**Relatatório Final**

Conectando-se à Internet das Coisas

Universidade Estadual de Feira de Santana

**Build 2.0a**

# Histórico de Revisões

| Data | Descrição | Autor(s) |
|---|---|---|
| 03/08/2019 | • Desenvolvimento do relatório; | Alyson Dantas, Lucas Cardoso e Marcos Vinícius |

# SUMÁRIO

# 1 | Introdução

## 1. Propósito do Documento

Este documento descreve a arquitetura do projeto Conectando-se à Internet das Coisas, incluindo especificações do circuitos internos e máquinas de estados de cada componente. O principal objetivo deste documento é definir as especificações do projeto Conectando-se à Internet das Coisas, descrevendo o processo de desenvolvimento de interfaces de E/S para o processador NIOS II.

## 2. Acrônomos e Abreviações

| Sigla | Descrição |
|---|---|
| UART | Universal Asynchrounous Receiver/Transmiter |
| LCD | Liquid Crystal Display |
| IoT | Internet of Things |
| AP | Access Point |

## 3. Visão Geral do Documento

O presente documento é apresentado como segue:

- **Capítulo 2** – Este capítulo apresenta uma visão geral da arquitetura, com foco em entrada e saída do sistema e arquitetura geral do mesmo;
- **Capítulo 3** – Este capítulo apresenta uma descrição geral da arquitetura e apresenta os resultados obtidos, com foco nas etapas de desenvolvimento, configurações necessárias para o funcionamento dos módulos e características físicas do hardware desenvolvido;

## 4. Definições

| Termo | Descrição |
|---|---|
| Qsys | System integration tool in Quartus Prime |
| Firmware | Conjunto de instruções embarcadas no hardware |

## 5. Objetivos do Projeto

O sistema desenvolvido possui como objetivo reutilizar a IHM desenvolvida no projeto anterior, de forma que neste novo projeto seja interligada ao processador por meio de uma interface de E/S (UART) o módulo ESP-12 que utiliza o CI ESP8266 para prover comunicação sem fio. Um dos requisitos do projeto é a implementação da UART e a configuração (via processador) do módulo ESP-12 utilizando Comandos AT.

# 2 | Visão Geral da Arquitetura

## 1. Arquitetura do Projeto

A arquitetura geral do projeto é apresentada nessa sessão, bem como os elementos de E/S implementados no processador

A figura 2.1 representa a arquitetura do projeto, que é composto pelo processador geração um NIOS II em sua versão mais simples (NIOS II/e) que se conecta aos componentes do projeto anterior: PI/O Botões, JTAG, Controlador LCD e Controlador 7 Segmentos, e possui uma conexão agora com a UART, responsável pela transmissão serial dos dados para o módulo ESP-12

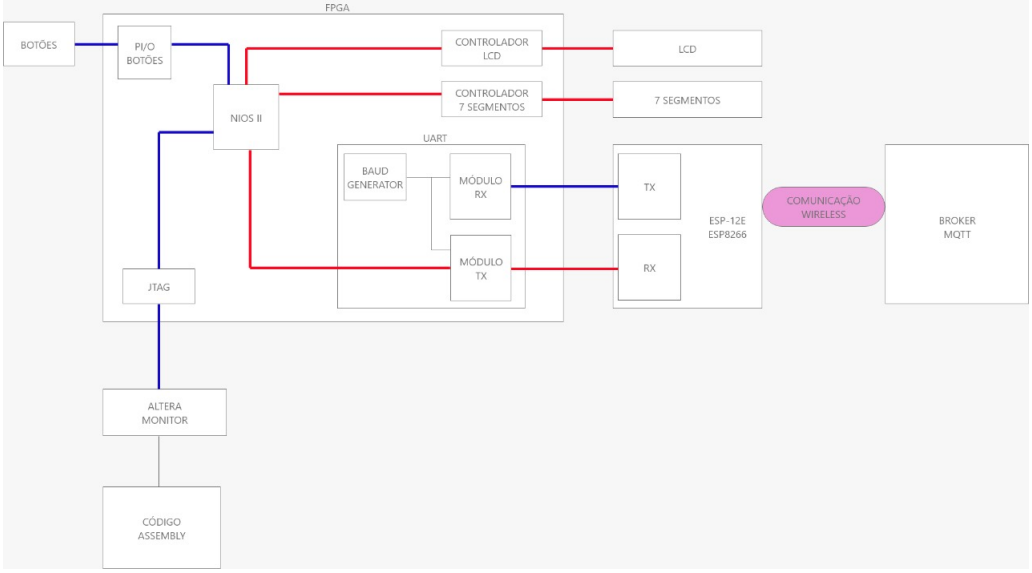


**Figura 2.1: Arquitetura do projeto**

*Especificações do projeto*

| | |
|---|---|
| Processador | NIOS II/e |
| Memória | *On-Chip Memory com 8192 bytes* |
| Periféricos de E/S | Botões, LCD 16x2, Display 7 segmentos, Interface RS232 |
| Placa | FPGA da Familia Cyclone IV com o chip EP4CE6E22C8 e clock de 50MHz |

