UNIVERSIDADE FEDERAL DO PAMPA

DIONATAS LONGARETTI

IMPLEMENTAÇÃO DE UMA LUVA MICROCONTROLADA PARA A CAPTURA DE GESTOS

Alegrete, RS
2015

DIONATAS LONGARETTI

IMPLEMENTAÇÃO DE UMA LUVA MICROCONTROLADA PARA A CAPTURA DE GESTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica, Área de Concentração em Microeletrônica Digital, da Universidade Federal do Pampa (Unipampa, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia Elétrica**.

Orientador: Prof. Dr. Alessandro Girardi

Universidade Federal do Pampa – Unipampa

Curso de Engenharia Elétrica

Alegrete, RS

2015

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

L849i Longaretti, Dionatas
IMPLEMENTAÇÃO DE UMA LUVA MICROCONTROLADA PARA A CAPTURA DE GESTOS / Dionatas Longaretti.
63 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA ELÉTRICA, 2015.
"Orientação: Alessandro Gonçalvez Girardi".

1. captura de gestos. 2. luva de dados. I. Título.

Autoria: Dionatas Longaretti
Título: IMPLEMENTAÇÃO DE UMA LUVA MICROCONTROLADA PARA A CAPTAÇÃO DE GESTOS

Trabalho de Conclusão de Curso apresentado como parte das atividades para a obtenção do título de Bacharel em Engenharia Elétrica do Curso de Engenharia Elétrica da Universidade Federal do Pampa.

Os componentes da banca, abaixo listados, consideram este trabalho aprovado

	Nome	Titulação	Instituição	Assinatura
1	Alessandro Gonçalvez Girardi	Prof. Dr.	unipampa	
2	Cristian Muller	Prof. Me.	unipampa	
3	Jumar Luís Russi	Prof. Dr.	unipampa	

Data da aprovação: 2 de Dezembro de 2015.

AGRADECIMENTOS

Agradeço aos meus pais e toda a minha família pelo incentivo e apoio durante todo o meu período de graduação. Ao meu orientador professor Alessandro Girardi por compartilhar seus conhecimentos e ajudar na realização deste trabalho. Agradeço também aos meus colegas e amigos que compartilharam comigo todas as alegrias e dificuldades dessa jornada. E a todas as pessoas que deforma direta ou indireta me ajudaram a conquistar este objetivo.

"Nesta jornada terrena, aprende muito quem anda,
sempre que a alma se agranda a estrada fica pequena."
(Jayme Caetano Braun)

RESUMO

Este trabalho apresenta o desenvolvimento de uma luva para a captura de gestos. Para o protótipo da luva foram utilizados 6 sensores do tipo MPU6050, que contêm acelerômetros em 3 eixos e giroscópios em 3 eixos. Estes sensores possuem conversores internos Analógico/Digital de 16 bits que fazem a conversão dos valores instantâneos de aceleração e inclinação do sensor. Os sensores possuem faixas de precisão configuráveis e se comunicam através do protocolo de comunicação I2C. Para o controle e comunicação com os sensores foi utilizado o microcontrolador PAMPIUM, descrito em SystemVerilog. Neste mricrocontrolador foi implementada a interface de comunicação I2C, pra o envio e recebimento de dados dos sensores. Os dados capturados dos sensores foram enviados para um computador, onde puderam ser analisados. Para a comunicação do PAMPIUM com o computador foi implementada também uma interface RS232. As interfaces de comunicação foram descritas em SystemVerilog e seu funcionamento foi testado com a utilização de FPGA. Para uma fundamentação teórica, estão apresentados tópicos referentes às interfaces de comunicação utilizadas, ao microcontrolador PAMPIUM, aos sensores utilizados e também ao protótipo da luva. Este trabalho traz também de forma detalhada a implementação das interface I2C e RS232 e seus blocos de controle. A velocidade de comunicação da interface I2C ficou próxima dos 400 kbits/segundo e a velocidade máxima de comunicação da interface RS232 foi de 28800 bits/segundo. Proporcionando assim uma taxa de amostragem de 30,1 amostras por segundo. Com a utilização da luva foi possível capturar e analisar graficamente alguns gestos. Estes resultados obtidos demonstram que a luva proposta pode ser utilizada para a captura de gestos.

Palavras-chave: Interface I2C, interface RS232, MPU6050, captura de gestos, luva de dados.

