frequência do clock pode ser dividida pelo valor escrito em UCBxBRW. O bit TR configura a interface como transmissora (TR=1) ou receptora (TR=0). Os bits STT e STP são usados para enviar a sinalização de start e stop, respectivamente. O endereço do escravo que o mestre quer se comunicar deve ser escrito em UCBxI2CSA (Slave Address). O valor deve estar disponível antes da sinalização de start. Durante a comunicação, os bytes podem ser enviados por TXBUF e recebidos por RXBUF. A flag TXIFG sinaliza quando o buffer de transmissão estiver vazio e pode ser escrito. Já a flag RXIFG sinaliza quando o buffer de recepção estiver cheio e deve ser lido. Recomenda-se uma implementação a base de polling para o mestre.

#### Configuração como escravo:

Não há a necessidade de se configurar o clock quando a interface for configurada como escravo. O endereço do escravo deve ser configurado no registro I2COA (Own Address). Recomenda-se uma implementação a base de interrupções para o escravo. Os eventos de TX, RX, NACK, STT e STP podem ser decodificados a partir do vetor de interrupções (IV) associado à interface.

O MSP430F5529 possui duas instâncias da USCI chamadas de B0 e B1. A interface B0 está mapeada nos pinos P3.0 (SDA) e P3.1 (SCL). Já a interface B1 está nos pinos P4.1 (SDA) e P4.2 (SCL). Você deve habilitar os resistores de pull-up internos para garantir o bom funcionamento da interface ou usar resistores externos. Recomendamos não ultrapassar 100kHz para o clock. Os resistores internos do MSP430 são elevados e limitam a velocidade de operação da linha. Caso queira atingir a velocidade de 400kHz, use resistores externos de 4k7.

A configuração deve ser feita na seguinte ordem: (1) Reseta a interface fazendo o bit RST = 1. (2) Configura todos os registros de controle. (3) Configura os pinos. (4) Desativa o reset fazendo RST = 0. (5) Habilita as interrupções que forem necessárias.

**Exercício 1: Teste da interface I<sup>2</sup>C.**

O MSP430 possui duas interfaces I<sup>2</sup>C completas, vamos usá-las para testar a comunicação serial. Configure a interface **B0 como mestre** @100kHz e a interface **B1 como escravo**. Faça a conexão dos pinos P3.1 com P4.2 e P3.0 com P4.1, como na Figura 7. Faça a conexão para ensaio com as duas USCI que operam com I<sup>2</sup>C.. Perceba que apenas o mestre controla o clock. Configure o endereço 0x34 para o escravo e teste o envio do byte 0x55. *Não se esqueça de habilitar os resistores de pull-up ou coloque resistores externos.*

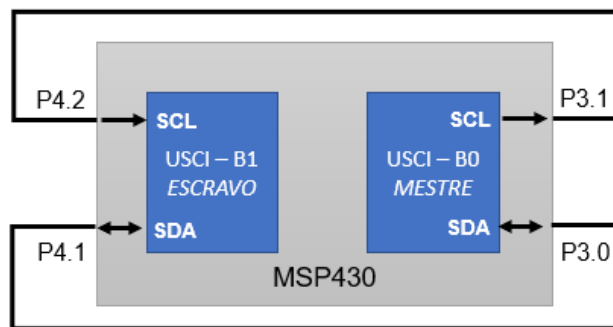


Figura 7 - Conexão para ensaio com as duas USCI que operam com I<sup>2</sup>C.

Para este exercício, escreva a função `uint8_t i2cSend(uint8_t addr, uint8_t data)` para fazer o mestre enviar um byte para o escravo. A função deve retornar o valor do ACK/NACK, ou seja, 0 se houve ACK e 1 se houve NACK. Recomenda-se a implementação do software seguindo as etapas abaixo:

1. Escreva o endereço do escravo em I2CSA
2. Faça TR = 1 e STT = 1 para requisitar um start como transmissor
3. Escrever em TXBUF quando TXIFG for p/ 1
4. Esperar o ciclo de ACK/NACK observando quando a STT vai p/ 0
5. Se houver NACK, enviar um stop com STP = 1 e retornar o NACK
6. Caso contrário, aguardar TXIFG p/ pedir o stop (STP=1) e retornar o ACK.

*Sugestão extra: Generalizar essa função para enviar nBytes.*

Use interrupções para receber o byte da interface escravo ou aguarde a flag RXIFG sinalizar o recebimento do byte e em seguida faça a leitura de RXBUF.

**Exercício 2: Scanner de endereços I<sup>2</sup>C**

Aproveite a configuração do exercício anterior para implementar um scanner de endereços do barramento I<sup>2</sup>C. O mestre deve varrer todos os endereços disponíveis no barramento e retornar uma lista com os endereços dos dispositivos conectados ao barramento. *É recomendada a técnica de amostragem (polling) das flags da interface USCI ao invés de interrupções.*

## Uso de um LCD pela interface I2C.

Além de LEDs, é comum utilizar displays LCD, como o da Figura 8, em aplicações com microcontroladores. A grande maioria dos displays do mercado utiliza o controlador HD 44780, que pode ser facilmente encontrado com baixo custo.



Figura 8 - LCD de 2 linhas e 16 colunas de caracteres.