## 1.1. UART

Para realizar a comunicação entre o processador e o módulo ESP foi necessário a implementação de uma UART, dispositivo capaz de enviar e receber informações via um canal de comunicação serial. A implementação foi realizada utilizando o componente *altera_up_avalon_rs232* do próprio Qsys, sendo associado no momento da geração do processador baseado no NIOS II/e.

Para a comunicação UART NIOS e UART ESP, foi utilizado a comunicação serial assíncrona que consistem em uma comunicação sem um relógio externo comum aos dois, sem uma sincronia entre os dois lados, o que define o inicio e fim de uma transmissão através de uma flag que é um bit de inicio e um de final, com oito bits de dados e sem bit de paridade.

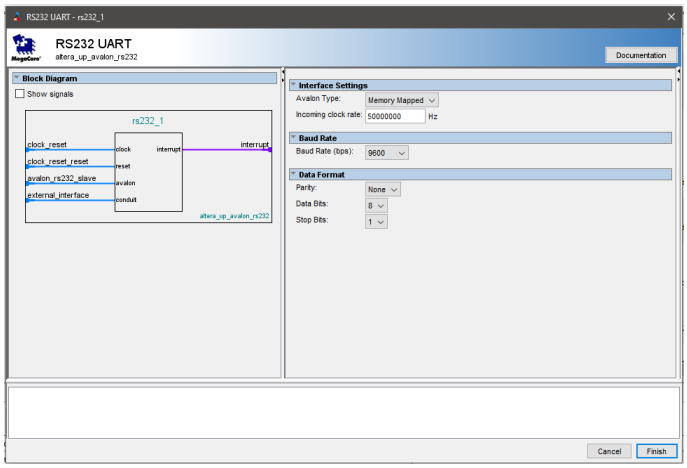


**Figura 2.2: Componente RS232 UART do Qsys**

## 1.2. MQTT

A IBM criou por volta do final dos anos 90 um protocolo de comunicação chamado de MQTT (Message Queuing Telemetry Transport). É um protocolo de mensagens leve para sensores e dispositivos.

O MQTT tem um arquitetura simples, onde tem dois personagens principais, o broker que é o computador principal e vários outros clientes que pode estar conectados ao serviço. Ele trabalha com o modelo chamado Publicador-Subscritor, onde o publicador adiciona uma mensagem em um tópico específico e o subscritor recebe toda e qualquer atualização.

No projeto o MQTT foi utilizado para publicar no tópico 'SDTopic', a informação publicada é a opção que foi selecionada no sistema. Essa informação é enviada para um broker pré-configurado.

## 1.3. Comandos AT

Comandos AT são uma linguagem de comandos para modens, desenvolvida pela empresa Hayes, sendo adotado como um padrão pela industria. Eles são constituídos de

uma cadeia curta de texto que se combinam para emitir comandos completos para realizar diferentes operações como configurar parâmetros de conexão, velocidade, etc.