ABSTRACT

This work presents the development of a glove to catch gestures. For the glove prototype we used 6 sensors MPU6050 type, which contains 3-axis accelerometers and 3-axis gyroscopes. These sensors have built-in 16-bit Analogue-to-Digital converters to make the conversion of instantaneous values of acceleration and tilt. The sensors have configurable precision tracks and communicate through I2C communication protocol. For control and communication with the sensors we used PAMPIUM microcontroller, described in SystemVerilog. An I2C communication interface was implemented for sending and receiving data from the sensors. The captured data from the sensors are sent to a computer, where they could be analyzed. For communicating with the computer we also implemented an RS232 interface. The communication interfaces have been described in SystemVerilog and its operation is tested by using FPGA. For a theoretical foundation, we presented topics related to communication interfaces to PAMPIUM microcontroller, sensors used and also the glove prototype. This work also brings in detail the implementation of I2C and RS232 interface and its control blocks. The communication speed of the I2C interface was close to 400k bit / sec and a maximum communication speed of the RS232 interface is 28800 bits / second. It provided a sampling rate of 30.1 samples per second. By using glove it was possible to capture and graphically analyze some gestures. These results demonstrate that the glove motion can be used to capture gestures.

Key-words: I2C interface, RS232 interface, MPU6050, capturing gestures, data glove.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Termos como ”computação vestível” e ”tecnologia vestível” referem-se a uma nova abordagem da computação, onde a finalidade é tornar a comunicação entre homem e máquina mais simples e intuitiva (KUMAR; RAUTARAY; AGRAWAL, 2012). Dispositivos que reconheçam automaticamente a intenção do usuário começam a se tornarem mais comuns e ganham mais espaço no mercado e nos campos da pesquisa. Esta forma de interação é chamada também de computação ubíqua, onde o usuário fica mais livre para executar tarefas que antes eram atreladas à interfaces muitas vezes fixas e projetadas para um único propósito, como: teclados, mouses e controles remoto (KIM; THANG; KIM, 2009). Podem ser considerados para a comunicação ubíqua diversas formas de interação, como por exemplo: falar, pensar, tocar, movimentar, etc.

Gestos são um componente chave na comunicação e seu reconhecimento pode ser utilizado para proporcionar novas formas de interação entre homem e máquina. Até agora, o reconhecimento de gestos é feito em sua grande maioria com base em sensores de movimento e vídeo. Reconhecimento de gestos a partir da captura de vídeos se tornou bastante popular na área de jogos, onde o computador ou videogame é capaz de interpretar os gestos executados pelo usuário e realizar determinadas ações a partir disso (JIN et al., 2011). No entanto, esta técnica possui diversas restrições por ser em sua maioria de uso interno, possuir limitação de espaço, dependência de luz e interferências com o ambiente (KIM; THANG; KIM, 2009).

O reconhecimento de gestos utilizando sensores de movimento possui algumas vantagens, como: não ser afetado pelo ambiente e ser atrelado ao usuário possibilitando maior mobilidade e abrangência (KIM; THANG; KIM, 2009). A luva para o reconhecimento de gestos, também conhecida como luva de dados é um desses dispositivos baseados em sensores de movimento como acelerômetros, giroscópios, sensores de pressão e curvatura.

1.1 Objetivos

O presente trabalho tem como objetivo principal a implementação de uma luva para captação de gestos, utilizando um microcontrolador para fazer o controle e a captação de dados de giroscópios e acelerômetros e o envio de dados para um computador.

1.2 Motivação

A relação entre homem e máquina vem se tornando cada vez mais natural e menos complexa, empregando interfaces mais intuitivas e menos perceptíveis. A implementação de uma luva de dados é um exemplo disto, onde o usuário pode por meio de gestos transmitir a intenção da execução de uma determinada atividade.

O projeto proposto parte deste princípio, da captação de gestos para a implementação de uma luva de dados, onde giroscópios e acelerômetros são incorporados à uma luva e transmitem as informações de posição e aceleração de cada dedo individualmente. Para o controle desta luva é usado o microcontrolador PAMPIUM, responsável pela configuração e captação de dados dos sensores. Estes dados são enviados para um computador onde são analisados. Para a comunicação entre os sensores e o PAMPIUM foi utilizado o protocolo de comunicação I2C e para comunicação do PAMPIUM com o computador foi utilizado o protocolo RS232. A Fig.1 mostra de maneira simplificada o projeto proposto.