Esse display tem 16 pinos de interface – 5 pinos de alimentação, contraste e backlight, 8 pinos de dados D[7:0] (formando 1 byte) e 3 pinos de controle ( $RS$ ,  $R/\overline{W}$  e  $EN$ ). Seu funcionamento está baseado em 3 registradores:

- IR → registrador de instrução (recebe os comandos a serem executados)
- DR → registrador de dado (recebe o código ASCII da letra a ser impressa no display)
- AC → contador de endereço (indica posição atual do cursor, local onde será impressa a próxima letra).

Os pinos de controle  $RS$ ,  $R/\overline{W}$  e  $EN$  viabilizam a comunicação com a interface do LCD. Uma escrita com  $RS = 0$  permite enviar instruções (escreve em IR). A leitura com  $RS = 0$  permite ler a posição atual do cursor (AC), junto com o bit indicativo de ocupado que é colocado na posição mais à esquerda (bit 7). A escrita de um caractere é feita com  $RS = 1$ . O controlador do LCD aguarda um pulso no bit  $EN$  (Enable) para realizar a leitura/escrita dos dados nos pinos D[7:0].

Tabela 1 - Controle de acesso ao LCD.

$RS$	$R/\overline{W}$	Reg de destino	Operação
0	0	IR	Escrever uma instrução em IR
0	1	AC	Ler o <i>bit</i> de ocupado (b7) e o valor de AC (b6 – b0)
1	0	DR	Escrever no DR (operação sobre DDRAM ou CGRAM)
1	1	DR	Ler o DR (operação sobre DDRAM ou CGRAM)

A Figura 9 apresenta a temporização para acessar o LCD. É importante seguir esse diagrama de tempo já que o MSP430 é capaz de escrever nas portas do LCD mais rápido do que o próprio LCD consegue ler. Faz-se necessário o uso de pequenos atrasos para garantir a temporização especificada.

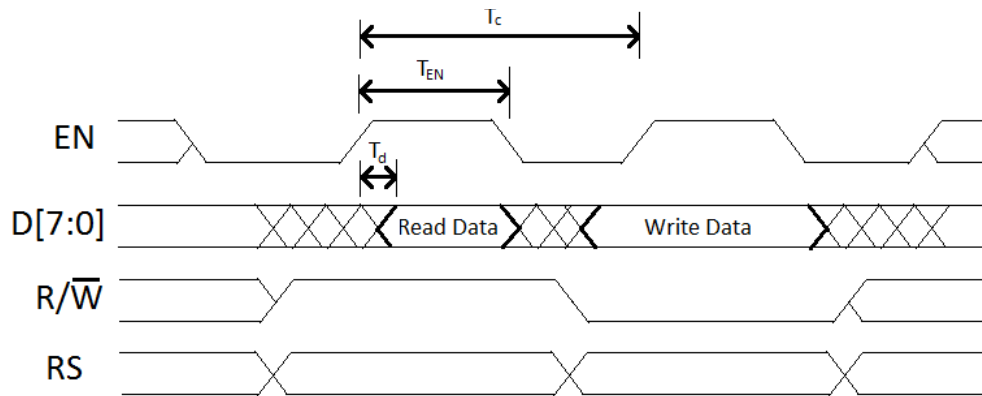


Figura 9 – Temporização de escrita/leitura no LCD

A duração do pulso de enable não deve ser menor que  $T_{EN} = 300ns$  e deve ser espaçado de  $T_C = 500ns$  no mínimo. Em modo de leitura, o dado fica disponível em D[7:0]  $T_d = 90ns$  após a subida do sinal enable. Em modo de escrita, o dado deve estar presente em D[7:0] antes da subida do pulso de enable. Os comandos e instruções do controlador de LCD estão resumidos na tabela 2.

O protocolo completo exige o uso de 11 pinos, entretanto, a quantidade de pinos de um microcontrolador é um recurso limitado e geralmente escasso. Alguns microcontroladores do mercado possuem apenas 6 pinos utilizáveis. Com tão poucos pinos, é impossível implementar o protocolo LCD em modo paralelo. Uma alternativa muito comum é usar a configuração do LCD para barramento de dados de 4 bits. Neste caso, a transação de cada byte é feita com 2 acessos de 4 bits, o primeiro para os MSBs e o segundo para os LSBs. A Figura 10 mostra o caso de uma escrita no LCD.

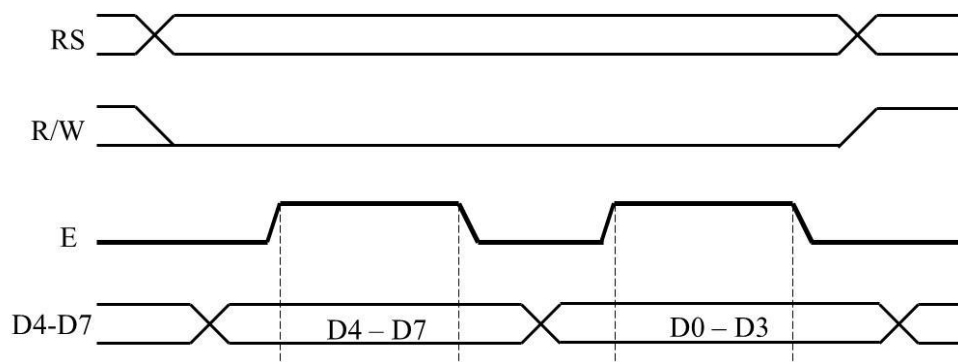


Figura 10 – LCD em modo 4 bits.

No modo de operação em 4 bits, são necessários apenas 7 pinos, que podem ser reduzidos para 6 se as operações forem apenas de escrita. Se a quantidade de pinos ainda for um fator limitante do projeto, pode-se usar um conversor de serial para paralelo através de um registrador de deslocamento com portas bidirecionais.