## 2. Síntese Lógica

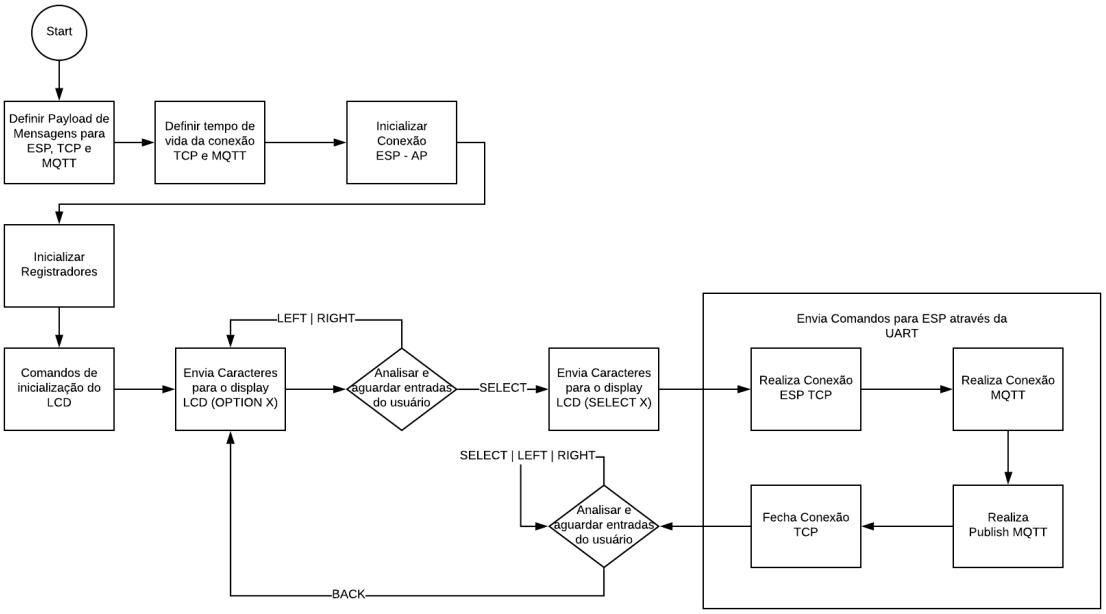


**Figura 2.3: Diagrama de Estados**

Optou se pela linguagem Assembly para desenvolver o software da IHM, responsável por gerenciar todo o hardware, essa escolha de uma linguagem de baixo nível foi feita através de um requisito de desenvolvimento para a IHM desenvolvida no projeto anterior, portanto manter essa opção foi decidida ao observar que dessa maneira proporcionaria desenvolver e manipular uma aplicação com comunicação direta hardware com hardware. Além de que mudar a linguagem traria a necessidade de desenvolver um novo código sem reuso do código pronto e funcional do projeto anterior, utilizando os mesmos hardwares que já desenvolvidos, portanto seria um trabalho a mais.

Esse software é responsável pelas seguintes rotinas na seguinte ordem exemplificada pela figura 3.1: Primeiro, durante a inicialização é necessário definir payloads para as mensagens de controle do modulo ESP, payloads para as mensagens para a comunicação TCP e definir payloads para os pacotes MQTT. Na segunda operação é necessário definir um tempo de limite de vida para as comunicações em TCP e MQTT, essas foram definidas em MQTT de um minuto e TCP de 18 horas. Terceira rotina é necessário realizar a inicialização da ESP e uma conexão com um Access Point, nesse caso com o SSID “WLessLEDS” e senha “HelloWorldMP31” para que seja possível encontrar um Broker MQTT no servidor de endereço TCP/IP “192.168.1.201:1883”.

A quarta operação faz parte do projeto anterior a IHM que já foi desenvolvida,

nessa etapa realiza uma inicialização de registradores que serão utilizados para a programação, como endereçamento de botões, LCD, display de 7 seguimentos, estado atual da máquina, dentre outros, considerando que o endereço da UART já foi previamente definido na operação um. Na quinta rotina ocorre a inicialização do LCD, para que ele opere corretamente durante o processamento e não contenha nenhum tipo de lixo nas mensagens exibidas. A sexta rotina é de enviar caracteres para o LCD, exibindo em qual a opção do menu se encontra, na primeira execução após sair da rotina de inicializar LCD é enviado “OPTION A” referente a primeira opção. O sétimo estado é de analisar e aguardar interação do usuário com os botões LEFT, RIGHT ou SELECT, sendo os dois primeiros responsáveis por mudar o estado da maquina e voltar para a sexta rotina para limpar o LCD e escrever a nova mensagem exibida no mesmo, sendo que vai variar de A,B,C,D e E para os cinco estados e os botões vão trocar entre eles na ordem indicada pelos seus nomes, o botão de SELECT é responsável por passar o hardware para a etapa seguinte. A oitava operação é a rotina de exibir para o usuário qual a opção foi selecionado com a mensagem SELECT X, sendo o X a opção A,B,C,D ou E em referente ao estado atual.

A nona rotina foi desenvolvida especificamente para esse projeto, onde o NIOS inicia uma conexão TCP utilizando a ESP através da UART, nessa conexão, é definido um socket para manter essa comunicação com o servidor. A decima operação inicia a conexão a conexão com o Broker MQTT, preparando para o envio do publish. Este, fica na decima primeira etapa, onde realizamos um publish com a seguinte mensagem “OPTION X” sendo o X referente a opção A,B,C,D ou E, que é o estado atual. A decima segunda rotina fecha a conexão TCP e consequentemente a comunicação MQTT.

