UNIVERSIDADE ESTADUAL DE PONTA GROSSA – UEPG
SETOR DE CIÊNCIAS AGRÁRIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA – DEINFO

Turma de Engenharia de Computação – Eletrônica II

Orientador: Juan Camilo Castellanos Rodriguez

PROJETO FINAL: HEARTSENSE ASIC

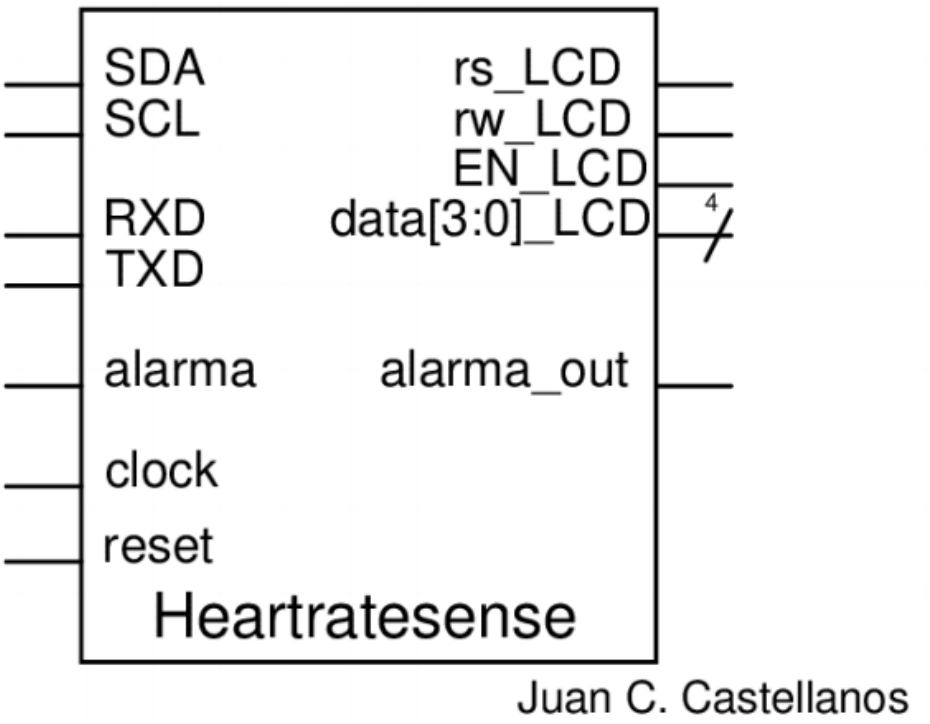
PONTA GROSSA
2019

SUMÁRIO

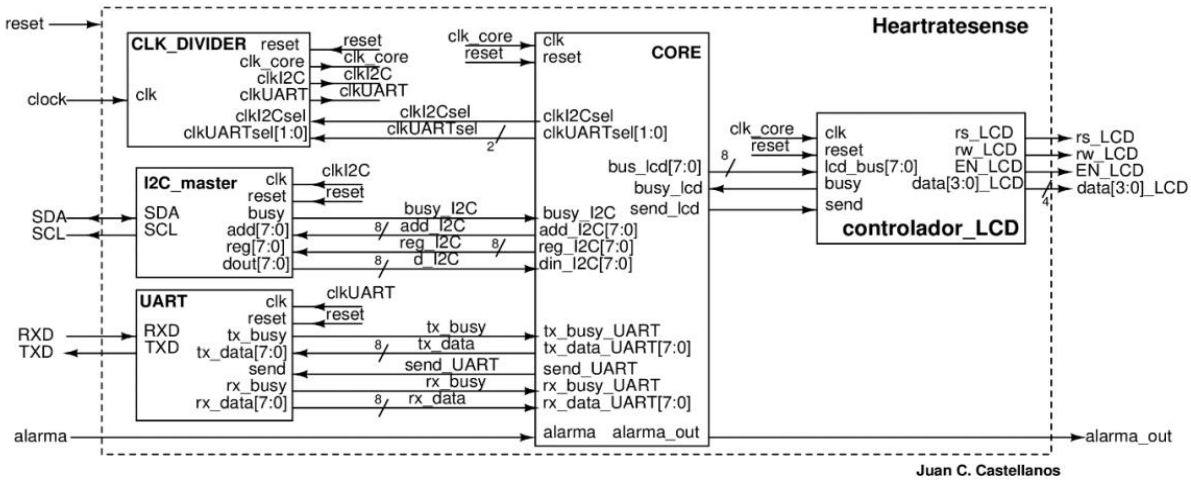
1. INTRODUÇÃO

1.1. OBJETIVO

Desenvolver um ASIC usando a tecnologia CMOS de 0.18µm (OSU018) que processe a informação dada por um sensor de batimento cardíaco através da porta I2C. O sistema deve contabilizar os batimentos por minuto (BPM) e deve visualizar o dado em um display LCD. O sistema também conta com um sistema de alarme, o qual pode ser acionado quando um interruptor externo é acionado ou quando os batimentos cardíacos (BPM) se encontrem fora da faixa permitida Blow<BPM.



O diagrama de blocos do seu ASIC é mostrado a continuação:



2. CLOCK DIVIDER

2.1. PROJETO

Foram definidas as entradas e saí'da do módulo, além de variáveis adicionais para auxiliar na divisão do *clock*. Utilizando as entradas de *clock*, *reset*, seleção do I2C e do UART, foram divididos os *clocks* e passados para as saídas do CORE, I2C e UART, utillizando os valores de divisão calculados, para obter os valores de *clock* desejados.

Foi feito um bloco sequencial *always* para borda de subida do *clock*, onde foram utilizados contadores para CORE, I2C e UART, indo de zero até o valor do divisor de cada *clock* de saída solicitado.

Foram feitos outros dois blocos sequenciais *always* para mudar o valor do divisor do I2C e do UART, conforme o valor de seleção das entradas clkI2Csel e clkUARTsel. Além disso, mais um bloco sequencial *always* foi feito para implementar o *reset*, que retorna o seletor do UART e do I2C para seus valores padrão, "10" e "0", respectivamente.

Para as saídas, foram utilizados os contadores e divisores dos três *clocks* de saída (CORE, I2C e UART), onde o *clock* de cada saída recebe 0 quando o respectivo contador for menor que metade do valor do divisor, e recebe 1 caso contrário. Assim, o divisor determinará a proporção do *clock* de entrada (clk) em relação às saídas, fornecendo os *clocks* requeridos pelos módulos CORE, I2C e UART.