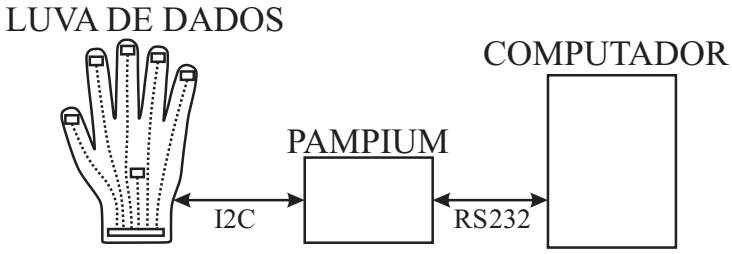


Fig. 1 – Ligação da luva de dados com o PAMPIUM e o computador.

O microcontrolador utilizado e as interfaces de comunicação foram descritos em SystemVerilog e implementados em FPGA. O modelo de FPGA utilizado foi Altera Cyclone II 2C35 (ALTERA, 2006).

O trabalho está dividido em cinco capítulos, que abordam as tapas do projeto, eles são apresentados resumidamente abaixo:

O Capítulo 2 descreve a fundamentação teórica utilizada para o desenvolvimento deste trabalho. Neste capítulo é feita uma abordagem sobre as interfaces de comunicação I2C, RS232. Também traz um detalhamento do sensor MPU6050 e do microcontrolador PAMPIUM.

O Capítulo 3 apresenta a implementação das interfaces de comunicação em FPGA e o primeiro protótipo do projeto.

No Capítulo 4 traz alguns dados coletados a partir do protótipo da luva, bem como uma análise sobre estes resultados obtidos.

Finalmente, no capítulo 5, são feitas as conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa descrever de forma detalhada os principais conteúdos utilizados no desenvolvimento deste trabalho, também traz uma explanação de alguns equipamentos utilizados.

2.1 Interface I2C

A interface I2C (Inter-Integrated Circuit, em português Circuito Inter-Integrado) é um protocolo de comunicação serial de dois fios para conectar dispositivos de baixa velocidade, como microcontroladores, conversores A/D e D/A, sensores, entre outros (INFO, 2015). A interface I2C se tornou muito popular pela sua simplicidade de utilização, sua comunicação é feita entre um mestre (dispositivo que comanda o barramento) e um escravo (dispositivo que responde a um comando enviado pelo mestre) ligados a um barramento de 2 fios (VENKATESWARAN et al., 2009). Podem ser ligados também outros dispositivos escravos ao mesmo barramento, contanto que cada dispositivo possua um endereço de acesso único. O barramento também deve possuir resistores de *pull-up* de forma a garantir que seu nível lógico será sempre alto quando o barramento não for comandado por nenhum dos dispositivos a ele ligados (PHILIPS, 2000). A Fig.2 mostra como é feita a ligação entre mestre e escravos.

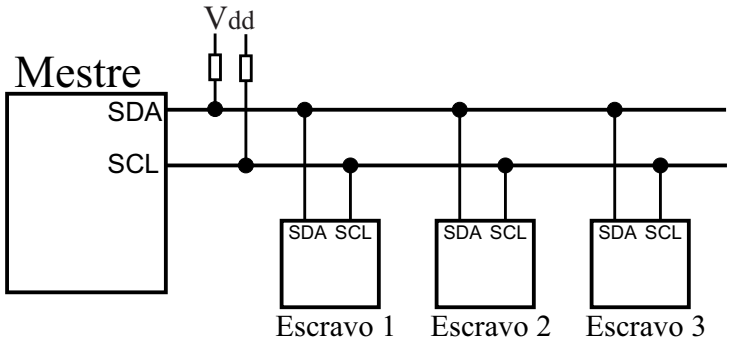


Fig. 2 – Forma de ligação entre Mestre e Escravos no barramento I2C.

A comunicação é feita através do pino SDA (serial data, em português dados em série), que é responsável pelo envio e recebimento dos dados e do pino SCL (serial clock, em português sinal de relógio serial) que é responsável pelo sinal de clock do circuito (PHILIPS, 2000).

2.1.1 Protocolo de comunicação I2C

A comunicação é feita de maneira serial em pacotes de 8 bits. Primeiramente o mestre envia ao escravo a condição de início (S), depois é enviado o endereço do escravo que é de 7 bits e em seguida um bit que indica se a próxima comunicação será de leitura ou escrita. Posteriormente um bit de reconhecimento deve ser enviado. Por fim o mestre envia

a condição de parada (P), que encerra a comunicação entre mestre e escravo (PHILIPS, 2000). A forma da comunicação pode ser vista na Fig. 3.