Por fim a decima terceira operação retorna para o projeto anterior, é responsável por analisar o Botão de BACK para retornar ao sexto estado, deve também travar para que os botões SELECT, RIGHT e LEFT nesse estado não realizem nenhuma operação, apenas a já citada operação de BACK, retornando para o loop de operações de seleção.

# 3 | Descrição da Arquitetura e Resultados

## 1. Conexão UART/NIOS e ESP

A primeira etapa do desenvolvimento do projeto foi, realizar a conexão física entre o processador e o ESP e o estabelecimento da comunicação dos dois por meio de uma UART.

### 1.1. Configuração da UART(NIOS)

O primeiro passo foi realizar a construção do processador utilizando a ferramenta *Qsys* do Quartus, onde utilizamos como referência o processador gerado para o projeto anterior, realizando a penas a adição do componente *altera_up_avalon_rs232* ao projeto.

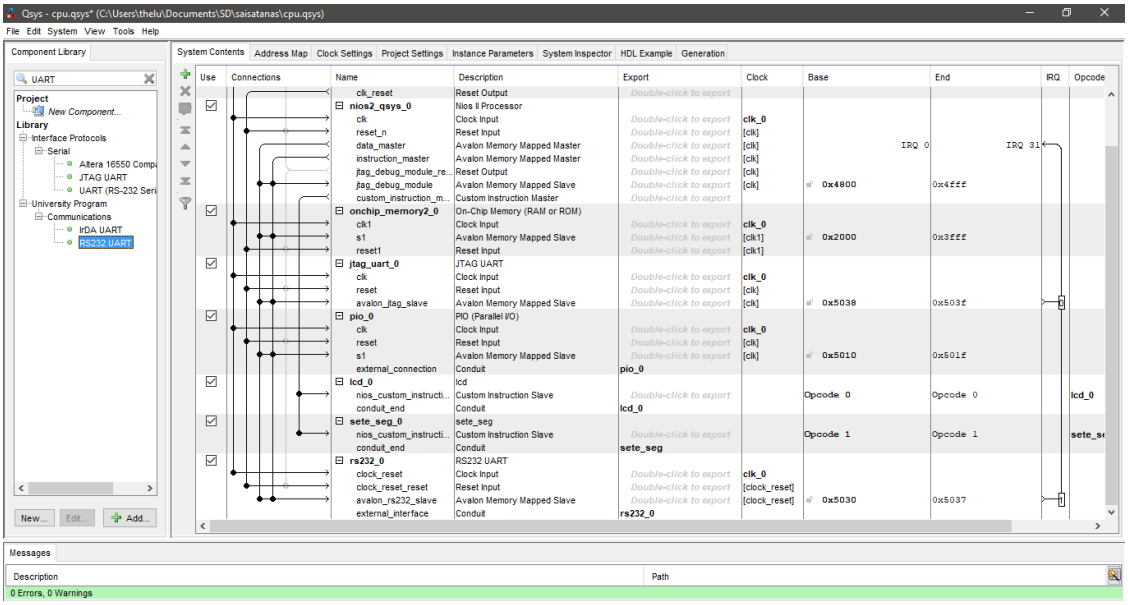


Figura 3.1: **Implementação da UART no processador**

Foram estabelecidas as seguintes configurações para a UART como ilustra a Figura 2.2: *baud rate:* 9600 Hz, 8 bits de dados, 1 bit de Stop, sem paridade e utilizando memória mapeada como interface entre o processador e hardware.

Ao implementar o módulo no processador, é mapeado na memória um espaço onde acontecem a "troca de informações" entre a UART e o módulo ESP, a Figura 3.2 descreve o espaço mapeado na memória para o funcionamento do componente.

*Table 2. RS232 UART Core register map*

| Offset in bytes | Register Name | R/W | Bit description | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 31...24 | 23...16 | 15 | 14...11 | 10 | 9 | 8 | 7 | 6...2 | 1 | 0 |
| 0 | data | RW | *(1)* | RAVAIL | RVALID | *(1)* | | PE | *(2)* | *(2)* | DATA | | |
| 4 | control | RW | *(1)* | WSPACE | *(1)* | | | WI | RI | *(1)* | | WE | RE |

*Notes on Table 2:*

(1) Reserved. Read values are undefined. Write zero.

(2) These bits may or may not exist, depending on the specified **Data Width**. If they do not exist, they read zero and writing has no effect.