O circuito integrado PCF8574 faz exatamente isso. Ele possui uma interface serial síncrona I2C que escreve num registrador de deslocamento transformando dados enviados de maneira serial num barramento paralelo de 8 bits. Desses 8 bits, 3 são usados para controle ( $RS$ ,  $R/\bar{W}$  e  $EN$ ), 4 bits para o barramento de dados reduzido (modo de 4 bits) e 1 bit para ligar e desligar a retroiluminação (backlight) do LCD, como mostrado na Figura 11, no pino 16 do LCD.

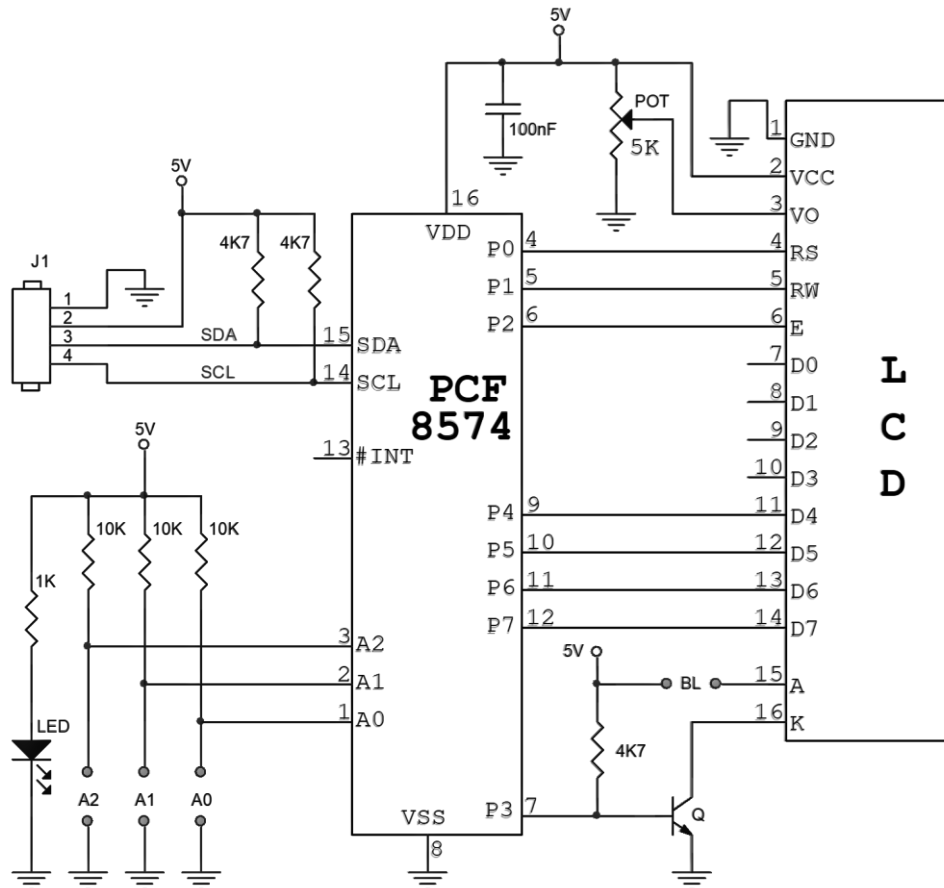


Figura 11 – Esquemático do conv. serial paralelo e sua conexão com o LCD

É de se notar que o PCF8574 opera em 5V enquanto o MSP opera em 3,3V. É preciso conferir se não há risco para o MSP quando ele recebe a linha SDA em nível alto, o que no caso é feito pelo resistor de pull-up. Sabemos que o diodo de proteção entra em ação quando a tensão no pino ultrapassa 3,6 V.



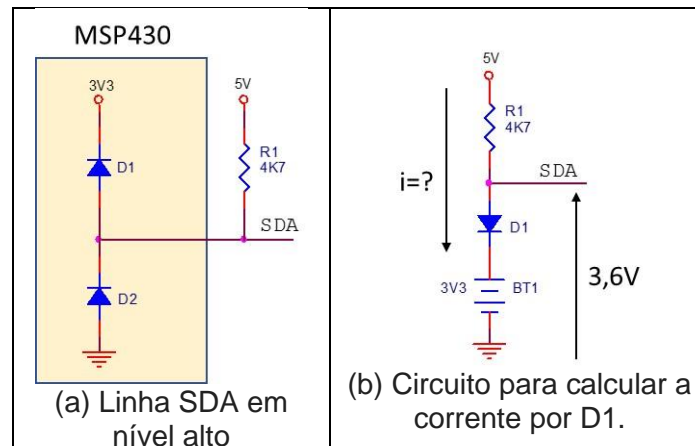


Figura 12 – Circuito para o cálculo da corrente pelo diodo de proteção (D1) quando a linha SDA é colocada em nível alto.

Para o caso da Figura 12, é fácil de ver que a corrente por D1 é de 300  $\mu$ A (vide equação), o que está bem abaixo do limite de 2 mA indicado pelo manual do MSP430F5529. Assim, podemos ligar diretamente o PCF8574 ao LaunchPad.

$$i = \frac{5 - 3,6}{4,7K} = 300\mu A$$

**Exercício 3:** Use a função `i2cSend(...)` p/ ativar e desativar a retroiluminação do LCD. Note, pela figura 10, que é o BIT3 que controla a retroiluminação. Usando essa função, faça o LCD piscar em 1 Hz, aproximadamente. Na configuração da interface serial, mantenha a frequência da linha SCL abaixo de 100kHz. Note que existem duas versões do chip PCF8574 e um endereço para cada uma. Faça uma tentativa para ver qual o endereço de seu chip.