2.2. VERIFICAÇÃO

Para a realização dos testes, primeiramente foram definidos valores para o clk, reset, clkI2Csel e clkUARTsel. Em seguida, foi definido um tempo de espera e então foram alterados os valores de clkI2Csel, clkUARTsel e reset, definindo intervalos de espera entre as alterações para que fosse possível visualizar o comportamento das saídas através de um gráfico de onda. Também foi definido um período de 60 unidades de tempo para o clk, para a realização dos testes.

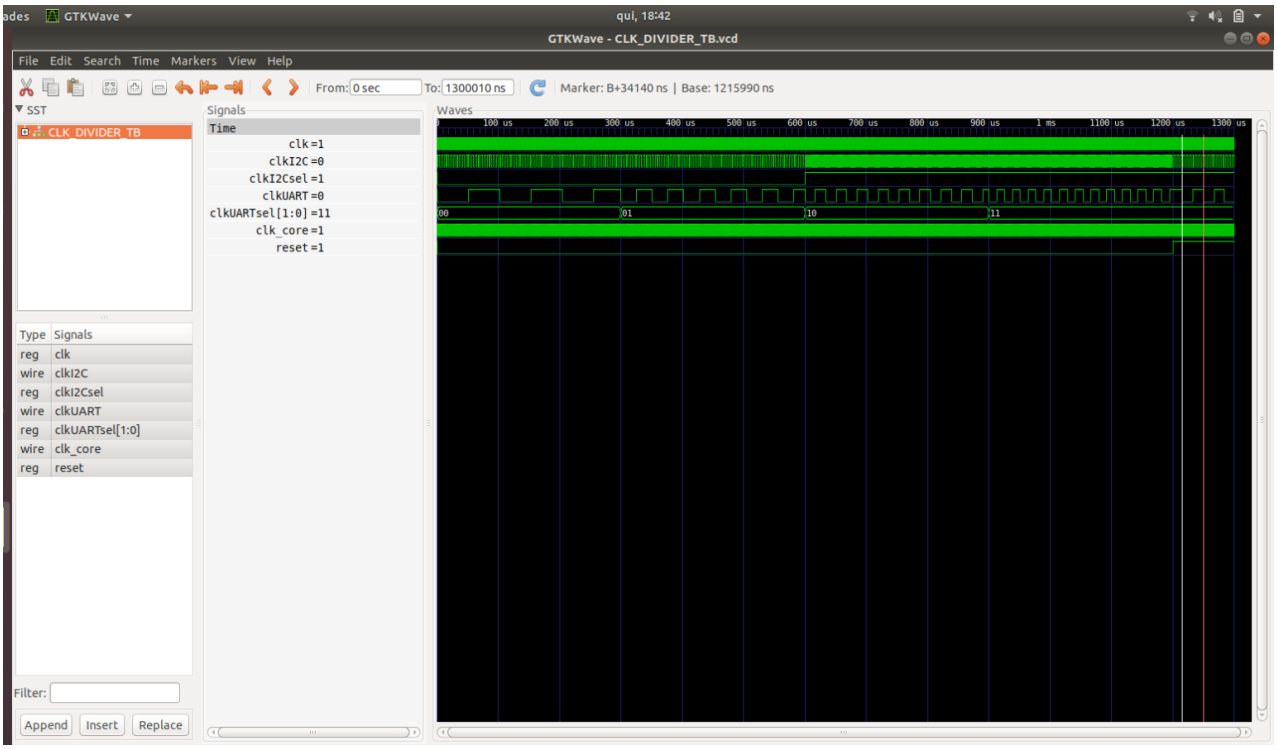
clk	clk (Teorico)	clk (Simulado)	Erro
----------	16384000	16666666,67	1,73
clk_core	**clk_core (Teorico)**	**clk_core (Simulado)**	
----------	1000000	980392,16	1,96
clkI2Csel	**clkI2C (Teorico)**	**clkI2C (Simulado)**	
0	200000	200803,21	0,4
1	800000	793650,79	0,79
clkUARTsel	**clkUART (Teorico)**	**clkUART (Simulado)**	
0	9600	9763,72	1,71
1	19200	19493,18	1,53
10	28800	29291,15	1,71
11	38400	39032,01	1,65

Comparação dos Resultados do Clock Divider

Obs: quando o reset assume o valor “1”, o CLKI2CSEL recebe “0” e CLKUARTSEL recebe “10”.

A partir dos resultados obtidos na tabela anterior, ao comparar os valores teóricos com os valores obtidos na simulação, foram obtidos os erros correspondentes. Como pode-se observar, o maior erro foi menor que 2%, o que indica que a divisão dos *clocks* apresenta uma precisão adequada ao projeto proposto.

A seguir é apresentado o resultado da simulação, em forma de onda.



3. I2C

3.1. PROJETO

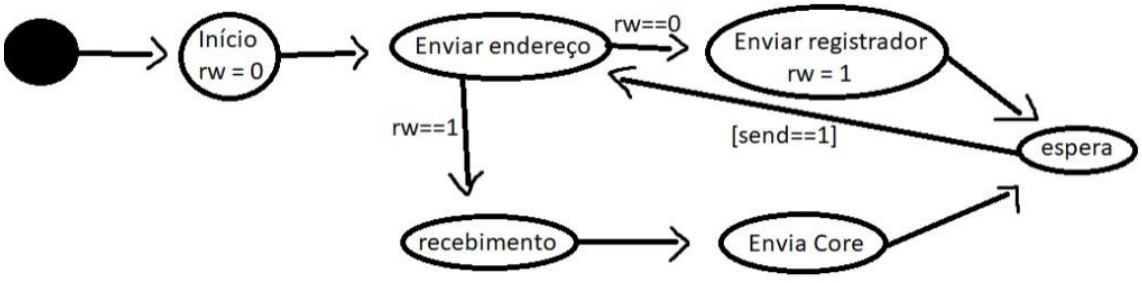
O objetivo específico do módulo I2C é executar a leitura bit a bit de um escravo, todo esse processo é feito baseado em uma máquina de estados Moore.

Primeiramente na implementação do código foi declarado as 6 entradas necessárias, uma para o Endereço (default: 0x0A), outra para o dado de registro (default: 0x00), uma para o reset, uma para o envio de dados, outra para o CLK e SDA. Além dessas entradas também foi preciso 3 saídas, sendo uma delas a mais importante, saída do batimento cardíaco e vale ressaltar a saída Busy que é utilizada pro Core entender que já é possível ler o dado solicitado, depois do Core ler esse dado ele desativa a entrada de envio e nesse momento o I2C fica em modo de espera.