**Figura 3.2: Descrição da memória mapeada do módulo *altera_up_avalon_rs232***

*Funções de Envio*

Para realizar o envio de dados com o componente foi utilizado como base um código em linguagem assembly para UART da Altera encontrado neste documento sendo adaptado ao nosso projeto e a arquitetura existente do projeto anterior, as Figuras 3.3 e 3.4 mostram o código de envio base e o modificado para o projeto.

```
/*****************************************************************************
 * Subroutine to send a character to the RS232 UART.
 *      r4  = RS232 UART base address
 *      r5  = character to send
 *****************************************************************************/
            .global    PUT_CHAR
PUT_CHAR:
            /* save any modified registers */
            subi       sp, sp, 4              /* reserve space on the stack */
            stw        r6, 0(sp)              /* save register */

            ldwio      r6, 4(r4)              /* read the RS232 UART Control register */
            andhi      r6, r6, 0x00ff         /* check for write space */
            beq        r6, r0, END_PUT        /* if no space, ignore the character */
            stwio      r5, 0(r4)              /* send the character */
END_PUT:
            /* restore registers */
            ldw        r6, 0(sp)
            addi       sp, sp, 4

            ret
```

**Figura 3.3: Código de envio de caractere**

```
# *****************************************************************************
# Rotina para envio de caracteres RS232 UART.
# r4 = Endereco base RS232 UART
# r5 = caractere para ser enviado
# *****************************************************************************
PUT_CHAR:
    subi sp, sp, 12                         # Reserva espaco na pilha
    stw r6, 0(sp)                           # Salva os registradores que serão utilizados
    stw ra, 4(sp)
    stw r4, 8(sp)
    movia r4, BASE_ADDRESS_UART             # Colocando em r4 o endereco base da uart

    call _delay_50ms
    ldwio r6, 4(r4)                         # Ler registrador de controle da uart
    andhi r6, r6, 0x00ff                    # Verifica se ha espaco para escrever
    beq r6, r0, END_PUT                     # Caso não tenha, o caracter não é enviado
    stwio r5, 0(r4)                         # Enviando letra
END_PUT:
    ldwio r6, 0(sp)                         # Recupera registradores
    ldwio ra, 4(sp)
    ldwio r4, 8(sp)
    addi sp, sp, 12
    ret
```

**Figura 3.4: Código de envio de caractere adaptado**

## 1.2. Configuração do ESP(Firmware e UART)

O módulo ESP utilizado no projeto possui um Firmware com Comandos AT e trabalha normalmente a um *baud rate* de 115200 Hz, afim de construir um projeto mais próximo do cenário comum, foi alterado a velocidade do *baud rate* para o valor de 9600 Hz, velocidade mais comum em dispositivos com comunicação serial assíncrona, essa alteração foi realizada utilizando o comando AT: AT+UART_DEF=9600,8,1,0,0, onde respectivamente, o 9600 é a velocidade do *baud rate*, 8 representa o número de dados de cada pacote, 1 representa a quantidade de Stop bits, 0 representa a ausência de paridade (par ou ímpar) e 0 representando a ausência de qualquer controle de fluxo.

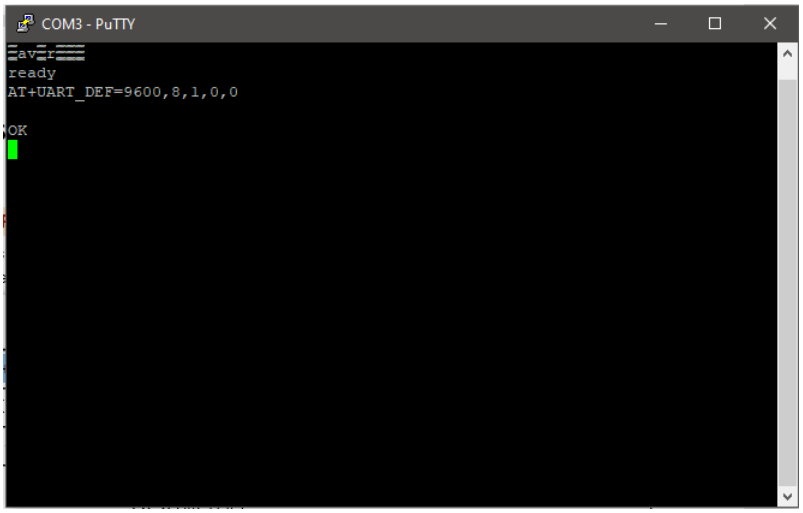


Figura 3.5: **Configuração do *baud rate* do ESP**

## 1.3. Configuração da conexão com AP do ESP(Comandos AT)

Com o módulo configurado para a velocidade desejada, o próximo passo foi realizar uma conexão do módulo ESP com uma rede Wireless existente, para isso utilizamos o seguinte comando: AT+CWJAP_CUR ="WLessLEDS","HelloWorldMP31", onde WLessLEDS é referente ao SSID da rede e HelloWorldMP31 é a senha, ao executar esse comando recebemos como resposta a confirmação, que a conexão foi estabelecida e que o ESP recebeu um endereço de IP válido.

