Identificador de CI’s

Eletrônica Embarcada (120871) - UnB/FGa

Rafael Feijó Leonardo
Brasília, Brasil
goldcard99@hotmail.com

Bismark Cotrim Teixeira
Brasília, Brasil
bismarkcotrim@hotmail.com

Resumo — A identificação de circuitos integrados em um laboratório de eletrônica pode se tornar uma tarefa custosa quando se trata de chips mais antigos. Isso porque, ocasionalmente, eles podem perder seu código. Nesse sentido, este trabalho detalha o desenvolvimento de um circuito microcontrolado para identificação de CI’s TTL, da família 74.

Palavras Chave — Circuito Integrado; Microcontrolador; Lógica Combinacional; Identificação; Hardware; Software;

Introdução

Para o profissional, ou mesmo para o hobbista em eletrônica, é comum ter posse de vários tipos de CIs e diversos componentes organizados e guardados pela bancada de trabalho. Entretanto, muitas vezes a identificação de um CI pode se tornar um desafio, seja devido ao desgaste natural de sua marcação ou simplesmente pelo fato de não conseguir acesso ao datasheet correto do componente. Nesse sentido, o Identificador de Circuitos Integrados proposto visa facilitar este processo, dando informações suficientes para sua identificação.

O modelo de produto visado é baseado no vídeo [1]*An IC-tester from Ebay*, onde em uma estrutura totalmente independente e compacta, os CI’s são posicionados e testados por um microprocessador que, por fim, apresenta a identificação por meio de uma interface gráfica, após alguns poucos segundos em modo teste.

Hoje no mercado, existem diversas opções de Identificadores de CI’s, com os mais variados valores - desde algumas dezenas de reais à alguns milhares de reais, e isso deve-se ao fato da tecnologia empregada para identificação. As principais diferenças entre eles são a capacidade de processamento e a variedade de testes para identificação dos *chips*.

Neste trabalho, o circuito deverá realizar testes para identificação de CI's TTL, da família 74, tal que o resultado será apresentado em um display OLED contendo o modelo do componente (ex: '74HC08') e sua lógica (ex: 'NOT', 'AND', 'OR').

Seu diferencial, dentre os produtos já existentes no mercado, é o uso de um microprocessador de baixo custo, o *MSP430G2ET*, que é capaz de atender todos os requisitos de processamento e manter o custo do projeto abaixo da média de mercado.

I. Desenvolvimento

A. A Solução

Visando maior praticidade na identificação de portas lógicas e, ainda, a possibilidade de testar sua integridade, a solução foi projetada para funcionar em uma estrutura móvel e independente, capaz de realizar testes por combinações lógicas em alta performance, comparando os resultados obtidos na saída do componente com as respostas cadastradas em *truth tables,* contidas na memória *FLASH* do microcontrolador.

B. Descrição do Hardware

O *hardware* foi projetado de forma simplista, à fim de atender os requisitos do projeto e manter o custo abaixo do apresentado no mercado.

Os requisitos levantados para esta parte, bem como suas justificativas, foram:

- *Push Button* - Inicialização do sistema (*On/Off*);
- Soquete ZIF (18 pinos) - Acoplamento dos CI’s ao sistema;
- Display OLED 128x64 - Interface gráfica;
- *MSP430G2ET* - Microcontrolador para realização da lógica combinacional;
- Bateria 9V - alimentação do circuito;

tal que seu funcionamento é descrito pelo diagrama esquemático apresentado à seguir.

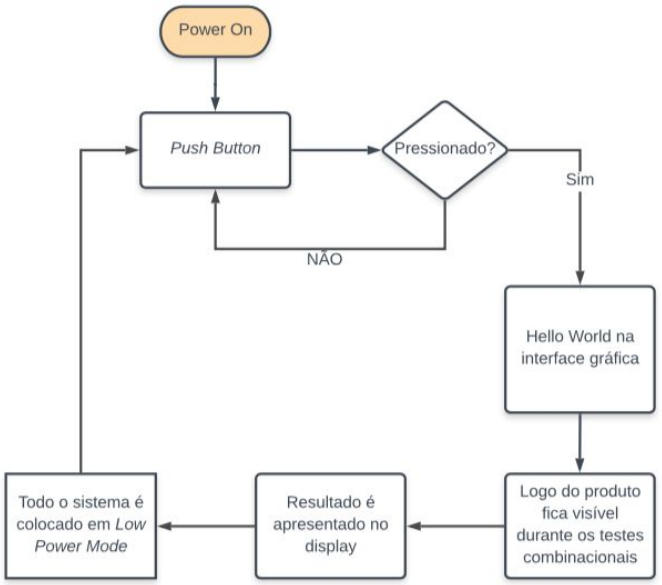

Fig. 1. Diagrama esquemático do *Hardware* projetado para o Identificador de CI's.