- PCF8574AT  $\rightarrow$  0x3F e
- PCF8574T  $\rightarrow$  0x27

**Exercício 4:** Escreva a função `uint8_t lcdAddr()` que retorna o endereço correto para operar seu LCD. Dica: verifique em qual dos dois endereços o escravo (PCF8574) responde. Busque inspiração no Exercício 2.

**Exercício 5:** Escreva uma função `lcdWriteNibble(uint8_t nibble, uint8_t isChar)` para escrever um nibble no LCD em modo 4 bits seguindo o diagrama ilustrado na Figura 9. Faça uso da função `i2cSend(...)` 3 vezes da seguinte forma:

1. Prepare RS e D[7:4] com EN=0 e R/W = 0 (write)
2. Em seguida mantenha R/W, RS e D[7:4] constantes e faça EN=1.
3. Por fim, faça EN voltar a 0 mantendo os outros campos constantes.

A variável `isChar` deve representar o valor de RS, ou seja, se for 1 o nibble representa parte de um caractere, se for 0 o nibble é parte de uma instrução.

**Exercício 6:** Escreva a função `lcdWriteByte(uint8_t byte, uint8_t isChar)`, chamando `lcdWriteNibble` duas vezes. O primeiro nibble enviado corresponde aos 4 bits mais significativos do byte conforme descrito na Figura 9.



**Exercício 7:** Escreva a função `lcdInit()` que inicializa o LCD seguindo o diagrama da Figura 13. Note que você irá utilizar a função `lcdWriteNibble` três vezes para garantir que o LCD parte de um estado conhecido, ou seja, do modo 8 bits. Em seguida, coloca-se o LCD em modo 4 bits chamando a função `lcdWriteNibble` uma quarta vez. A partir daí devemos usar apenas a função `lcdWriteByte` para escrever as próximas instruções. Lembre-se que toda instrução é escrita com `RS = 0`, ou seja, `isChar = 0`.

**Exercício 8:** Escreva a função `lcdWrite(char *str)` que escreve uma string no LCD e gerencia a quebra de linha. Lembre-se que uma string é representada por uma sequência de caracteres que termina no byte `0x00`.

**Exercício 9:** Repita a lógica dos exercícios 5 e 6 para fazer a leitura de um nibble e depois de um byte numa função `lcdReadByte(...)`. O momento ideal de leitura é no meio do pulso do `EN`.

**Exercício 10:** Use a rotina do exercício 9 para escrever a função `uint8_t lcdBusy()` e verificar se o LCD está ocupado. A função não recebe nenhuma parâmetro de entrada e deve retornar o valor do bit busy.

#### SUGESTÕES:

- Recomenda-se separar a resolução deste experimento em duas bibliotecas. Uma para interface serial (`libserial`) e outra para as funções de alto nível de acesso ao LCD (`liblcd`). Crie dois arquivos para cada biblioteca, um de cabeçalho contendo os protótipos das funções que você irá escrever e outro com o conteúdo das funções.
- Inicie o seu projeto pela biblioteca serial e em seguida escreva a biblioteca do LCD usando a sua biblioteca da interface serial.
- Recomendamos usar a técnica de amostragem de flags (polling) ao invés de interrupções para a interface serial I<sup>2</sup>C do MSP430.
- Na sua biblioteca de acesso ao LCD, escreva funções para controlar os sinais `E`, `RS` e `RW`. Teste **separadamente** essas 3 funções, o que pode ser feito facilmente com um multímetro. Depois crie uma função para escrita no barramento de dados.
- Escreva uma função específica para inicializar o LCD no modo de 4 bits (figura 12).
- Escreva uma função para escrever um caractere / instrução no LCD e use essa função numa outra que escreve uma sequência de caracteres (strings).
- Utilize defines para definir a temporização do LCD (tempo de ativação do `Enable`, tempo de espera entre instruções, etc).



*Tabela 2 - Instruções do controlador de LCD*

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ)
	RS	R/W	DB <sub>7</sub>	DB <sub>6</sub>	DB <sub>5</sub>	DB <sub>4</sub>	DB <sub>3</sub>	DB <sub>2</sub>	DB <sub>1</sub>	DB <sub>0</sub>		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC <sub>5</sub>	AC <sub>4</sub>	AC <sub>3</sub>	AC <sub>2</sub>	AC <sub>1</sub>	AC <sub>0</sub>	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC <sub>6</sub>	AC <sub>5</sub>	AC <sub>4</sub>	AC <sub>3</sub>	AC <sub>2</sub>	AC <sub>1</sub>	AC <sub>0</sub>	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC <sub>6</sub>	AC <sub>5</sub>	AC <sub>4</sub>	AC <sub>3</sub>	AC <sub>2</sub>	AC <sub>1</sub>	AC <sub>0</sub>	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Read data from internal RAM (DDRAM/CGRAM).	43us



