Sistemas Eletrónicos - Projeto Final

Beatriz Garrido (up201504710)

Francisco Lobo Àvila (up201506637)

Abstract - Este projeto foi realizado no âmbito da unidade curricular de Sistemas Eletrónicos. Este é composto por duas tarefas: o controlo de um microcontrolador ChipKIT Uno32 usando unicamente a sua porta JTAG; e o desenvolvimento de um projeto à escolha, tendo por base uma comunicação I2C. Previamente a este trabalho, também foi elaborado um outro projeto, cujo propósito era auxiliar a realização deste, tendo como maior diferença a comunicação assíncrona do microcontrolador

I. Introdução

A indústria de sistemas eletrônicos atualmente é extremamente competitiva e requer cada vez mais níveis de elevada qualidade e fiabilidade, a um baixo custo. Dito isto, a fase de testes no processo de desenvolvimento é crucial para assegurar o desejo do cliente.

Para solucionar este problema, o "Joint Test Action Group" (JTAG) desenvolveu métodos para verificar o bom funcionamento de circuitos integrados após a sua impressão. Daí nasceu o uso de *Boun*dary *Scan*, que permite o acesso aos sinais lógicos dos circuitos integrados da maior complexidade. É com base neste método que consiste a primeira tarefa deste projeto: testar o correto funcionamento de um microcontrolador ChipKIT-32, a partir da sua porta MCU JTAG.

Atualmente a comunicação entre sistemas embebidos é algo que tem um papel importante no design dos mesmos, assim é crucial o uso de um protocolo consistente e fidedigno. A comunicação I2C é um dos protocolos de comunicação mais usado nesta área, devido à sua rapidez e flexibilidade. Como tal, este protocolo foi escolhido para realizar a segunda tarefa deste trabalho, juntamente com o sensor TSL2561, um sensor de luminosidade com uma grande precisão nos seus resultados.

II. Task 1 - JTAG

O objetivo deste trabalho é controlar um microcontrolador PIC32, através da sua porta JTAG. Este controlo é feito através de comandos inseridos no computador pelo utilizador e dos sinais TMS, TCK e TDO que estão ligados aos pinos do Arduino.

A. lógica do programa

O fundamento deste programa consiste no uso na máquina de estados que controla o JTAG:

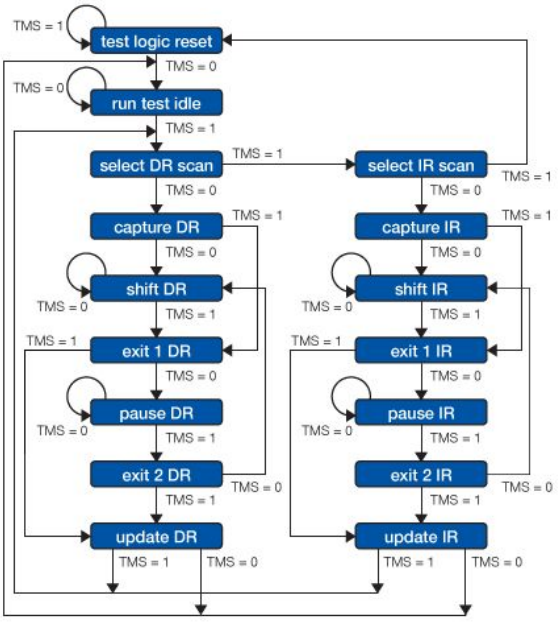

Fig. 1 TAP state machine

Esta tem como entradas os sinais TMS e TDI que, juntamente com um sinal de clock (TCK), fazem a transição de estado. Dependendo do estado em que se encontra, serão realizadas instruções que irão escrever o valor correto em TDI.

É de realçar que o sinais TDI e TDO do arduino estão ligados aos TDO e TDI do ChipKit, respetivamente. O valor de TCK por sua vez, representa o clock do PIC32

B. Funções realizadas

O código desta tarefa dá uso a 3 instruções:

- ***SAMPLE/PRELOAD:*** de modo a poder aceder aos valores internos do circuito integrado
- ***EXTEST:*** de modo a poder interferir com o boundary scan sem mudar variáveis internas do circuito integrado
- ***IDCODE:*** coloca no Data Register o valor do ID CODE

Dependendo do estado onde se encontra a máquina de estados, uma destas instruções poderá ser usada para alcançar o objetivo final do projeto. Para isso, foram criadas as seguintes funções no código:

- *void TAPclk(int value)* - Dependendo do valor de entrada, envia combinações diferentes de TMS e TDO (TDI do JTAG a ser testado). Juntamente com um sinal de clock, avançando para o estado seguinte da TAP state machine. Esta função permite também colocar bits no registo de dados e de Instruction Register, se a state machine estiver no estado **shift DR** ou **shift IR,** respetivamente.
- *uint32_t read_TDI(int n)* - Utilizando a função *TAPclk()*, coloca a zero o valor do data register, de modo a obter no TDI (do arduino) o valor que se pretende ler. Uma vez que o primeiro bit recebido é o LSB, é necessário inverter os valores obtidos.
- *uint32_t read_TDI_b()* - Esta função é inspirada na anterior, com a diferença de que lê apenas 1 bit. É utilizada para ler o estado do botão.
- *void test_reset()* - Coloca a 1 o TMS, durante 5 ciclos de relógio consequentes. Isto para colocar a TAP state machine no estado test reset.
- *void setup()* - Aqui são definidoes os inputs e output do arduino,os registos e as instruções. Bem como é colocada em **test reset** a máquina de estados.
- *void shift_IR()* - A state machine transita para o estado **shift IR**, a partir do estado test reset, **update IR** ou **update DR**
- *uint32_t read_IDCODE()* - Usada para colocar o Instruction Register a 0001 (instrução correspondente ao IDCODE). No estado **shift DR,** é feita a leitura do IDCODE
- *void shift_DR()* - A state machine transita para o estado **shift DR**, a partir do estado **test reset**, **update IR** ou **update DR**
- *void print_button_state()*- Usada para colocar o Instruction Register a 0010 (instrução correspondente ao SAMPLE/PRELOAD). No data register, lê o 4º bit que corresponde ao estado do botão, segundo o BSDL.
- *void write_led1()* - Usada para colocar o Instruction Register a 0110 (instrução correspondente ao EXTEST). No Data Register, as posições 19, 20 e 129 são colocadas a 1 para que o output do led seja definido como ligado. A posições 129 tem *este valor para evitar ativar o reset*
- *void write_led0()* - Semelhante à função write_led1() mas o valor do led é definido como 0, ou seja, desligado
- *void loop()* - Dependendo dos comandos inseridos pelo utilizados, o código faz o seguinte:
    - **d ->** imprime o IDCODE do PIC32 em binário e em hexadecimal
    - **b ->** imprime o estado do botão conectado ao pino 29 do ChipKIT.
    - **1 ->** liga o LED D5 da placa ChipKIT
    - **0 ->** desliga o LED D5 da placa ChipKIT

É também de realçar que sempre que este pedaço de código está presente nas funções:

```
TAPclk(TMS0);
TAPclk(TMS0);
TAPclk(0);
```

faz com que o estado final da máquina de estados após qualquer função, seja sempre **run test idle**

III. Task 2 - Comunicação síncrona

A.

B. Componentes auxiliares

O objetivo pessoal desta tarefa do projeto será controlar o movimento de um pequeno carro, previamente montado, a partir de um raio luminoso (a partir de uma lanterna ou laser). Este é composto por 2 motores DC que controlam as 2 rodas do carro, juntamente com um Driver L298N.

É de salientar que a montagem deste pequeno projeto foi realizada numa outra unidade curricular, pelo que não foram aqui especificadas as suas ligações ou a sua integração, visto que este

deve ser tratado como um todo ou seja, um componente à parte.

Para além do microcontrolador Arduino e do sensor TSL2561, foi também usado um sensor LDR que tem como objetivo auxiliar a direção das rodas face à direção do feixe luminoso incidido. Para fazer a leitura do LDR, foi utilizada uma porta analógica do Arduino

C. Hardware

As ligações dos componentes desta tarefa não foram da maior complexidade, visto que eram poucas e simples. Apenas foi necessário ligar os pins SDA e SCL entre o microcontrolador e o sensor fornecido. Para além disso, também foram feitas as ligações entre o LDR, uma resistência de 5.6kΩ e os respetivos VCC e GND para cada componente. Todas estas ligações físicas podem ser observadas na figura seguinte:

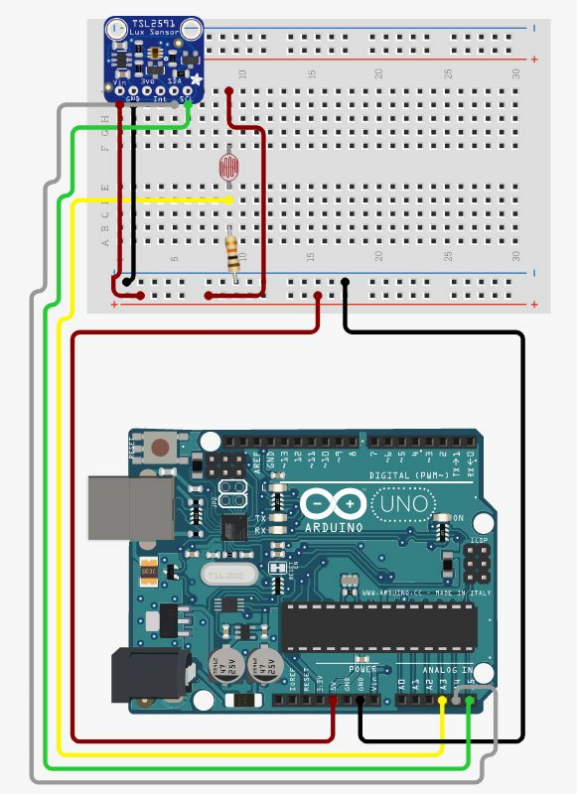

Fig 2. Esquemático das ligações dos componentes

O PCB que representa a *shield* desenvolvida foi criado usando *EasyEDA*. O design deste hardware foi criado de modo a economizar a ocupação da área.