Logo após foi preciso criar 6 estados, 2 estados para enviar endereço e outro registrador; Mais 2 estados, um para receber dados e o demais para enviar para o core; os 2 últimos são de espera e de início.

As variáveis são inicializadas com sda e scl = 1, no estado do inicio ocorre a descida do sda, e passa para o estado enviar_endereço, o qual toda subida de scl ele envia um bit do endereço, depois de enviados os 9 bits sendo o ultimo deles o ack=0, passa para o estado enviar_registro, o qual toda subida de scl ele envia um bit do registro, depois de enviados os 9 bits, ele muda para o estado espera e sda volta para 1.

Ele faz processo do enviar_endereço e agora com rw = 1, ele vai para o estado de recebimento, o qual todasubida de scl ele recebe um bit de dado, depois disso ele vai para o estado envio_core o qual depende do sinal sendo qual vem do core, para mandar os dados para a saída.

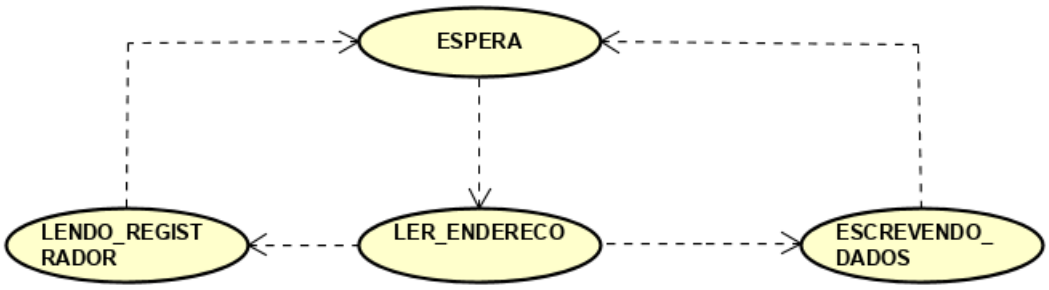


3.2. VERIFICAÇÂO

O projeto do I2C master funciona da seguinte maneira: contando com 4 estados, no primeiro estado, inicio, é realizado o primeiro envio de endereço ou registro, sendo esses com 9 bits. No segundo estado, envio, a tarefa é semelhante a do primeiro estado (?). No terceiro estado, recebimento, o endereço ou registro é recebido pelo master e, por fim, no quarto estado envio_core, o registro é enviado para o módulo core.

Tendo o projeto do I2C *master* foi possível iniciar, por sua vez, o projeto do I2C *slave. C*omo havia a necessidade de construir o I2C com máquina de estados, foi definido da seguinte maneira: O primeiro estado é de ESPERA(00), ou seja, é verificada a borda de descida do sda(dados) para poder mudar de estado. Quando a borda de descida é analisada, é definido um contador de bit iniciando em 0, que muda de estado(01), indo para LER_ENDERECO, onde é lido o endereço de 8 bits.

Quando chega no nono bit(“bit de verificação”), é feita uma comparação com o oitavo bit, o qual decide se o módulo estará apto para ler ou escrever. Já no terceiro estado, ESCREVENDO_DADOS(10), é obtido o valor do registrador e colocado na saída, assim podendo passar para o quarto estado, LENDO_REGISTRADOR(11). Este último tem a finalidade de pegar o que está escrito no registrador e armazenar no sda.



Máquina de estado do slave

Tendo os estados definidos foi possível construir o projeto do I2C *slave.* Após esse processo inicializou-se a verificação do mesmo, que constituía em definir uma

entrada por um tempo predeterminado e observar se os estados e as variáveis mudavam de acordo com o esperado, ou seja, realizava as operações propostas pelos estados definidos anteriormente.

4. CORE

4.1. PROJETO

Para início do projeto do core, foi desenvolvida sua máquina de estados, levando em consideração a maneira com que o core deve trabalhar, não especificando muito cada etapa, abstraindo e deixando de simples compreensão. Os estados foram subdivididos em: Início, ler dado, conferir se dado foi lido, incrementar contador, conferir se passou 1 segundo, conferir frequência cardíaca, acionar *buzzer*, converter para ASCII, enviar LCD, enviar *UART*. A seguir será descrito sucintamente o que cada estado realiza.

Início: é o estado inicial, ele é utilizado quando iniciado o dispositivo, quando resetado ou também quando o ciclo do core é finalizado após o envio do sinal a UART.

Ler dado: é o estado em que se inicia a leitura de dados da *UART*.

Conferir se dado foi lido: este estado confere se o dado da *UART* foi lido, ele possui dois casos, caso a leitura tenha sido feita o próximo estado o contador é incrementado para se ter controle de quantas vezes foram feitas leituras desses dados, caso a leitura não tenha sido feita o próximo estado pula o incremento do contador e o próximo estado é conferido se o tempo decorrido foi de 10 segundos.

Incrementar contador: simplesmente incrementa o contador utilizado para saber quantas vezes o dado foi lido.

Conferir se passou 10 segundos: estado utilizado para conferir se o tempo especificado já passou, caso sim o próximo estado será conferir a frequência cardíaca, caso não espera-se 100ms e volta ao estado de leitura de dado.

Conferir frequência cardíaca: de acordo com um intervalo especificado confere-se se a frequência cardíaca é maior que uma frequência mínima e menor que uma frequência máxima, caso estes limites se extrapolem o próximo estado é acionar o *buzzer*, caso a frequência esteja neste intervalo estabelecido o próximo estado é converter para ASCII, o intervalo estabelecido para frequência é de 50bpm a 120bpm.

Converter para ASCII: converte o valor de bpm medido para ASCII.

Enviar LCD: envia o valor convertido para ASCII para o *display* LCD.

Enviar *UART*: envia um sinal para a *UART* de que o ciclo foi completo e pode-se iniciar uma nova sequência de leitura.

A princípio as portas de entrada e saída não foram utilizadas no código verilog, foi feito apenas a lógica básica de funcionamento para o grupo de verificação fazer a correção do módulo utilizando-se de valores lógicos na entrada e saída em conjunto com o grupo de projeto.

Após a verificação notou-se a necessidade de unir em um estado todos os estados que precisavam ocorrer simultaneamente, os únicos estados que ficaram fora deste estado principal é o contar e o acionar *buzzer*.