Ou seja, ao alimentar o circuito, o microcontrolador espera até que o botão seja pressionado para começar os testes combinacionais, de forma a funcionar em baixo consumo, com corrente de, aproximadamente, 0.5uA. Ao pressionar o botão, é apresentado na interface gráfica a logo do produto e, após a finalização dos testes, o resultado obtido.

Dessa forma, a *BOM* (*Bill of Materials*) obtida é apresentada a seguir.

Componente	Identificação	Qnt	Preço (R$)
Microcontrolador	MSP430G2ET	1	12.99
Interface Gráfica	OLED 0.96" 128x64	1	10.30
Soquete	ZIF (18 pinos)	1	4.99
Resistor	4k7Ω, 1/4W	4	0.04
Placa de Fenolite	Face Simples, 7x9cm	1	2.99
Chave Táctil	4t, 6x6x5cm	2	0.12
TOTAL	--		31,67

Fig. 2. Tabela contendo a lista de materiais do projeto (BOM).

C. *Descrição do Software*

Para o desenvolvimento do *firmware*, foram criadas duas frentes, sendo elas a lógica para identificação dos CI's e a interface gráfica.

I. *Lógica para Identificação dos CI's*

O sistema de análise combinacional é composto por um vetor e duas matrizes, no qual o vetor contém o mapeamento dos pinos que serão utilizados durante os testes, uma matriz contém a configuração dos GPIOs e outra matriz armazena os resultados obtidos em cada rodada de testes.

Os testes consistem na troca ordenada dos níveis lógicos das saídas do microcontrolador. O primeiro teste realizado é para a lógica *NOT*, em que primeiramente é aplicado nível lógico baixo e, posteriormente, nível lógico alto. Caso não bate com a *truth table* cadastrada, o sistema passa para o teste de CI's de duas entradas, passando pelas portas *AND*, *OR*, *XOR*, *NAND*, *NOR*, *XNOR*.

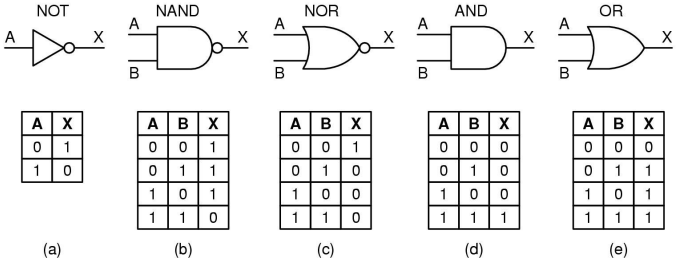

Fig. 3. (a) Tabela verdade lógica *NOT*. (b) Tabela verdade lógica *NAND*. (c) Tabela verdade lógica *NOR*. (d) Tabela verdade lógica *AND*. (e) Tabela verdade lógica *OR*.

Ao passo em que o *software* identifique respostas semelhantes às cadastradas, o *loop* de iterações é interrompido com um *"break"* e o contador de iterações é enviado a função "*printResult()*", para apresentar o resultado na interface gráfica.

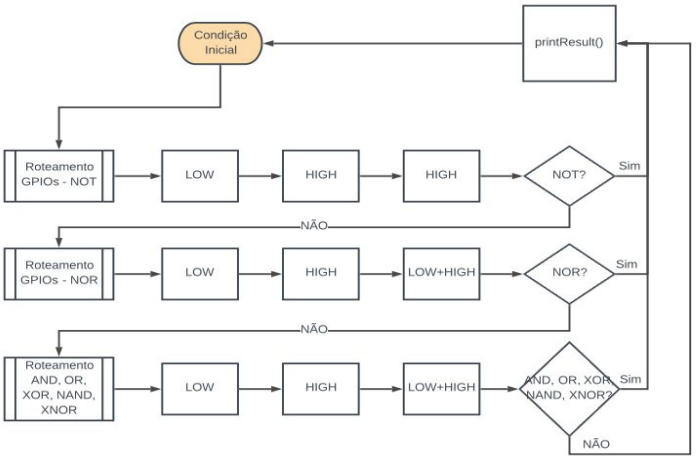

Fig. 4. Diagrama esquemático da lógica de testes desenvolvida no *firmware*.