## Inicialização do LCD

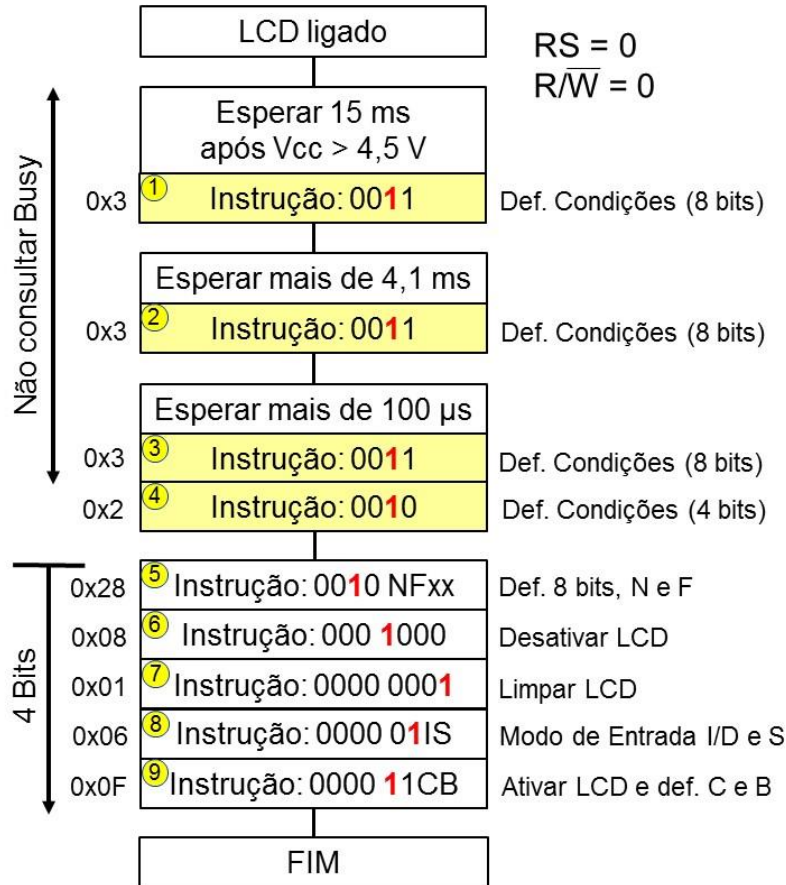


Figura 13 - Inicialização do mostrador LCD via software para barramento de 4 bits.

## Comunicação Serial Assíncrona

A comunicação serial assíncrona é provavelmente a forma serial mais simples de todas. Sua popularidade vem justamente da sua simplicidade e facilidade de implementar usando lógica digital ou microcontroladores.

Para cada linha serial, a comunicação está ociosa quando em nível alto. Para iniciar uma transmissão, é necessário puxar a linha para nível baixo e transmitir bit a bit de maneira sequencial, começando pelo menos significativo. A taxa de transferência é um acordo prévio, ou seja, ambas as partes devem estar configuradas para se comunicar na mesma velocidade.

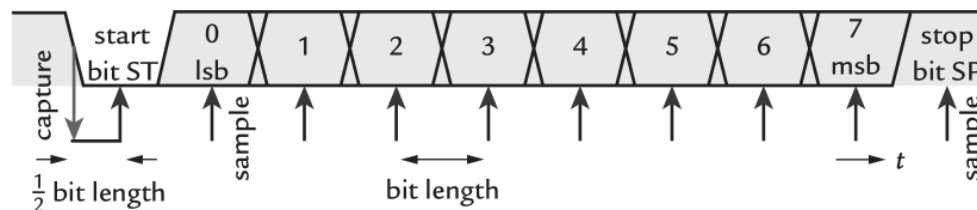


Figura 14 – Comunicação serial assíncrona

Um dispositivo UART (Universal Asynchronous Receiver/Transmitter) não permite comunicação bidirecional através de um único fio. São necessários pelo menos dois fios para uma comunicação full-duplex. Uma particularidade desta forma de comunicação é a ausência de sincronia entre o transmissor e receptor. Isso pode levar a erros de comunicação caso haja diferença significativa entre os clocks internos de cada dispositivo e limita a taxa de comunicação possível entre dispositivos. O máximo recomendado é de aproximadamente 100kbps.

O MSP430 possui um módulo de comunicação serial dedicado. Isso torna o processo de comunicação tão simples quanto escrever num registro e esperar que o byte seja enviado pela linha serial. Isso nem sempre acontece com microcontroladores mais simples. Nesses casos o protocolo deve ser implementado manualmente através de leituras e escritas dos pinos. Este tipo de implementação é chamado de *bit-banging* e exige cuidado extra com a temporização para não perder sincronia.

### Exercício 11:

A interface serial USCI-A0 está roteada nos pinos P3.3 (TX) e P3.4 (RX). Configure esses pinos para sua funcionalidade dedicada. Conecte os dois como na Figura 15 para testar a interface de comunicação serial. Note que a USCI transmite e recebe o dado transmitido. Para este exercício, e para os próximos, iremos trabalhar à uma taxa (baudrate) de 1200 bps. Teste a interface enviando um byte qualquer e verifique que ele é recebido de volta. Verifique, através de breakpoints nas interrupções, ou por amostragem da flag de recepção (RXIFG), se o valor foi recebido corretamente no buffer de recepção.

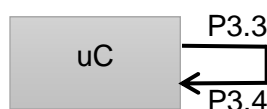


Figura 15 – USCI transmitindo pelo pino P3.3 e recebendo o dado transmitido pelo pino P3.4.