4.2. VERIFICAÇÃO

A verificação do core iniciou com a inserção das portas de entrada e saída no código, também posteriormente foram feitas as máscaras necessárias para configurar o endereço do sensor I2C, registro lido do sensor I2C, *clock* do I2C e da *UART*, assim foi possível começar a verificação por *testbench,* começando pela checagem do *range* dos batimentos cardíacos entre 50-160 bpm, o alarme foi verificado caso os batimentos fujam dos limites já estabelecidos no *range,* verificado em sequência as saídas para *display* e *UART,* onde obedecem ao mesmo parâmetro do *send, a* verificação do contador também ocorreu observando seus sinais lógicos na saída.

É valido comentar que a verificação foi através de um modo empírico, onde assumimos a saída através de uma análise e com auxílio um software de simulação é possível simular o código e observar seus sinais de saídas externos e internos também, podendo comprovar a existência de erros.

Quando feito o *testbench* notou-se um erro onde após o *reset* o estado deveria ser o inicial, porém por alguma causa o estado subsequente era o de envio para a *UART*. Outro problema encontrado foi para implementar um segundo contador, o problema é que o mesmo necessitaria tomar conta do estado que deveria ser tomado em sequência, interrompendo assim o fluxo geral do core. Uma das maneiras seria implementar esse contador salvando o estado atual antes de assumir o novo estado e após a execução retornar ao estado armazenado previamente, ou então fazer este contador ser concorrente ao primeiro contador que é responsável pela condicional de enviar os dados a *UART* a cada um segundo, as duas maneiras seriam possíveis

porém a adotada para o projeto do *core* foi a concorrência. Uma nova variável chamada ler_I2C foi criada para informar a I2C quando o tempo de leitura for atingido.

Após a simulação do testbench, foi detectado um erro quando a escala de tempo é alterada, para uma escala menor, o verilog aponta um erro de “warning” que o timescale já está definido. Porém o código continua funcionando.

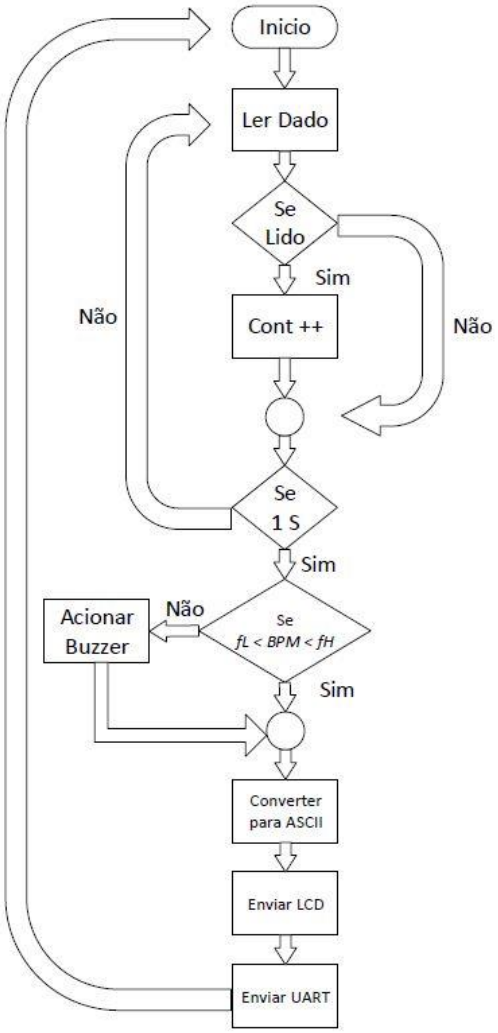
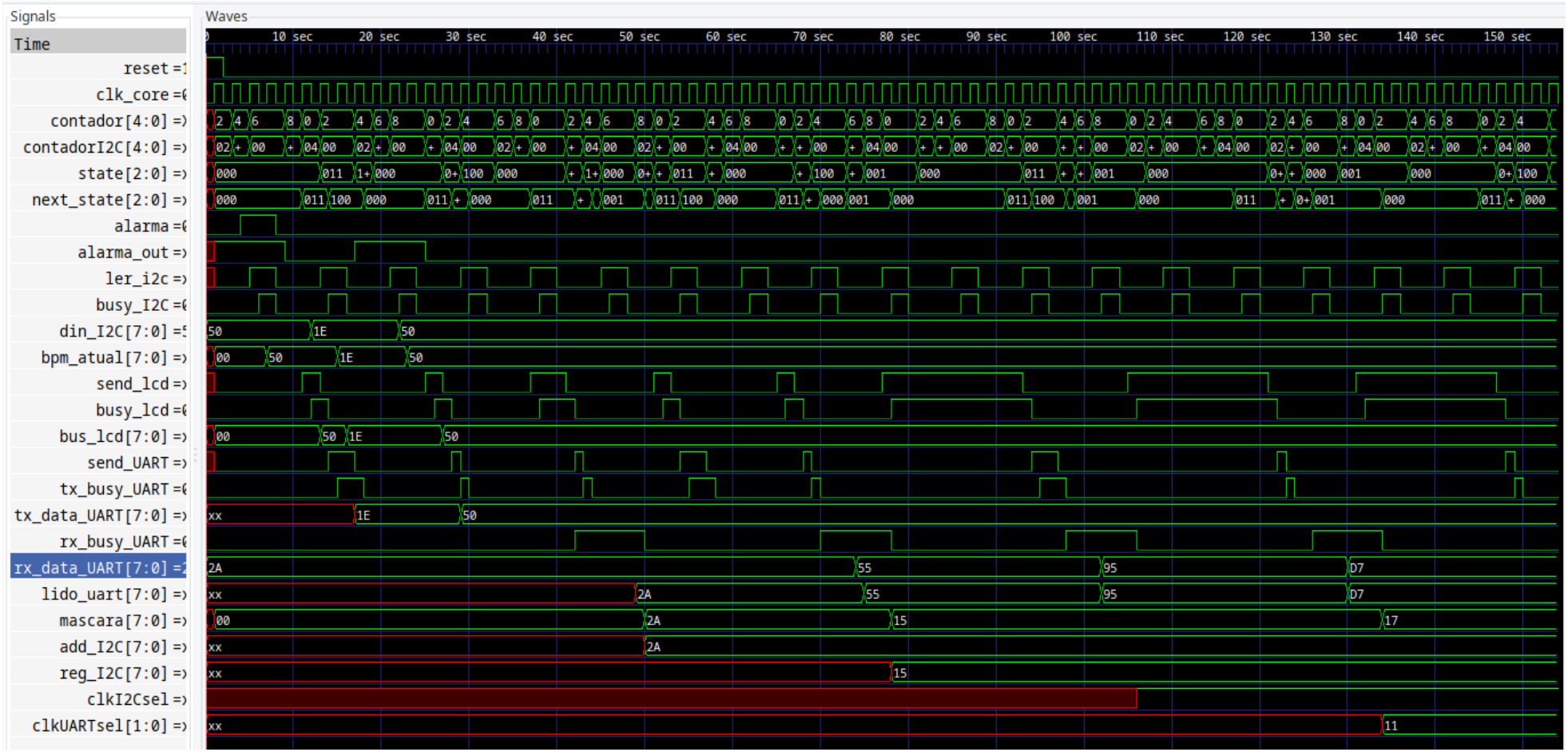


Diagrama de estados do core.



