Estudo e Desenvolvimento de Sistemas Assíncronos e Síncronos

Cláudia Rodrigues Fernandes, Francisco Fragoso de Oliveira

January 10, 2020

Resumo: **Este trabalho prático divide-se em três partes. Na primeira parte, desenvolveu-se um sistema de comunicação assíncrono, utilizando um master e dois slaves. Utilizado dois botões, foi possível observar o controlo exercido pelo master sobre cada um dos slaves ligando o LED do slave correspondente. Na segunda parte, utilizando comunicação assíncrona e a interface JTAG, foram várias informações do microcontrolador e controlar o seu LED LD5. Por fim, desenvolveu-se um sistema de comunicação síncrona para fazer a medição da temperatura ambiente.**

1 Considerações gerais e desenvolvimento do código

Primeira Parte: Sistema de Comunicação Assíncrona

A comunicação USART (Universal Synchronous and Asynchronous Serial and Transmitter) é uma comunicação em série. Neste trabalho pretende-se utilizar o modo assíncromo USART.

Começamos por inicializar a comunicação assíncrona (ver função asynch9_init()), utilizando três registos de controlo e de estado (UCSR0X, USART Control and Status Register 0 X; X=A, B, C), UCSR0A, UCSR0B e UCSR0C. Para selecionar o modo assíncrono normal e ativar o modo multiprocessador usamos o registo UCSR0A. Para permitir a transmissão e recessão de 9 bits usamos o registo UCSR0A. Finalmente, para definir o formato do frame é utilizado o registo UCSR0C em conjunto com o registo UCSR0B, visto que com o registo UCSR0C são definidos oito bits e com o registo UCSR0B é definido o nono bit, (1<<UCSZ02). Ou seja, colocamos todas as flags relativas ao size com o valor lógico "1".

Para definir o modo assíncrono normal utilizamos também $BAUD = \frac{f_{osc}}{16(UBRR0+1)} \iff UBRRn = \frac{f_{osc}}{16BAUD-1}$

onde BAUD é o baud rate (bits por segundo), f_{osc} é a frequência do relógio do oscilador do sistema e UBRRn (USART Baud Rate Register) contém os registos UBRRnH e UBRRnL. Denotar que o registo UBRRnH contém os quatro bits mais significativos, enquanto que o UBRRnL contém os oito bits menos significativos do USART baud rate.

Em seguida foram construidas a funções para envio do address e de dados (ver funções send_addr(addr) e send_data(data)), onde o address e os dados ficam guardados (cada um) no UDR0 (USART Data Register). Depois foi construida a função get_data() para receber os dados no modo multiprocessador.

Para configurar os address dos dois slaves foram utilizados quatro pinos do arduino tomado como master. Quando carregamos num dos botões, é feita uma leitura diferente do address e mudamos o ligamos o LED do slave correspondente.

É impotante referir que a utilização do modo multiprocessador requer a existência de um nono bit. Se o 9º bit é 1 estamos perante um address frame e se for 0 é um data frame. Quando o modo multiprocessador está ligado, os slaves ignoram os data frames e recebem os address frames. Finalmente, quando os slaves recebem o seu address, desliga-se o MPCM e é possível receber dados.

Segunda Parte: Controlo remoto do ChipKIT Uno32 usando a sua porta JTAG

O chipKIT UNO32 inclui o microcontrolador PIC32MX320F128H. Ver descrição das funcionalidades deste dispositivo na secção 2.

Começamos por definir uma função send (valor), tendo em conta o TAP State Diagram presente no programa goJTAG. Esta função irá ajudar a alterar o modo do registo de instrução e depois a enviar dados via EXTEST.

Pretendemos pressionar

i. "d" para obter o ID Code;
ii. "1" para ligar LED LD5;
iii. "0" para desligar LED LD5;
iv. "b" para obter estado do botão;

i. "d" para obter o ID Code

Com recurso à BSDL file do PIC32MX320F128H tiramos que para obter o ID CODE temos de enviar a instrução (00001). Após enviar a instrução é possível ler os 32 bits correspondentes ao ID CODE do dispositivo.

ii. "1" para ligar LED LD5

Enviamos a instrução (00110), correspondente ao modo EXTEST. Nesta parte não conseguimos descobrir qual o número das células de boundary scan para ligar o LED LD5, portanto após enviar esta instrução, colocamos sempre "1" no TDO.

iii. "0" para desligar LED LD5

Enviamos a instrução (00110), correspondente ao modo EXTEST. Nesta parte colocamos sempre "0" no TDO.

iii. "b" para obter estado do botão

Tal como nos dois casos anteriores, enviamos novamente a instrução (00110), correspondente ao modo EXTEST. Determinamos que o botão ligado ao 29 do chipkit corresponde a 3, com o auxílio do ficheiro BSDL.