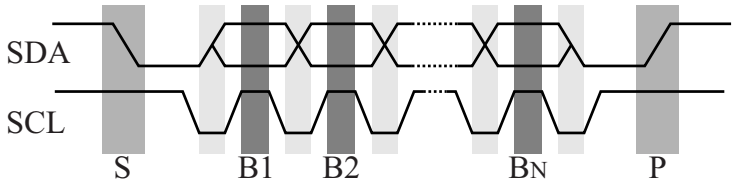


Fig. 3 – Protocolo de comunicação I2C.

O bit de reconhecimento é sempre enviado do dispositivo que está recebendo informações. O escravo envia este bit quando está sendo escrito e o mestre envia no momento em que está lendo do escravo. Este bit indica que o dispositivo terminou de receber os dados e já está pronto para receber mais ou indica que a comunicação deve ser encerrada. Sempre que o dispositivo for receber novos dados ele envia um bit de reconhecimento (ACK) que é enviado ao final da comunicação (PHILIPS, 2000). A condição para o ACK ser enviado é o barramento SCL dar um pulso enquanto o barramento SDA estiver em nível lógico baixo. Se a comunicação deve ser interrompida e estiver sendo feita uma rotina de leitura um bit de final de transmissão (NACK) deve ser enviado. A condição para este bit ser enviado é SCL dar um pulso enquanto SDA estiver em nível lógico alto. As condições de ACK e NACK podem ser vistas na Fig.4 (PHILIPS, 2000). Sempre após o envio de um NACK a condição de parada (P) é efetuada, portanto no caso do ocorrimento de um erro durante a transmissão os resistores de *pull-up* garantem que o barramento vai para nível lógico alto e a condição de NACK é efetuada.

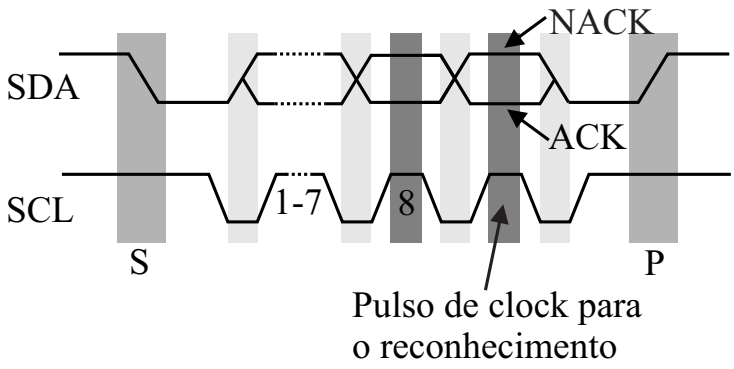


Fig. 4 – Condicões de reconhemcimento (ACK) e final de transmissão (NACK).

Como é possível verificar nas Fig. 3 e 4 a transição do barramento SDA durante a transmissão deve ser feita apenas com o barramento SCL em nível lógico baixo. Esta condição deve ser assegurada para que não ocorram condições de início e parada equivocadas durante a transmissão (VENKATESWARAN et al., 2009).

2.2 MPU 6050

O MPU 6050 é um sensor que combina um giroscópio de 3 eixos e um acelerômetro de 3 eixos em um único dispositivo. Possui 3 conversores Analógico/Digital (A/D) de 16 bits para digitalizar as saídas do giroscópio e mais 3 conversores A/D de 16 bits para digitalizar as saídas do acelerômetro. O giroscópio possui uma faixa programável de precisão que pode variar de ±250, ±500, ±1000 e ±2000 $graus/seg$. O acelerômetro também possui uma escala programável de $\pm 2G$, $\pm 4G$, $\pm 8G$ e $\pm 16G$ (INVENSENSE, 2013a). A comunicação é feita utilizando o padrão I2C em no máximo 400 kHz e opera a partir de uma faixa de alimentação que pode variar de $2,375\ V$ a $3,46\ V$. Os níveis de tensão para a comunicação são iguais aos valores de alimentação, $0\ V$ para nível lógico "0" e de $2,375\ V$ a $3,46\ V$ para nível lógico "1".

Como é possível ver na Fig.5, o dispositivo possui 8 pinos. VCC e GND são pinos de alimentação, SCL e SDA são os pinos de comunicação seguindo o protocolo I2C. O pino AD0 é usado para a alteração do endereço de acesso ao dispositivo, possibilitando colocar 2 sensores sendo controlados pelo mesmo mestre. O MPU 6050 também pode funcionar como mestre controlando um sensor externo, que pode ser de pressão, de direção de fluxo magnético, entre outros. Os pinos utilizados para o controle de dispositivos externos são XDA, SCl e INT (INVENSENSE, 2013a).