Waveform: Core

5. CONTROLADOR LCD HD44780

5.1. PROJETO

O LCD HD44780 tem objetivo mostrar os dados de batimento cardíaco.O LCD será utilizado no modo de 4 bits.

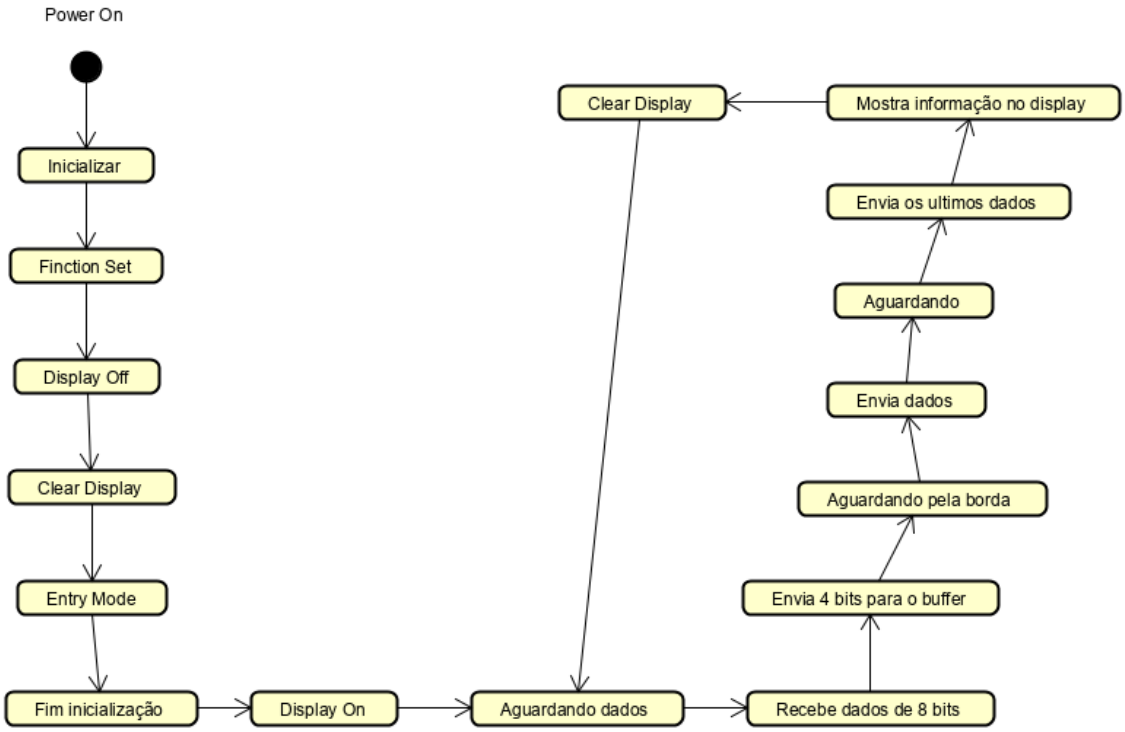


Diagrama de fluxo geral

Inicializar -> Rs=0, Rw=0, ENABLE=0, zera os dados do display com 8’b00000000.

Function Set -> se estado = 0, estado+=1. O estado é executado 4 vezes. No último estado ele troca para o modo de 4 bits.

Display off -> Desliga o display, o cursor e o blinking. D=C=B=0. RS = 0.

Clear Display -> Limpa o conteúdo do display. RS=0.

Entry mode -> Determina qual caminho o cursor irá fazer. I/D=1, S=0, Rs=0.

Display On -> Liga o display.

Recebe os dados 8 bits -> ENABLE=0, RS=1, RW=0.

Envia 4 bits para o buffer -> ENABLE=0, RS=1, RW=0.

Aguardando pela borda -> ENABLE=1, RS=1, RW=0.

Envia os dados -> ENABLE=1, RS=1, RW=0. Ocorre a borda de descida.

Envia os últimos 4 bits para o buffer -> ENABLE=0, RS=1.

Aguardando -> ENABLE=1, RS=1.

Envia os últimos dados -> ENABLE=0.

Mostra a informação no display -> ENABLE=1, RW=1.

A máquina de estados tem 3 principais estados:

- Inicialização -> é ativado o POWER ON para ligar o sistema e são setados rs_LCD, rw_LCD, EN_LCD, busy, data_LCD. Dessa forma o LCD começa a operar.

- Leitura de dados -> estado onde o sistema está disponível para leitura de dados. É verificado se o SEND foi enviado do core, se sim pode-se passar para o estado de leitura de dados.