II. Interface Gráfica

Para a interface gráfica, foi criada uma biblioteca para configuração e utilização do I2C BUS para o display OLED 128x64 bits, contendo os arquivos "*Library.h*", que contém as matrizes que geram as letras e a logo do display, "*OLED_128x64.h*" e "*OLED_128x64.cpp*", que contém as funções principais da biblioteca, e "*SSD1306.h*" e *"SSD1306.cpp"* que contém o *driver* para o OLED.

Para seu uso, foi incluído o *header* "OLED_128x64.h" e, em seguida, criado a instância "oled".

Inicialmente deve-se chamar a função "begin()", que configura o barramento I2C, conforme apresentado no fragmento de código à seguir.

```
// Configure I2C BUS
void I2C_begin(void)
{
  UCB0CTL1 |= UCSWRST;                      // Enable SW reset
  UCB0CTL0  = UCMST + UCMODE_3 + UCSYNC;  // I2C Master, synchronous mode

  UCB0CTL1 = UCSSEL_2 + UCSWRST;          // Use SMCLK, keep SW reset
  UCB0BR0  = 10;                          // fSCL = SMCLK/10 = 100kHz

  UCB0BR1 = 0;

  P1SEL |= BIT6 + BIT7;                   // Assign I2C pins to USCI_B0
  P1SEL2|= BIT6 + BIT7;                   // Assign I2C pins to USCI_B0

  UCB0CTL1 &= ~UCSWRST;                   // Clear SW reset, resume operation
  //IE2 |= UCB0RXIE + UCB0TXIE;           //Enable RX and TX interrupt
}
```

Fig. 5. Printscreen do trecho de código que contém a função *I2C_begin()*.

Na biblioteca criada, as funções *beginTransmission()*, *write()* e *endTransmission()* foram recriadas para o CCS e podem ser observados nas figuras 6, 7 e 8.

```
// Begin Transmission to slave via I2C BUS
void I2C_beginTransmission(volatile int slave_address)
{
  while(UCB0CTL1 & UCTXSTP)               // Ensure stop condition was sent
      ;

  UCB0I2CSA = slave_address;

  UCB0CTL1 |= UCTR;
  UCB0CTL1 |= UCTXSTT;                    // I2C TX, start condition
}
```

Fig. 6. Printscreen do trecho de código que contém a função *I2C_beginTransmission()*.

```
// Begin Transmission to slave via I2C BUS
void I2C_write(unsigned char data, unsigned int txControl)
{
  UCB0TXBUF = data;

  if (txControl)
  {
    while(UCB0CTL1 & UCTXSTT)
      ;
  }
  else
  {
    while((IFG2 & UCB0TXIFG) == 0)
      ;
  }
}
```

Fig. 7. Printscreen do trecho de código que contém a função *I2C_write()*.

```
// End Transmission on I2C BUS
void I2C_endTransmission(void)
{
  UCB0CTL1 |= UCTXSTP;

  while(UCB0CTL1 & UCTXSTP)
    ;
}
```

Fig. 8. Printscreen do trecho de código que contém a função *I2C_endTransmission()*.

A função "printLogo()" permite desenhar a logo do produto no display, em preto e branco, antes de apresentar o resultado obtido.

Fig. 9. Logo do produto.

Por fim, a função "printResult()" permite escrever, à partir de um *switch case*, todas as especificações dos CIs cadastrados - NOT, AND, OR, XOR, XNOR, NAND e NOR.

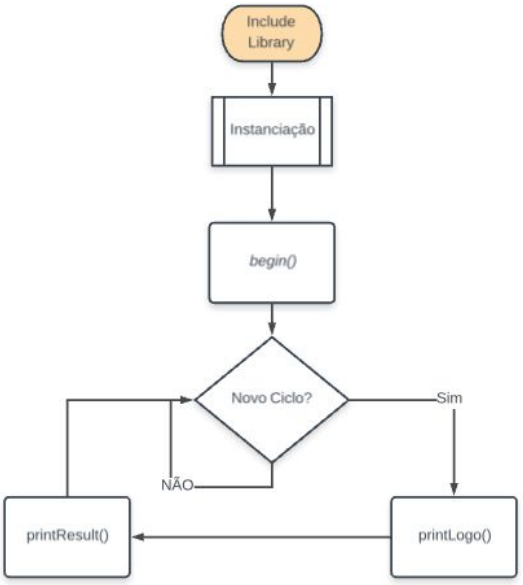

Fig. 10. Diagrama esquemático da Interface Gráfica.