Para garantir que o módulo estava conectado, realizamos uma verificação utilizando o comando AT+PING="192.168.1.103", onde 192.168.1.103 era um computador também conectado na mesma rede. A Figura 3.6 apresenta o resultado da conexão do ESP na rede wireless e a execução do comando PING.

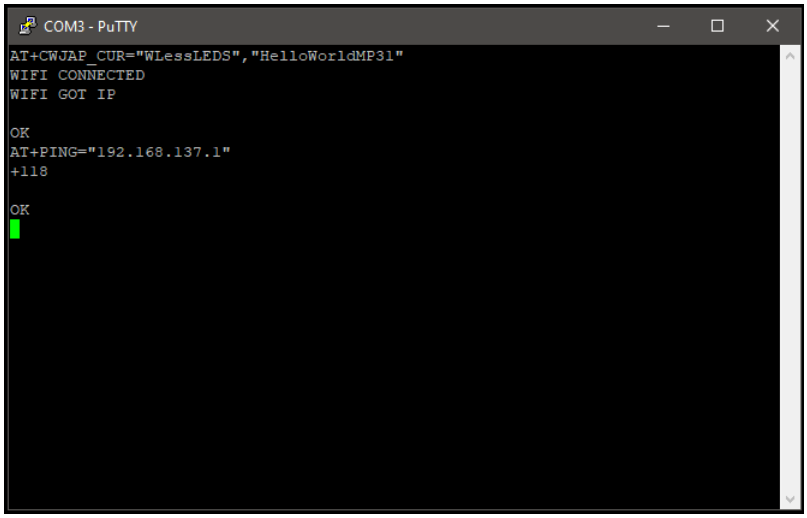


Figura 3.6: **Conectando e testando conexão ESP na rede Wireless**

## 2. Configuração do pacote MQTT

O serviço é executado sob o protocolo TCP, existe outra variação do protocolo que pode ser usado sem o TCP. Ele tem mensagens pré-definidas no seu cabeçalho. Além disso, o serviço possui alguns pacotes de controle, sendo alguns deles: CONNECT, PUBLISH, SUBSCRIBE e outros. Esse pacote serve para informar ao broker qual a ação do dispositivo que quer se conectar a ele.

O pacote de CONNECT é necessário para informar ao broker que o dispositivo vai iniciar um conexão com o servidor. O sistema projetado envia esse pacote de controle, sempre que existe a necessidade de publicar em um tópico.

| | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|---|
| Protocol Name | | | | | | | | | |
| byte 1 | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | Length LSB (4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| byte 3 | 'M' | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| byte 4 | 'Q' | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| byte 5 | 'T' | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| byte 6 | 'T' | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| Protocol Level | | | | | | | | | |
| | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 7 | Level (4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Connect Flags | | | | | | | | | |
| byte 8 | User Name Flag (1)<br>Password Flag (1)<br>Will Retain (0)<br>Will QoS (01)<br>Will Flag (1)<br>Clean Session (1)<br>*Reserved* (0) | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| Keep Alive | | | | | | | | | |
| byte 9 | Keep Alive MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 10 | Keep Alive LSB (10) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Figura 3.7: **Pacote CONNECT [1]**

Com ele pode configurar as *flags*, para a forma de operação que será utilizado. No projeto foi usado o QoS de nível zero, dessa forma não precisando de confirmação do broker informando que a mensagem chegou.

## 2.1. Payloads e Pacotes para publish

Para que o sistema consiga mandar uma mensagem para o broker, algumas regras devem ser seguidas. Uma delas é o envio do pacote de conexão, que foi mostrado no tópico anterior. Após isso, o sistema tem que enviar um pacote chamada de *PUBLISH*, para que possa envia uma mensagem e o broker exiba.