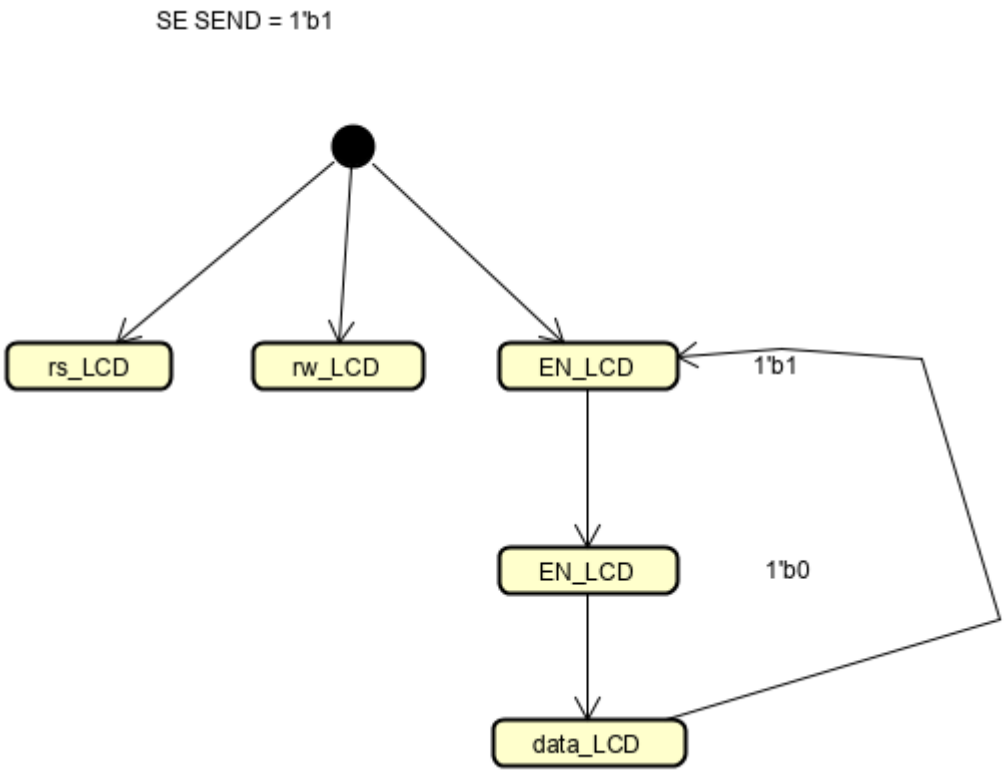


Diagrama de fluxo do leitura de dados

- Recebimento de dados -> estado onde pode-se receber dados.

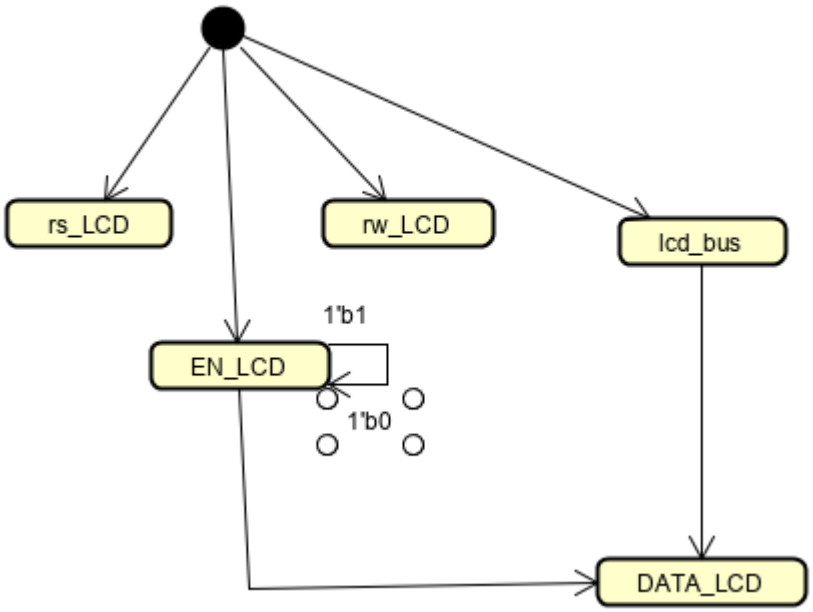
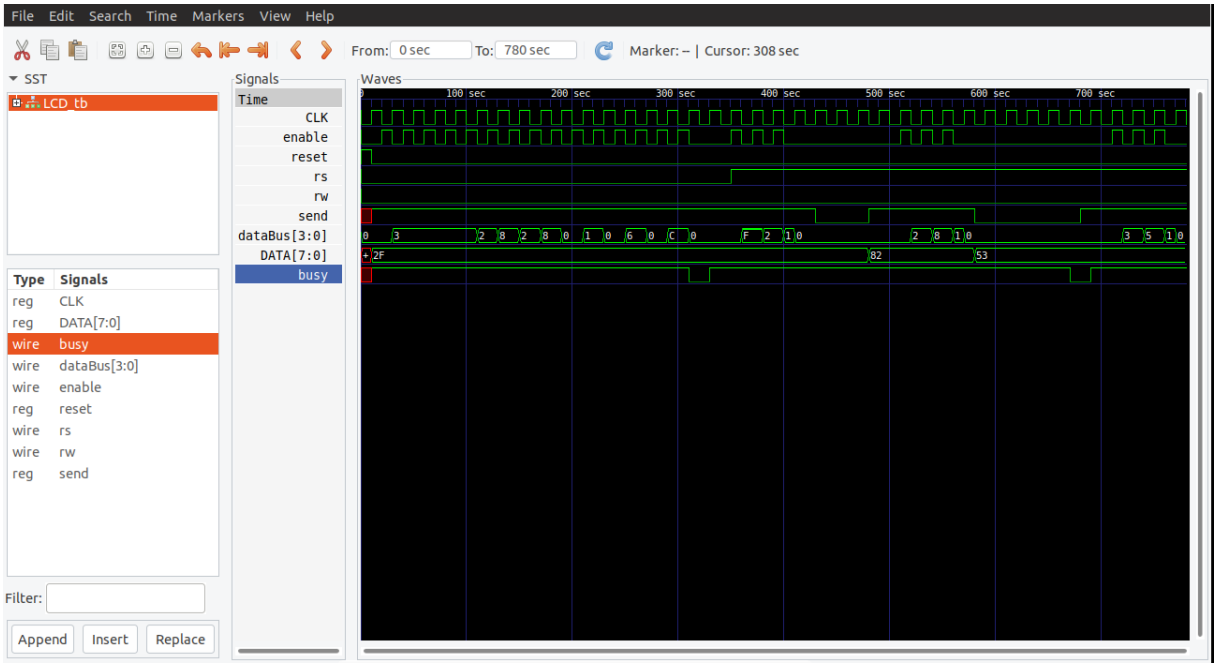


Diagrama de fluxo do recebimento de dados

5.2. VERIFICAÇÃO

Para os testes foi usado o teste bench do verilog. Para isso foram setados valores para CLK, RESET, SEND, DATA. O CLK é o clock e é inicializado com 1’b0 (LOW). O RESET é usado vezes em 1’b0, vezes 1’b1 (usado para setar o sistema, reiniciando as variáveis). SEND é uma flag para avisar se pode - se passar para o estado de recebimento de dados, portanto ele é modificado antes do DATA ser atualizado. DATA recebe um dado de 8 bits para os testes.

O teste bench foi feito para testar se a manipulação de dados por parte do sistema estava funcionando de acordo com especificado. Para testar isso usa - se o DATA com uma cadeia de 8 bits.