II. Resultados

Inicialmente, o código foi desenvolvido na IDE Energia V1.8.7, tal que foi utilizado o *Serial Console* para realizar o *debug* da lógica combinacional e calibrar o *firmware* até conseguir um acerto de 100% dos CI's testados *NOT*, *AND* e *OR*.

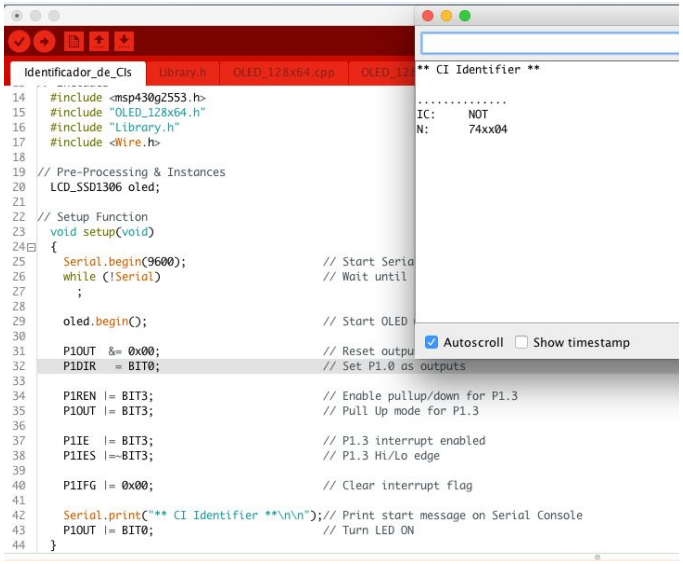

Fig. 11. *Printscreen* do resultado obtido via *Serial Console,* na IDE Energia, para um teste de uma porta NOT - 74HC04.

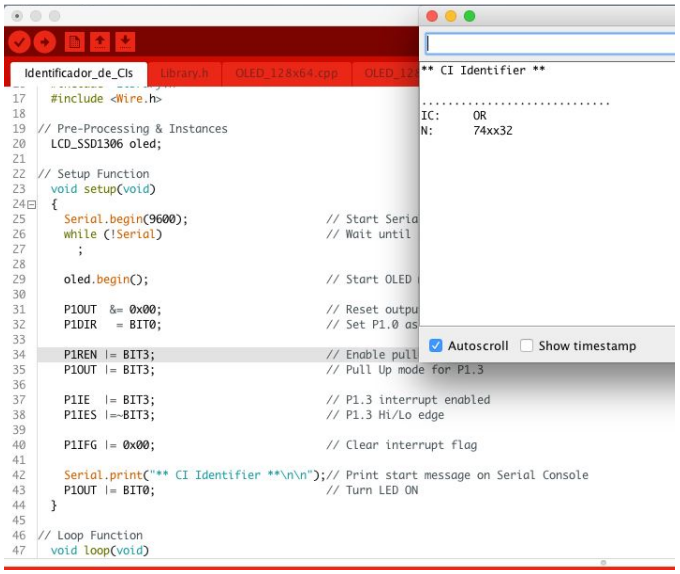

Fig. 12. *Printscreen* do resultado obtido via *Serial Console*, na IDE Energia, para um teste de uma porta OR - 74HC32

Após obter sucesso na identificação dos circuitos integrados mais básicos, a lógica foi implementada para identificação de CI's *XOR*, *NAND* e *XNOR*, que possuem mesmo roteamento de portas *GPIOs*. Nessa etapa, também foi configurado a primeira versão do display *OLED*, baseado na biblioteca *Wire.h*, para apresentar números e letras.

Na terceira etapa, a lógica de testes foi expandida para portas NOR e implementada, finalmente, no *Code Composer Studio V9.0*. Em relação a Interface Gráfica, foi necessário recriar a biblioteca *Wire.h*, de forma simplificada, tal que a maior dificuldade foi a tradução e obtenção de trechos desta biblioteca que faziam referências a outras bibliotecas ou a partes sem documentação precisa.