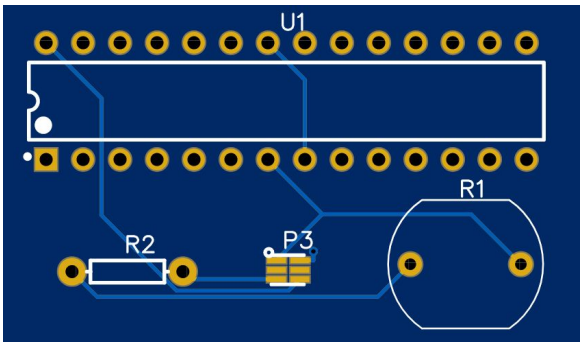

Fig. 3 PCB - comunicação síncrona

IV. Comunicação assíncrona

Previamente a este projeto foi também realizado um outro trabalho cujo objetivo era alcançar uma ligação assíncrona entre um master e dois slaves, baseada num microcontrolador ATmega328P e em transceptores MAX485.

Foi também pedido que se desenvolvesse um esquemático do hardware utilizado para teste. Esta foi a primeira interação com a ferramenta *EasyEDA*, para desenvolver o esquemático e o PCB

Para o desenvolvimento do código, foi aconselhado o uso do 9º bit para a distinção entre o envio de endereços e dados.

A. Software desenvolvido

Para a realização deste projeto, foram criadas as seguintes funções:

- *void asynch9_init(long BAUD):* para configurar o *BAUD rate*
- *void send_addr(uint8_t addr):* coloca o 9º bit da mensagem a 1, sinalizando que será enviado um endereço. Envia o valor do endereço para o slave desejado
- *void send_data(uint8_t data) :* coloca o 9º bit da mensagem a 0, sinalizando que serão enviados dados. Envia o valor dos dados para o slave desejado
- *uint8_t get_data():* investiga o valor do 9º bit para saber se estará a lidar com dados ou endereços:
    - se se tratar de um endereço diferente do do próprio MCU, liga MPCM0, senão desliga MPCM0
    - se se tratar de dados, retorna o valor dos mesmos
- caso o 9º bit indicar um endereço mas o slave não estiver em *receiving mode*, esta função ignora o valor

- *void setup():* configura os pins ligados aos butões 1 e 2. Configura as entradas e liga as resistências de pull up aos pinos que escolhem o endereço. Configura o LED e os enables de write/read como saídas

As resistências de pull up são usadas para assegurar que o valor de *standby* é 1

Na função *loop()* são realizadas as instruções que o master e o slave devem realizar.

Para o caso do **slave**: liga o LED caso sejam enviados os dados referentes ao mesmo, caso contrário desliga o LED e termina o seu *receiving mode*

Para o caso do **master** aplica-se a seguinte máquina de estados:

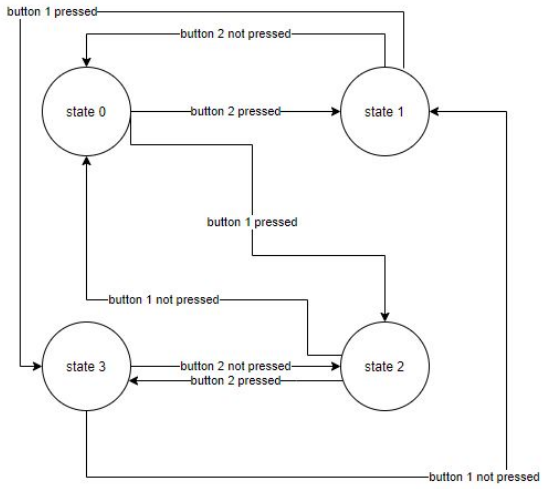

Fig. 4 state machine de master

Onde:

- **state 0:** ambos os LEDs estão desligados
- **state 1:** LED do *Slave2* ligado, LED do *Slave1* desligado
- **state 2:** LED do *Slave1* ligado, LED do Slave2 desligado
- **state 3:** ambos os LEDs estão ligados

B. Hardware desenvolvido

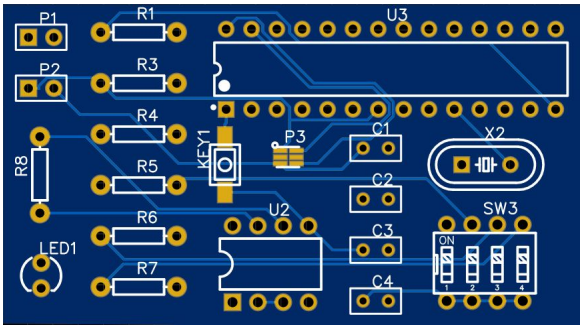

Fig. 5 PCB - comunicação assíncrona

V. Conclusões

As tarefas necessárias para realizar este trabalho foram realizadas com sucesso. Todos os componentes realizaram as tarefas necessárias, com um tempo de resposta aceitável.

Podemos concluir que melhoramos os nosso conhecimentos na área de testes de circuitos integrados, nomeadamente JTAG, bem como na agilidade em trabalhar com registos dos microcontroladores.