Waveform: LCD

6. UART

A UART, ou Universal Universal asynchronous receiver/transmitter, é um CI cuja função é a conversão de dados que são recebidos em série para um barramento de dados paralelos, ou vice-versa. Diz-se que a transmissão é assíncrona porque apesar de obedecer a um clock interno esse clock não é enviado junto com a mensagem. Sendo assim, o UART utiliza apenas duas portas para enviar e receber informação, sendo elas a RXD e TXD, que serão explicadas detalhadamente mais adiante.

6.1. PROJETO

Para o projeto foram implementados dois sub módulos, transmitter e o receiver, sendo o transmitter o responsável por envio de mensagens e o receiver pelo recebimento de mensagens. Os dois módulos foram projetados inicialmente na forma da uma máquina de estados para depois serem convertidos em verilog.

Como na transmissão UART o clock não é enviado, é necessário que sejam definidos um tamanho de mensagem e um bit de aviso. O bit de de aviso, também chamado bit inicial é uma forma de comunicar o receiver que uma mensagem está prestes a ser transmitida. Para esse projeto foi definido um bit inicial igual a 0 (zero) e um tamanho de mensagem de oito bits, optou-se por não utilizar bit de paridade.

Outras portas utilizadas foram o send, para avisar o transmitter que a mensagem pode ser enviada, o reset para reiniciar os módulos de forma síncrona e as portas rx_busy e tx_busy, para avisar que módulo está ocupado.

Transmitter

A função do transmitter é ler os dados inseridos em paralelo no barramento tx_data e enviar sequencialmente através da porta TXD, todas as portas do transmitter estão listadas na tabela. Sua Máquina de estados é composta por três estados básicos, sendo eles “esperando”, que é estado onde nenhuma mensagem está sendo transmitida e o TXD se mantém em sinal alto, quando a entrada send recebe o valor 1, significa que algo deve ser transmitido, então um bit 0 é enviado pela porta TXD para avisar que uma mensagem começará a ser transmitida, ao mesmo tempo a porta tx_busy é setada como 1 para que os outro módulos esperem que essa mensagem termine de ser enviada. O estado de envio sequencial é onde todos os bits do tx_data são enviados sequencialmente pela porta TXD.

Porta	Tipo
tx_data	entrada
clk	entrada
reset	entrada
send	entrada
RXD	saída
rx_busy	saída

Portas do Transmitter

O Receiver funciona de maneira oposta, ele é responsável por ler a mensagem sequencial enviada pelo transmitter de outro UART, e escrevê-las no barramento rx_data, todas as portas do transmitter estão listadas na tabela. Sua máquina de estados também é composta por três estados, no primeiro estado que também foi nomeado “esperando” o receiver aguarda o bit inicial para que a mensagem comece a ser lida, caso receba um bit inicial este avisa os outros módulos que uma mensagem está sendo lida setando 1 na porta rx_busy, enquanto e então, um a um os bits são lidos em sequência pela porta RXD, e escritos paralelamente no barramento rx_data. Quando a leitura é encerrada a porta rx_busy é setada para 0, e o receiver volta a esperar por novas mensagens.

Porta	Tipo
RXD	entrada
clk	entrada
reset	entrada
rx_data	saída

rx_busy	saída

Portas do Receiver

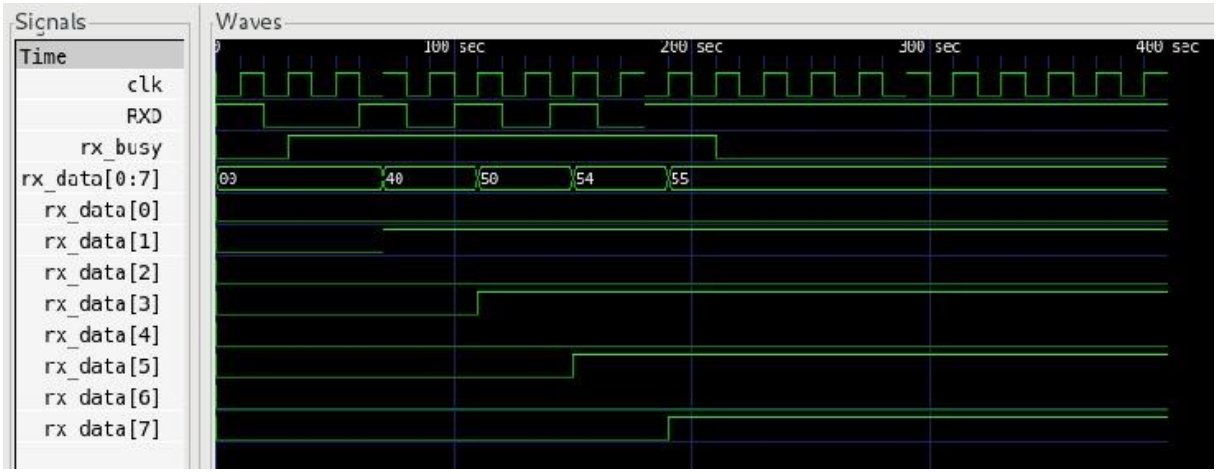
6.2. VERIFICAÇÃO

Para fazer a verificação, deve-se imaginar que existem duas UART's se comunicando, onde o receiver de uma está conectado no transmitter da outra, e vice-versa.