Por fim, o código foi otimizado para o *CCS* e, em teste, foi capaz de identificar todos os CI's acoplados, validando seu funcionamento.

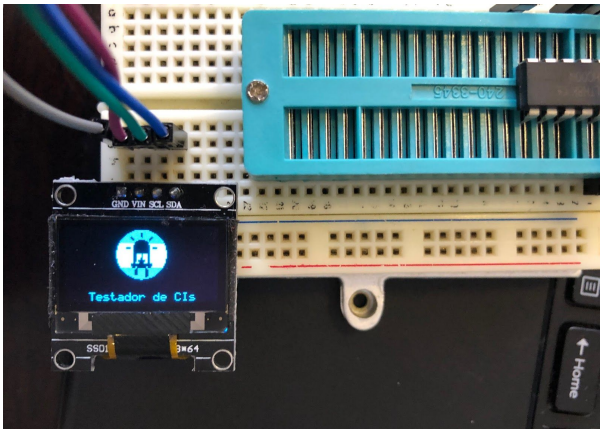

Fig. 13. *Foto da inicialização da Interface Gráfica.*

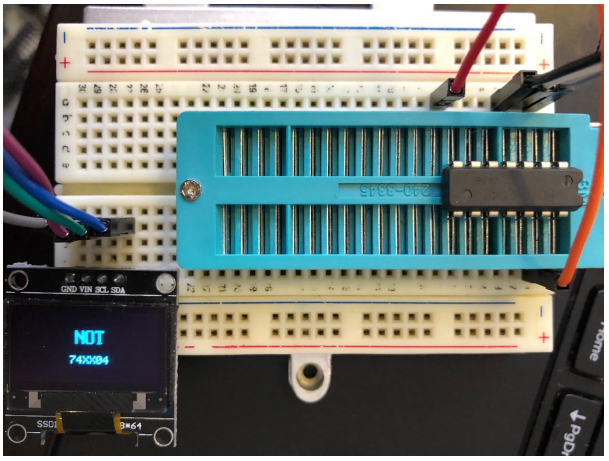

Fig. 14. *Foto do resultado obtido na Interface Gráfica ao acoplar um CI 74HC04 no soquete.*

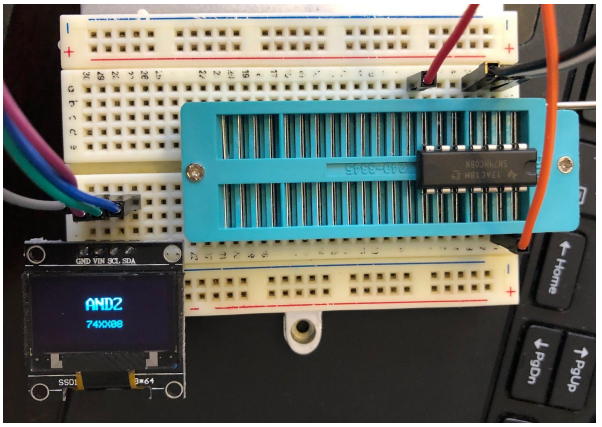

Fig. 15. *Foto do resultado obtido na Interface Gráfica ao acoplar um CI 74HC08 no soquete.*

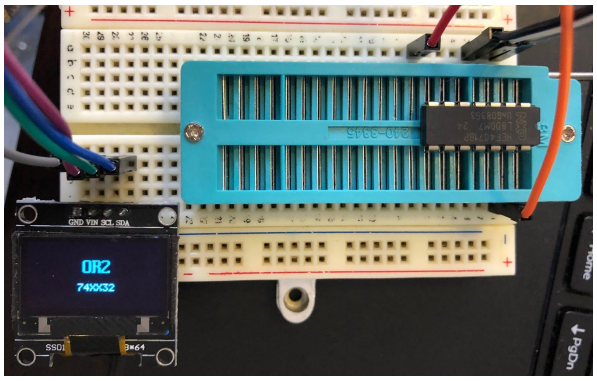

Fig. 16. Foto do resultado obtido na Interface Gráfica ao acoplar um CI 74HC32 no soquete.