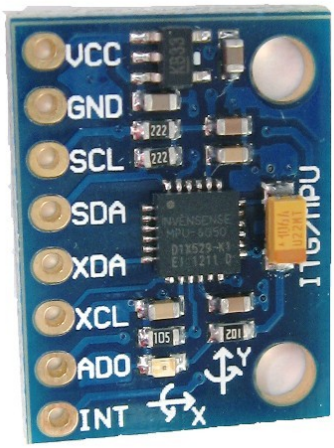


Fig. 5 – Sensor MPU 6050.

O MPU 6050 digitaliza os valores dos eixos x, y e z coletados dos giroscópios e acelerômetros e armazena em registradores internos. Também possui alguns registradores de configuração que têm a funcionalidade de alterar a escala de precisão do sensor e configurações do dispositivo (INVENSENSE, 2013b). O tamanho de cada registrador é 8 bits, sendo que a precisão dos giroscópios e acelerômetros é de 16 bits, portanto o dado coletado é dividido e armazenado em 2 registradores.

A tabela 1 mostra alguns dos registradores internos do MPU 6050. Os registradores

27 e 28 são responsáveis pela configuração da escala utilizada nos giroscópios e acelerômetros. Os registradores 59 a 64 armazenam as mais recentes medições feitas pelos acelerômetros, sendo que a parte mais significativa dos dados é armazenada em um registrador e a menos significativa em outro, os dados são armazenados em forma de complemento de 2. Os registradores 67 a 72 armazenam as mais recentes medições feitas pelos giroscópios e seus dados são armazenados de forma semelhante aos acelerômetros. O registrador 107 permite que o usuário reinicie o dispositivo com as configurações iniciais, desabilite o sensor de temperatura existente e controle a função economia de energia. Como padrão a função economia de energia está ativa quando o sensor é ligado, portanto para o início da atividade do sensor esta função deve ser desabilitada (INVENSENSE, 2013b).

Tabela 1 – Registradores do sensor MPU 6050.

End.	Nome do Reg.	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
27	GYRO_CONFIG	XG_ST	YG_ST	ZG_ST	FS_SEL [1:0]		-	-	-
28	ACCEL_CONFIG	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-	-	-
59	ACCEL_XOUT_H	ACCEL_XOUT[15:8]							
60	ACCEL_XOUT_L	ACCEL_XOUT[7:0]							
61	ACCEL_YOUT_H	ACCEL_YOUT[15:8]							
62	ACCEL_YOUT_L	ACCEL_YOUT[7:0]							
63	ACCEL_ZOUT_H	ACCEL_ZOUT[15:8]							
64	ACCEL_ZOUT_L	ACCEL_ZOUT[7:0]							
67	GYRO_XOUT_H	GYRO_XOUT[15:8]							
68	GYRO_XOUT_L	GYRO_XOUT[7:0]							
69	GYRO_YOUT_H	GYRO_YOUT[15:8]							
70	GYRO_YOUT_L	GYRO_YOUT[7:0]							
71	GYRO_ZOUT_H	GYRO_ZOUT[15:8]							
72	GYRO_ZOUT_L	GYRO_ZOUT[7:0]							
107	PWR_MGMT_1	RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

A comunicação de um dispositivo mestre com o MPU 6050 segue o padrão do protocolo I2C, sendo que uma sequência de passos deve ser respeitada para a leitura ou escrita de um registrador interno do sensor. A Fig. 6 mostra a sequência a ser seguida para uma rotina de escrita no registrador. Primeiramente o mestre envia a condição de início (S) e em seguida os 7 bits do endereço do sensor (AD) e o bit de escrita (W). O endereço padrão do MPU 6050 é 1101000 quando o pino AD0 é em nível lógico baixo ou 1101001 quando AD0 é em nível lógico alto e o bit que representa a escrita no sensor é 0 (INVENSENSE, 2013a). Após o envio, o escravo deve responder com um bit de reconhecimento (ACK) e em seguida o mestre envia o endereço do registrador interno (RA) que deve ser escrito. Em

seguida, um novo bit de reconhecimento é enviado. Depois de dizer qual o registrador, então o dado a ser escrito (DATA) é enviado e o escravo envia mais um bit de reconhecimento. Esta sequência de envio de dados é interrompida quando o mestre envia a condição de parada (P).