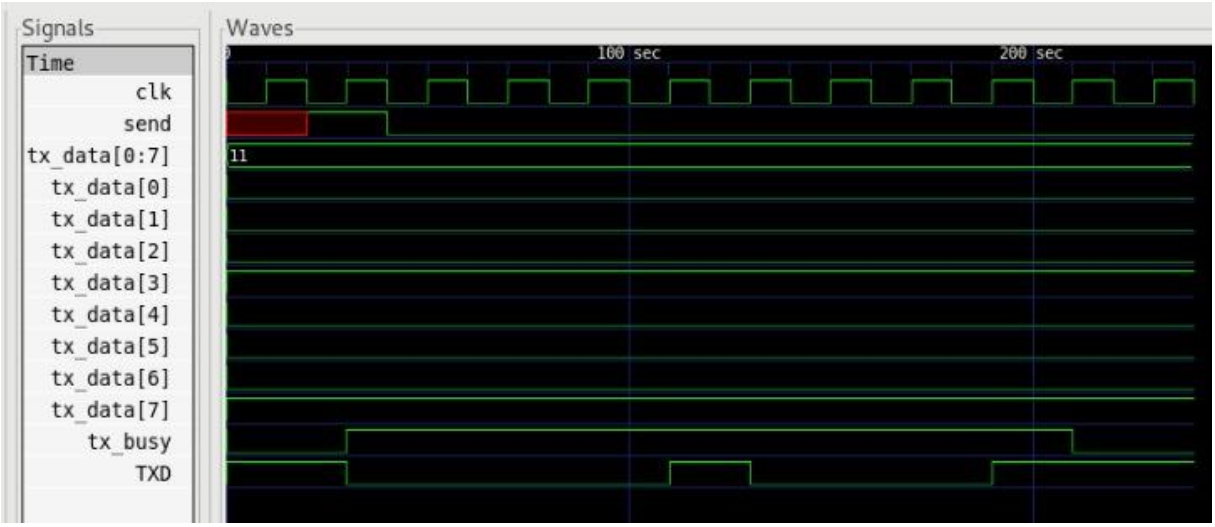
Antes que uma UART transmita realmente a mensagem, ela deve avisar a UART que estará recebendo essa mensagem, por causa de não existir clock. Enquanto ela não está enviando mensagens, apenas bit's 1 são enviados, mantendo a porta RXD da UART receptora no nível alto. A partir do momento que uma mensagem será enviada, a UART transmissora emite um bit 0, ou bit inicial, que será recebido pela porta RXD da receptora, a qual agora estará em nível lógico baixo. Após esse procedimento, a receptora irá ler a mensagem. Deste momento em diante o nível lógico da porta RXD muda de acordo com a mensagem, no entanto, ao final dos oitos bit's que compõem a mensagem, ela retornará a ser 1, indicativo de que não existe mensagem a ser lida.

A porta rx_busy trabalha em paralelo a RXD, quando o bit depois do inicial começa a ser lido, ela entra em nível lógico alto, indicando que uma mensagem está sendo lida e sendo assim a UART está "ocupada". Quando a mensagem é lida por completo, seu nível passa a ser baixo.

Com relação ao transmitter, primeiramente ele se encontra no estado de espera. As portas TXD e send se mantém altas enquanto não há mensagens a serem transmitidas. Quando aparece uma mensagem, passam a ter nível baixo, TXD enviando esse bit para o RXD, como falado anteriormente, e durante a transmissão transita entre alto e baixo, até o fim da transmissão, parando em nível alto.



Waveform: Receiver



Waveform: Transmitter

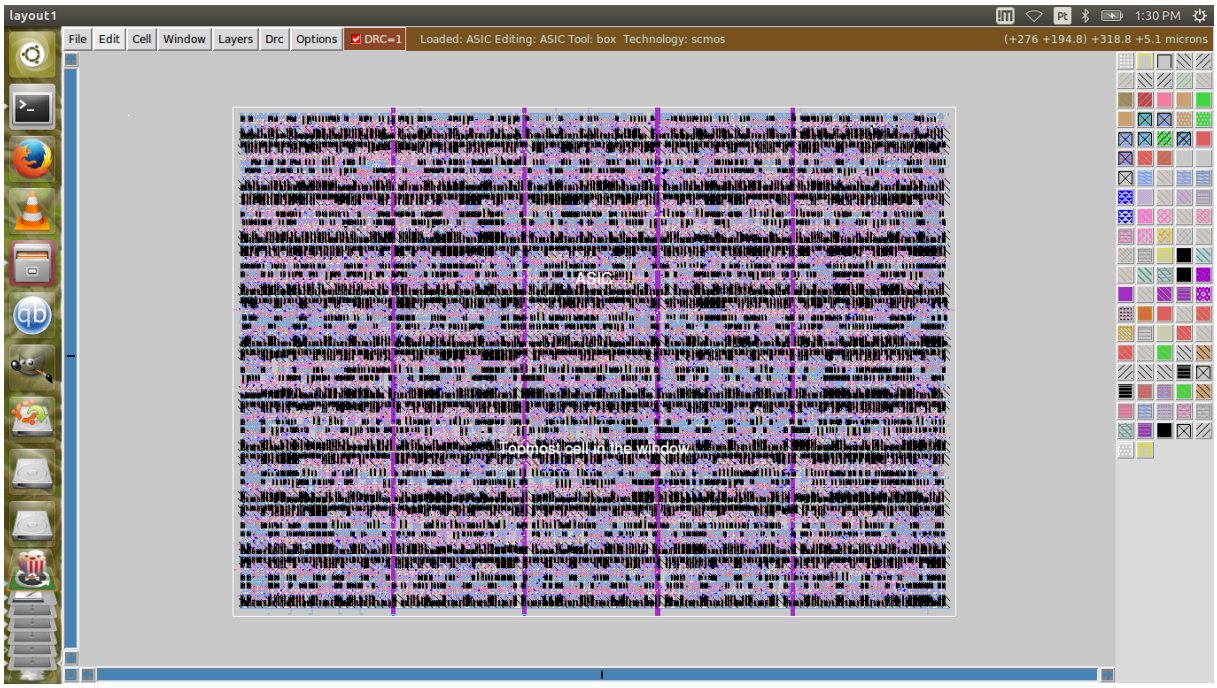
7. CONCLUSÃO

O andamento do projeto seguiu o modelo apresentado inicialmente, tendo apenas uma única mudança. Foi inserido um canal de comunicação chamado “send” entre os módulos I2C e core, que tem como função, informar ao I2C quando uma leitura de batimento cardíaco deve ser realizada, este então, permanece ativo (high) até que o dado seja processado pelo I2C, enviado, e recebido pelo core.

Após as equipes construírem os módulos de Core, UART, I2C, Clock Divider e LCD, foi realizado a união de todos estes módulos instanciando-os e realizando ligações tipo wire. Dentro de um único código chamado ASIC foram inseridos todos os módulos, incluindo o módulo de união.

Os códigos foram compilados separadamente módulo a módulo para certificação de bugs e erros e ao fim o arquivo final ASIC foi compilado, sem nenhum erro de compilação.

Todos os testes e execuções foram feitas até que o chip final foi realizado contendo todas as ligações entre e dentro dos módulos. A tecnologia utilizada para a produção foi de 180nm, e as dimensões finais do chip a ser produzido foram de 275,6x193,4um, uma área total de 0,0533 milímetros quadrados.



Layout dos transistores do chip ASIC final.

Um único erro de DRC foi obtido, onde o pino RDX ficou sem conexão. Podendo ser futuramente corrigido e testado. O projeto poderia ainda incluir um teste final de testbench com as novas entradas e saídas.

O algoritmo estará disponível em código aberto permanentemente no link público: https://github.com/tarcisiomazur/heartsenseEletronicaEngComp2019