### Exercício 12:

Ainda com a mesma configuração do exercício anterior, envie e receba o seu nome pela interface serial, caractere por caractere. Use dois vetores:

- `char nome_tx[] = "Joaquim José"` → com o nome a ser transmitido e
- `char nome_rx[] = "xxxxxxxxxx"` → espaço para receber o nome

*Lembre-se de que as strings terminam com zero ('0'). Use este fato para finalizar tanto a transmissão quanto a recepção.*

### Exercício 13:

Utilize a interface serial para controlar os LEDs de outro microcontrolador. Ao pressionar o botão S1 o LED vermelho do outro uC deve alternar, já o botão S2 controla o LED verde. A ligação entre dois microcontroladores deve ser feita conforme a Figura 16.



Figura 16 – Conexão para comunicação entre dois MSPs.

O receptor responder aos seguintes códigos:

- 0x01 – Liga o LED **vermelho**
- 0x0A – **Apaga** o LED **vermelho**
- 0x10 – Liga o LED **verde**
- 0xA0 – **Apaga** o LED **verde**

O transmissor deve enviar apenas um código a cada pressionar de botão:

- S1 – Envia **0x01** e **0x0A** alternadamente
- S2 – Envia **0x10** e **0xA0** alternadamente

Exemplo: Ao pressionar uma vez o botão S1 o transmissor deve enviar a palavra **0x01**. Ao pressionar uma segunda vez, o transmissor envia **0x0A**.

### Exercício 14:

Iremos repetir o exercício anterior usando a técnica de *bit-banging*. Este exercício é importante pois nem todos os microcontroladores disponíveis no mercado possuem interfaces de comunicação serial dedicada. Você pode usar os mesmos pinos do exercício anterior, porém em modo GPIO. Mantenha a mesma taxa de transferência que os exercícios anteriores. A recepção dos dados deve ser feita no centro da janela (temporal) de cada bit, conforme a figura 13, pois assim se evita erros de recepção caso os dispositivos estejam ligeiramente fora de sincronia.

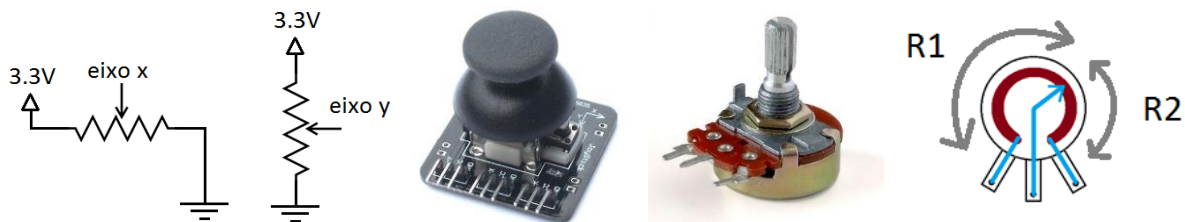
## Conversor A/D e DMA

Conversores A/D são dispositivos de interface entre o mundo analógico e o mundo digital. Eles permitem que um microcontrolador colete dados de transdutores que convertem grandezas físicas (como som, luz, temperatura, pH) no formato de tensão.

Neste roteiro, iremos estudar o princípio da conversão A/D. Para isso, utilizaremos divisores resistivos (na forma de potenciômetros) na entrada do conversor. Dessa forma, iremos realizar leituras de tensões proporcionais à posição do potenciômetro.

Um dispositivo simples que aplica esse princípio é o manche de um joystick. Esse dispositivo é tipicamente composto por dois potenciômetros, um para o eixo x e outro para o eixo y, como mostrado na figura 16. Os dois terminais de leitura da posição do eixo x e y são na verdade valores de tensão entre uma referência positiva (ex: 3,3V) e uma referência negativa (ex: 0V).

**Importante:** ao conectar o joystick à LaunchPad, o pino rotulado de +5V deve ser ligado ao pino 3,3V. NÃO O LIGUE AOS 5V. Um sinal acima de 3,3V pode queimar o pino do MSP430.



*Figura 17 – Joystick com dois potenciômetros e visualização interna de um pot.*

O MSP430 possui um conversor A/D integrado de 12 bits que funciona por aproximações sucessivas. Num primeiro momento, o sinal de entrada é amostrado e a tensão da entrada carrega um capacitor que guarda essa tensão. Numa segunda etapa, o conversor testa bit a bit, começando do MSB até o LSB para descobrir qual é o valor binário que melhor representa aquele nível de tensão. A Figura 18 mostra as etapas de conversão de um caso simplificado de apenas 3 bits.