Nota: Se dermos 5 impulsos de relógio com TMS(1), voltamos ao estado Test-Logic-Reset.

Terceira Parte: Medição da Temperatura utilizando comunicação I2C

Nesta parte estabeleceu-se uma ligação assíncrona entre o arduíno e o computador e em seguida uma ligação síncrona, I2C entre o arduino e o sensor de temperatura MCP9808. Para estabelecer uma comunicação I2C entre o arduino e o sensor utilizamos a 2-wire Serial Interface do ATmega328P (ver capítulo 21 ATmega328P). Utilizando este protocolo é possível projetar um sistema com até 128 dispositivos diferentes interconectados. São utilizados 2 linhas bus biderecionais, uma para o o clock (SCL) e outra para os dados (SDA).

Começamos por estabelecer uma comunicação assincrona utilizando USART, tal como na primeira parte do trabalho, para ligar o ATmega328P ao computador. Em seguida criamos uma função para inicializar o I2C. Nesta parte colocamos o valor lógico "1" apenas na flag correspondente à ativação da interface 2-wire, TWEN, no TWCR (Two Wire Control Register), todas as restantes são zero.

Quando pretendemos que o master (ATmega328P) envie o address para o slave (MCP9808), enviamos uma condição para começar a comunicação, sendo que esta só termina com a condição de que o TWCR não deve ter a flag do interrupt, TWINT, com o valor lógico "1". De modo semelhante, são criadas funções para parar a comunicação I2C (I2C_stop()), uma para escrita (I2C_write()) e para leitura dos dados presentes no sensor de temperatura, uma com ACK e outra com NACK. Finalmente criamos uma função para fazer a configuração do registo de configurações do sensor, para definir como queremos receber os dados.

De notar que é de extrema importância a flag correspondente ao enable bit, TWEN, estar com o valor lógico "1" sempre ao longo do programa.

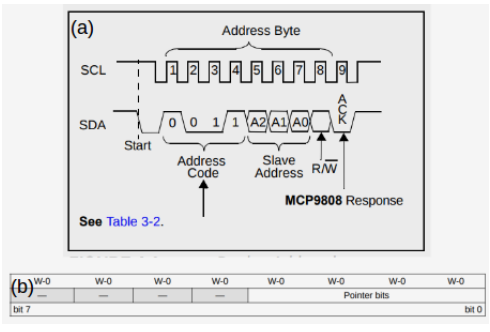

Figure 1: **(a) Address do sensor MCP9808**, definido como 0x30 para escrita e 0x31 para fazer a leitura; **(b) Descrição do pointer ou address de cada registo**, para aceder ao registo de configurações usou-se o address 0x01, e para aceder ao registo de temperatura T_A usou-se 0x05.

Para aceder ao registo de configurações e ler ou escrever o address consultamos a datasheet do sensor, conforme se observa na figura anterior.

Após fazer as configurações do CONFIG register, criamos uma função para ler a informação proveniente do registo de temperatura T_A.Para isso começamos por enviar o address 0x30 para escrita, depois enviamos o address do resgisto de temperatura T_A e enviamos a condição de paragem. A seguir enviamos o address para fazer a leitura e obtemos dois bytes, um com acknowledge (ACK) e outro com not-Acknowledge (NAK).

Após fazer a conversão no setup, é possível observar os valores de temperatura com $\triangle t = 2.5\ s$.

2 Descrição do Setup Experimental

Primeira Parte: Sistema de Comunicação Assíncrona

Para desenvolver o sistema de comunicação assíncrona foram utilizados um master e dois slaves com o mesmo hardware; um microcontrolador ATmega328P e MAX485 transceivers.

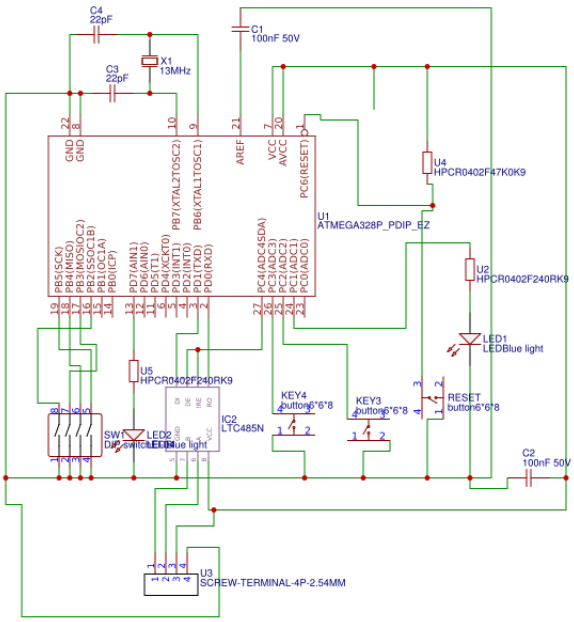

Figure 2: **Template EasyEDA:** Representação do circuito necessário para executar o programa. De notar que o programa não está escrito exatamente para este circuito, uma vez que o DIP switch utilizado não está ligado aos pinos que definem o address no programa.