O pacote tem uma representação em cada byte, sendo elas explicadas em seguida. O cabeçalho do pacote *PUBLISH*, contém as *flags* informadas na figura abaixo.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| byte 1 | MQTT Control Packet type (3) | | | | DUP flag | QoS level | | RETAIN |
| | 0 | 0 | 1 | 1 | X | X | X | X |
| byte 2 | Remaining Length | | | | | | | |

**Figura 3.8: Cabeçalho PUBLISH [1]**

Na figura 3.8, mostra que no primeiro byte, os quatro primeiros bites são utilizados para identificar o tipo da mensagem. O restante do byte é informado o nível de qualidade e outras configurações. As *flags* que não foram utilizadas, foram mantidas nos valores padrão.

| | Description | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| Topic Name | | | | | | | | | |
| byte 1 | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | Length LSB (3) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| byte 3 | 'a' (0x61) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| byte 4 | '/' (0x2F) | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| byte 5 | 'b' (0x62) | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| Packet Identifier | | | | | | | | | |
| byte 6 | Packet Identifier MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 7 | Packet Identifier LSB (10) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

**Figura 3.9: Pacote PUBLISH [1]**

Em relação a figura 3.9, o primeiro e segundo *byte*, informa o tamanho do texto, em bytes, da mensagem que informa o tópico que será publicado. A partir do terceiro byte, é inserido o tópico onde a mensagem contida no pacote será publicada. Após a informação anterior, é usado mais dois bytes para informar o numero do pacote, serve como um identificador para a mensagem enviada.

Os próximos bytes são utilizados para informar o tamanho do *payload* da mensagem. Logo em seguida é colocado a mensagem que será publicada no broker.

# 3. Updates IHM Projeto 1

Ao longo do desenvolvimento do projeto atual, notamos que poderíamos melhorar o código do problema anterior. Para fazer uma sequência de envios de caracteres para o display LCD era necessário uma rotina que enviava byte a byte, sendo que esse bytes são valores, em hexadecimal, que correspondem a uma letra na tabela ASCII. Esse valor era adicionado manualmente em um registrador e era chamado a instrução customizada para o envio para o LCD.

Nessa atualização foi adicionado uma manipulação, melhor dos dados enviados para o LCD. O uso do espaço de dados, do código em assembly, simplifica a implementação de rotinas mais elaboradas.

A melhora consiste em salvar na memória o texto que será enviado para o LCD. Esse endereço, onde o texto foi salvo, é referenciado na rotina de envio. Essa rotina percorre o espaço de memória, referenciado enviando toda a palavra que tá escrita na memória, até achar o carácter responsável por final de texto.

Por causa dessa mudança, conseguimos reduzir o código em algumas linhas, além de deixar mais legível e facilitar o entendimento do sistema em si, evitando redundância de código e otimizando a execução.

# 4. Testes

## 4.1. Teste geral do projeto

# 5. Análise de Síntese Física

A partir dos relatórios gerados pelo *software* Quartus foi possível obter dados como a área total ocupada pelo circuito desenvolvido, o número de elementos utilizados, etc.

## 5.1. Área total ocupada pelo circuito

A tabela a seguir demonstra a quantidade de elementos internos da FPGA utilizados ao final do projeto.

| Elemento | Quantidade |
|---|---|
| LAB | 161 / 392 ( 41 % ) |
| LE | 1,977 / 6,272 ( 32 % ) |
| Regsitradores | 1120 |
| Memória | 78,848 / 276,480 ( 29 % ) |

**Tabela 3.1: Elementos internos da FPGA utilizados**

## 5.2. Elementos Lógicos por modo de operação

| Modo | Quantidade |
|---|---|
| Normal | 1668 |
| Aritmético | 203 |

## 5.3. Elementos Lógicos por modo de operação

## 5.4. Área total ocupada pelo circuito

A Figura 3.10 mostra a área total ocupada pelo projeto na FPGA, essa informação foi conseguida com o auxilio do *Chip Planer* disponível no Quartus.