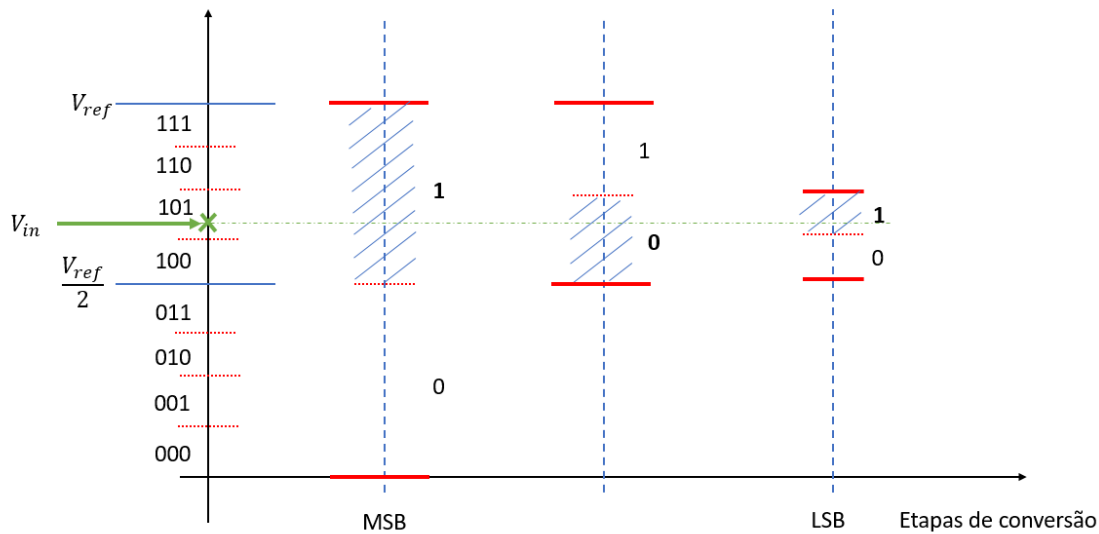


Figura 18 – Etapas de conversão de um conversor por aproximações sucessivas

No exemplo da Figura 18, a tensão de entrada  $V_{in}$  está acima da metade da referência. Assim, na primeira etapa de conversão, o conversor A/D atribui o valor “1” ao MSB. Em seguida, o conversor divide a escala da etapa anterior (região hachurada) em duas e verifica se a entrada se encontra na parte superior ou inferior. No exemplo a tensão se encontra na parte inferior da segunda etapa e então o segundo bit é “0”. Por fim, repete-se esse procedimento até que todos os bits sejam definidos. No exemplo, a palavra binária que representa a tensão  $V_{in}$  é **101**.

O MSP430 disponibiliza um conversor A/D de 12 bits, compartilhado entre 16 entradas, como mostrado na Figura 19. Somente uma entrada analógica por ser convertida a cada vez.

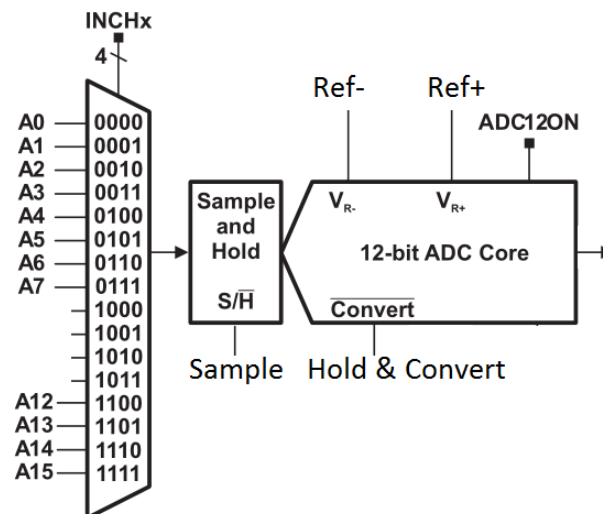


Figura 19 – Conversor AD do MSP430.

Para que a conversão seja feita em etapas sucessivas, é necessário que a entrada  $V_{in}$  esteja estável durante todo o processo de conversão. Isso é garantido através de um capacitor na entrada do conversor. Devido à natureza desse elemento, o capacitor leva um certo tempo para

acumular carga, então é necessária uma etapa de amostragem do sinal antes de iniciar a conversão. Esse processo é conhecido como *sample and hold*. A Figura 20 modela a entrada do conversor A/D. O resistor  $R_i$  representa a resistência de entrada do multiplexador e o capacitor  $C_i$  representa o bloco “Sample and Hold”.

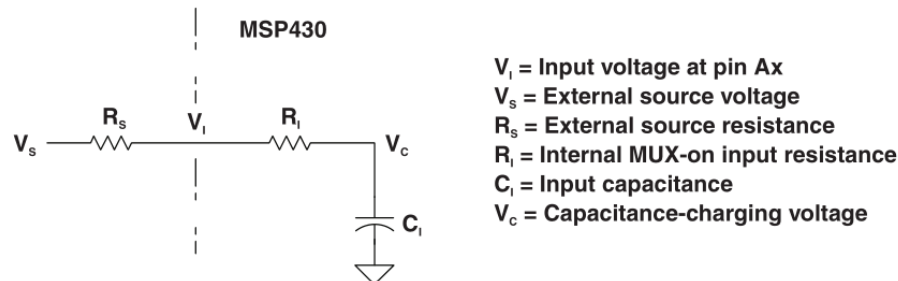


Figura 20 – Amostragem da entrada

A resistência de saída do dispositivo externo ( $R_s$ ) conectado à porta do conversor AD influencia o tempo de amostragem, ou seja, o tempo que leva para o capacitor da entrada ( $C_i$ ) carregar. Inicie o seu projeto calculando o tempo de carga do capacitor de entrada do conversor A/D seguindo a seguinte expressão:

$$t_{sample} > (R_s + R_i) \times \ln(2^{n+1})C_i + 800ns$$

Considere a resistência interna de  $1,8k\Omega$  (resistência do mux de entrada do conversor) e o capacitor interno de  $25pF$ . A resistência  $R_s$  deve ser o valor do potenciômetro no pior caso de medida, ou seja, em 50%. Para pequenas variações (quando alteramos a posição do potenciômetro) o MSP “enxerga” dois caminhos para a referência e no pior caso a resistência vista pelo MSP é dada por:

$$R_s = \frac{R_{pot}}{2} \parallel \frac{R_{pot}}{2} = \frac{R_{pot}}{4}$$

### Exercício 18:

Configure o módulo do conversor A/D para amostrar a tensão de entrada que representa apenas um eixo da posição do joystick (ou de um potenciômetro qualquer). Não se esqueça de ajustar os limites superior e inferior da tensão de referência para maximizar a precisão da leitura. Configure um timer para realizar essa amostragem a cada 10ms. Grave num vetor de memória os códigos dos últimos 5 segundos de amostragem. Você pode plotar o resultado da leitura no próprio CCS colocando o vetor na aba de expressões e clicando com o botão direito, selecione a opção “Graph”, como mostrado na Figura 21.