Segunda Parte: Controlo remoto do ChipKIT Uno32 usando a sua porta JTAG

O chipKIT UNO32 Além de incluir o microcontrolador PIC32MX320F128H, o chipKIT UNO32 possui muitas compatibilidades com o arduino. Possui 42 I/O pinos, dois user LEDS e opera a 3.3V. Desses dois user LEDS, pretendemos ligar o LED LD5.

Terceira Parte: Medição da Temperatura utilizando comunicação I2C

Para esta parte fez-se a montagem esquematizada abaixo.

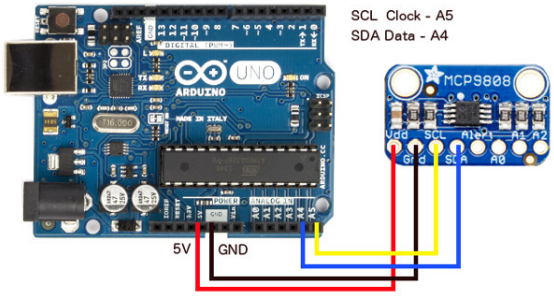

Figure 3: **Montagem experimental:** Ligações utilizadas para estabelecer a comunicação I2C.

A seguir está representado o template EasyEDA.

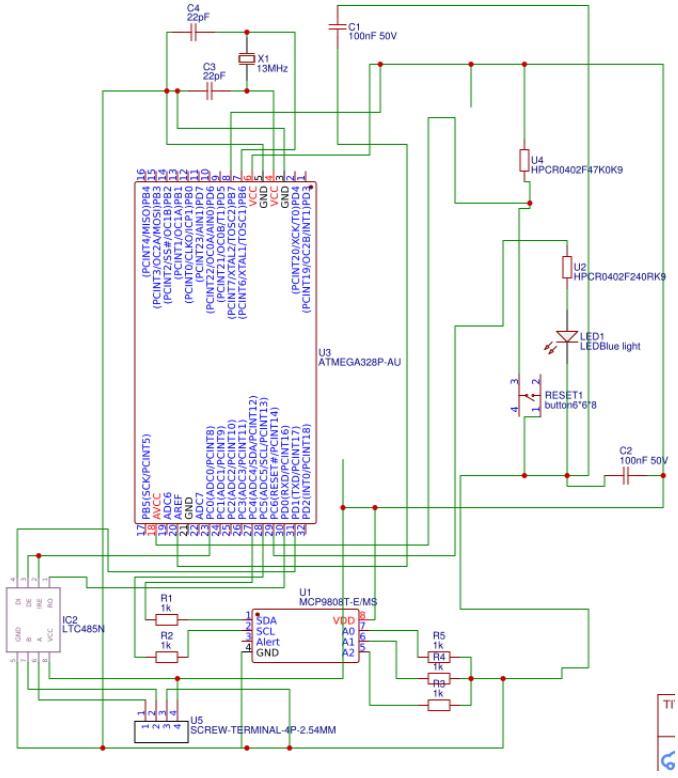

Figure 4: **Template EasyEDA:** Representação do circuito necessário para executar o programa. De notar que o programa não está escrito exatamente para este circuito, uma vez que o DIP switch utilizado não está ligado aos pinos que definem o address no programa.

3 Resultados

Primeira Parte: Sistema de Comunicação Assíncrona

Ao escrever o código cometemos um erro ao definir a frequência de oscilação como $f_{osc} = 1843200$. Apesar de o objetivo ser obter um baud rate de 9600, tal não foi possível devido a este erro. Foi obtido um baud rate superior ao previsto. Contudo, apesar desta diferença, foi possível efetuar comunicação e executar comandos porque o hardware do master e dos slaves é igual. A f_{osc} correta é $f_{osc} = 16000000$.

Segunda Parte: Controlo remoto do ChipKIT Uno32 usando a sua porta JTAG

Conseguimos executar todos os comados. Contudo, quando pressionamos a tecla "1" para ligar o LED LD5, não só ligamos esse LED, como também ligamos o outro user LED, LED LD4.

Terceira Parte: Medição da Temperatura utilizando comunicação I2C

Apesar de algumas dificuldades a estabelecer a comunicação, conseguimos criar um sistema de comunicação I2C para, com um sensor, medir a temperatura ambiente.

As principais dificuldades deveram-se ao facto de em algumas funções não termos colocado no TWCR a flag do enable bit com o valor lógico "1". Ora, o TWEN bit permite a operação TWI e ativa a interface TWI. Em muitas das nossas funções colocamos essa flag a "0" (por lapso) e com isto terminam todas as transmissões TWI.