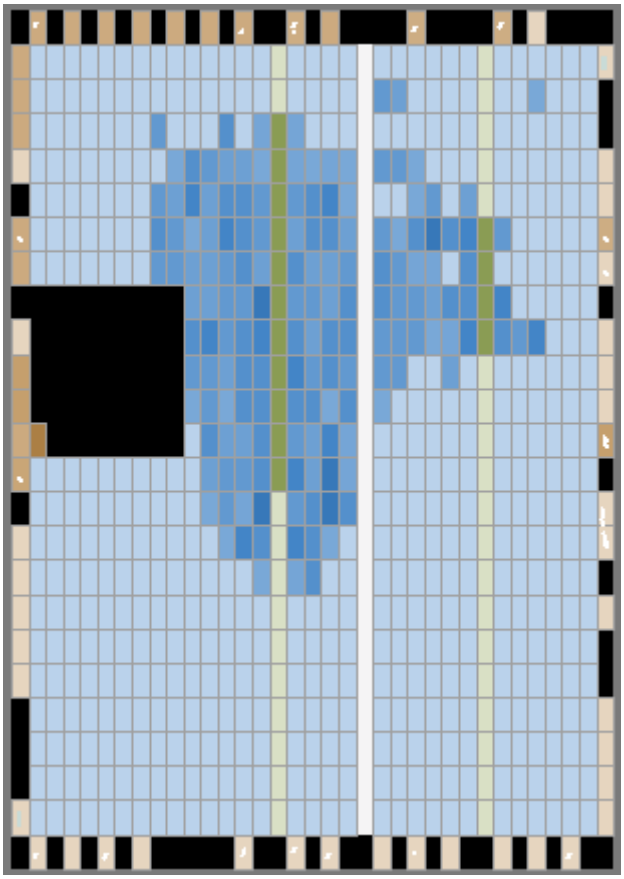

**Figura 3.10: Área total ocupada pelo circuito**

## 5.5. Caminho crítico do sistema

A partir dos relatórios gerados pela ferramenta *TimeQuest Timing Analyzer* também disponível no Quartus, foi possível observar o *caminho crítico* do sistema, ou seja, o segmento do circuito, no qual o trafego de informação tem o pior tempo de propagação até chegar no destino.

Assim, verificou-se que esse caminho crítico se dá entre o resultado de uma operação da ALU e a escrita em um registrador do NIOS II, medindo 14.107 de delay, conforme a Figura 3.11. Esse caminho, pode ser identificado no circuito através da ferramenta Chip Planner e é apresentado na Figura 3.12 que é apresentado a seguir, extraída da mesma:

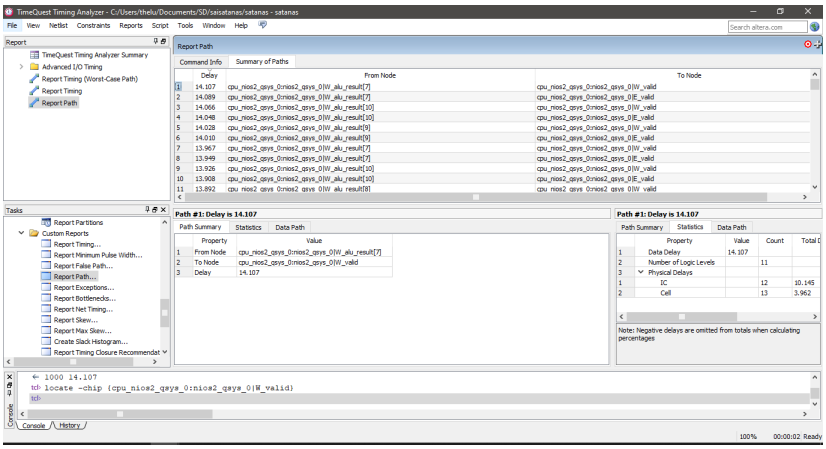

Figura 3.11: **informações do Caminho Crítico**

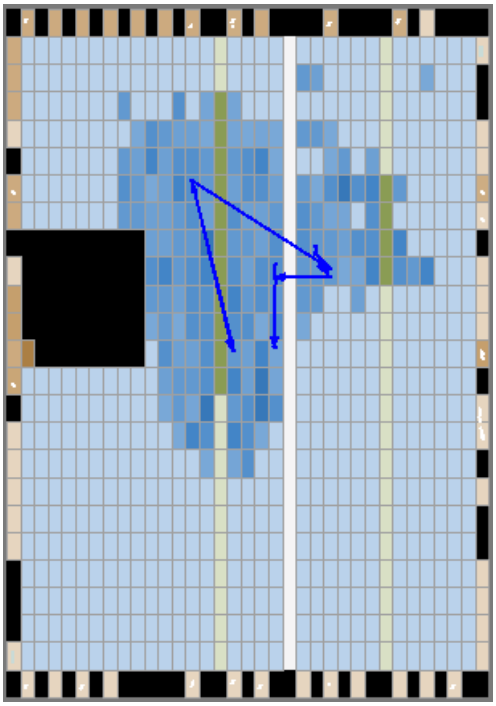

Figura 3.12: **Caminho Crítico do circuito**

## 6. Referências

[1] O. T. Committee, apr 2014.