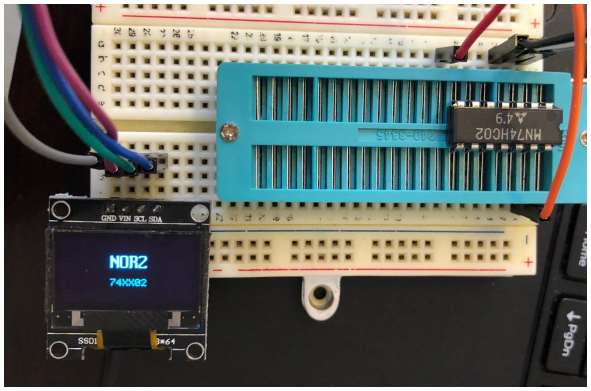

Fig. 17. Foto do resultado obtido na Interface Gráfica ao acoplar um CI 74HC02 no soquete.

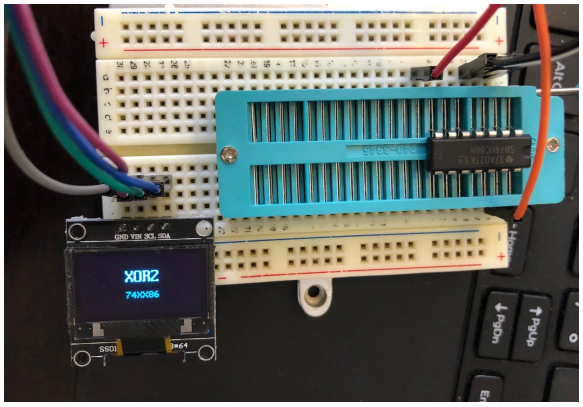

Fig. 18. Foto do resultado obtido na Interface Gráfica ao acoplar um CI 74HC86 no soquete.

III. Conclusão

Ao fim desse trabalho, foi possível desenvolver o projeto ao nível almejado, solucionando o problema principal, a identificação de circuitos integrados lógicos - da família 74, TTL - e criando material suficiente para a continuação do mesmo voltado ao mercado eletrônico.

Dentre os maiores desafios enfrentados durante o projeto, vale citar a configuração do barramento I2C e a otimização do armazenamento das *truth tables* na memória RAM do controlador.

No caso do I2C, a maior dificuldade foi criar funções robustas, como as utilizadas no início da implementação na biblioteca *Wire*. A solução foi, à partir de topologias de configuração do barramento I2C para a *MSP430*, recriar as próprias funções do arquivo *Wire.cpp*.

Para o caso do armazenamento, primeiramente foi excluído um nível de caso teste da lógica combinacional, isso porque para todos os casos de duas entradas, aplicar “01” ou “10” deverá apresentar o mesmo resultado. Não suficiente, as tabelas foram implementadas de forma que os resultados eram verificados com *if*s e *else*s dentro das funções teste de cada porta lógica.

Em conclusão, com todos esses obstáculos vencidos, foi possível entregar o protótipo funcional do projeto, montado em *protoboard*, com os seguintes entregáveis:

I. Relatório completo do desenvolvimento do projeto;
II. Protótipo funcional, montado em *protoboard*;
III. Código principal + biblioteca da Interface Gráfica.
IV. Precificação do produto final;

Durante o desenvolvimento, foi considerado utilizar um expansor de portas IOs, o *MCP23017*, que também utiliza o protocolo I2C e desafogaria o uso das portas do microcontrolador, já que a lógica requer doze GPIOs, sem falar, também, que esta tem maior capacidade para drenar corrente para os *chips* lógicos.

Outro ponto pensado para a continuação do projeto é a expansão da identificação para circuitos integrados com mais de 2 entradas, por exemplo CIs com três entradas (ex: AND tripla).

Bibliografia

[1] KainkaLabs. An IC-tester from Ebay. 2017. (20m16s). Disponível em: <https://www.youtube.com/watch?v=66hPg_r4IOw>. Acesso em: 09/04/2019.

[2] How to Identify Integrated Circuit Chips. In: ItStillWorks, “ED Wagner”. Disponível em: <https://itstillworks.com/identify-integrated-circuit-chips-8636383.html>. Acesso em: 09/04/2019.

[3] DAVIES, Jhon. MSP430 Microcontroller Basics. UK: Newnes, 2008.

[4] INSTRUMENTS, Texas. Mixing C and Assembler with MSP430 MCU’s. SLAA140A: 2002.