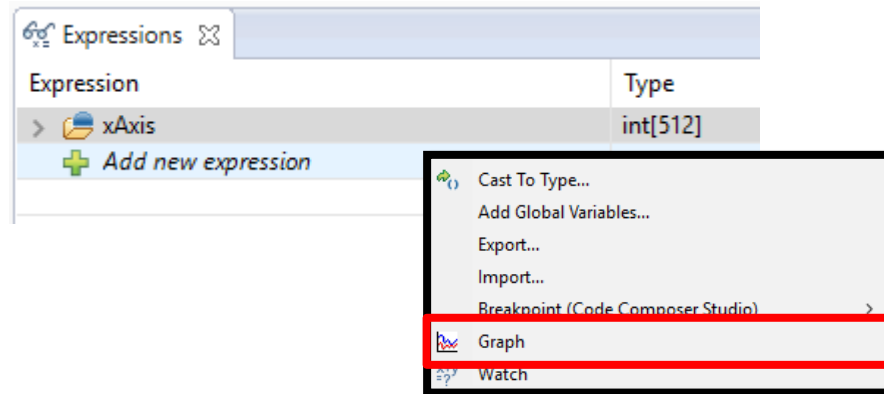


Figura 21 – Plotar um gráfico a partir de um vetor no CCS

**Exercício 19:**

Neste programa, iremos ler simultaneamente (ou quase) o valor da posição de dois eixos do potenciômetro. Use a informação da posição dos dois eixos para controlar a luminosidade dos dois LEDs da LaunchPad. Se a posição do joystick estiver no centro, os dois LEDs devem estar em 50% da luminosidade.

Use o eixo X para controlar a luminosidade do LED vermelho e o eixo Y para controlar a luminosidade do LED verde. Trabalhe com um sinal PWM a 100Hz com 256 passos (de 0 a 255). Ajuste a resolução do conversor A/D para 8-bits de forma a ficar na mesma resolução da PWM. Amostre a posição do potenciômetro a cada 20ms (50Hz). A figura 21 apresenta um diagrama do sistema proposto.

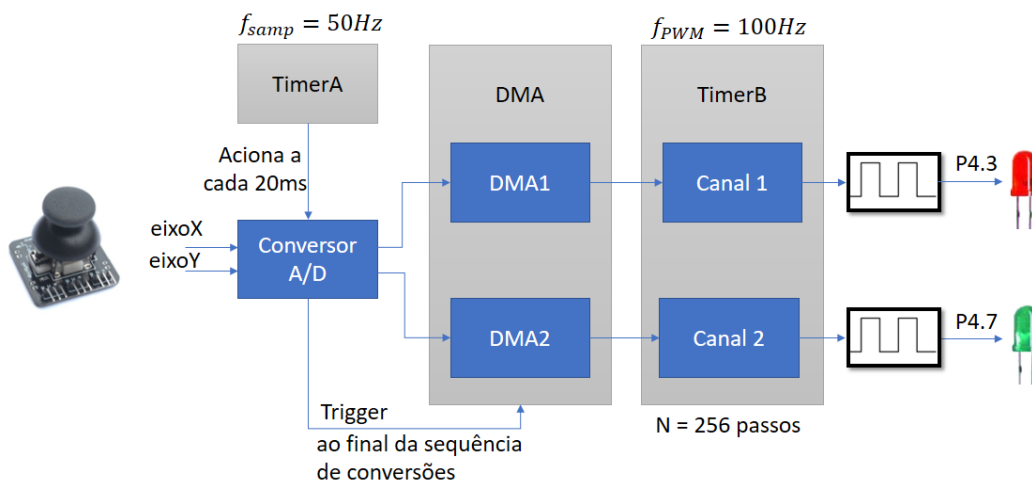


Figura 22 – Controle do brilho dos leds usando um joystick



Neste exercício, utilizaremos a saída dos canais 1 e 2 do timer B para controlar a luminosidade de ambos os LEDs. Para isso, use a porta P4 que permite remapear a funcionalidade dedicada de seus pinos como no exemplo abaixo.

```
// Configure Port Mapping
PMAPKEYID = 0x02D52;      // Get authorization to remap port function
P4MAP3     = PM_TB0CCR1A;  // Map P4.3 to output of timer B, channel 1
P4MAP7     = PM_TB0CCR2A;  // Map P4.7 to output of timer B, channel 2
```

Copiar regiões de memória, seja entre periféricos ou entre posições de memória, é uma tarefa trivial e bastante comum em programas de sistemas embarcados. O DMA (Direct Memory Access) é um periférico dedicado para realizar transferência de dados e permite que cópias sejam realizadas sem intervenção do processador. Neste exercício, utilize o DMA para mover o resultado da conversão para os canais de saída do timer B e conseqüentemente ajustar a largura do pulso de saída automaticamente (isso irá modificar a luminosidade do LED).