Mestre	S	AD+W		RA		DATA		DATA		P
Escravo			ACK		ACK		ACK		ACK	

Fig. 6 – Sequência de escrita no MPU 6050.

A rotina de leitura de um registrador é semelhante à de escrita, a Fig.7 mostra a sequência de passos que devem ser seguidos. Assim como a rotina de escrita, a leitura inicia-se com o envio de um bit de início, seguido do endereço do MPU 6050 e o bit de escrita, depois o escravo responde com um bit de reconhecimento e o endereço do registrador a ser lido é enviado e um novo bit de reconhecimento é enviado do escravo. A seguir o mestre envia um novo bit de início, depois envia o endereço do escravo seguido do bit de leitura (R), que agora deve ser 1, o escravo responde com um bit de reconhecimento e logo após começa a enviar os dados gravados no registrador requerido. Ao término de cada envio de dados o mestre responde com um bit de reconhecimento e quando deseja-se terminar a comunicação o mestre envia um bit de fim de comunicação (NACK) e logo após a condição de parada.

Mestre	S	AD+W		RA		S	AD+R			ACK		NACK	P
Escravo			ACK		ACK			ACK	DATA		DATA		

Fig. 7 – Sequência de leitura do MPU 6050.

2.3 Interface RS-232

A interface RS-232 é um padrão para comunicação serial de dados. Este padrão de comunicação foi utilizado pela primeira vez no ano de 1962 e desde então se popularizou muito, principalmente pela sua funcionalidade e praticidade. O padrão RS-232 é comumente usado em portas serias de computadores(AXELSON, 2007).

Este protocolo utiliza sinais de tensão negativos para nível lógico "1"e positivos para nível lógico "0", variando de $\pm 3V$ a $\pm 25V$, dependendo da aplicação. Os canais de comunicação são separados, tornando possível o envio e recebimento de dados paralelamente (HOROWITZ; HILL, 1989). A Fig.8 mostra como é feita a comunicação entre dispositivos.

Fig. 8 – Protocolo de comunicação RS-232.

Para a condição de início (S), o barramento deve mudar de nível lógico alto para baixo, em seguida são enviados 8 bits de dados e para a condição de parada o barramento

deve se manter em nível lógico alto. O último bit transmitido pode ser utilizado também como bit de paridade. Sempre que a soma dos dados enviados for par o bit de paridade é "0", caso for ímpar este bit é "1". A utilização do controle de paridade aumenta a confiabilidade do dado recebido (AXELSON, 2007).

A comunicação é feita em velocidades padrões (*Baud Rate*) que variam de 1200, 2400, 4800, 9600, etc. bits/s. Portanto os dois dispositivos devem estar operando na mesma velocidade de transmissão e recebimento para que a comunicação opere de maneira correta. Para determinar a velocidade de comunicação o dispositivo receptor deve analisar o primeiro byte enviado, que deve ser padronizado, calculando a sua velocidade (INSTRUMENTS, 1999). Para conectores o padrão especifica dois tipos: DB-9 e DB-25, de nove e vinte e cinco pinos, respectivamente. O padrão mais utilizado é o DB-9, a Fig.9 mostra a pinagem deste conector.

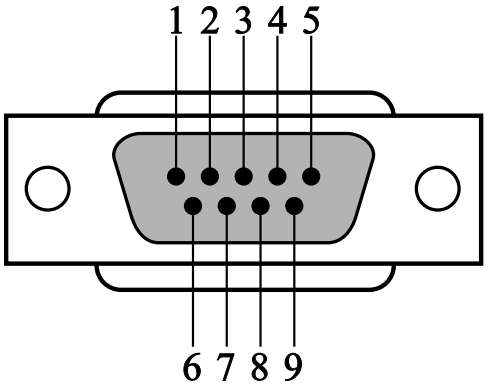


Fig. 9 – Conector DB-9.

Os sinais correspondentes de cada pino são:
1 - Detector de transmissão de dados.
2 - Receptor de dados (Rx).
3 - Transmissor de dados (Tx).
4 - Terminal de dados prontos.
5 - Sinal de terra (GND).
6 - Conjunto de dados pronto.
7 - Solicitar envio.
8 - Livre para envio.
9 - Indicador de anel.

Os pinos que devem ser evidenciados são: Rx (pino 2) que é responsável pelo recebimento de dados, Tx (pino 3) responsável pelo envio de dados e GND (pino 5) que é utilizado como referência. Utilizando somente estes três pinos é possível criar uma interface simples de comunicação entre dois dispositivos (INSTRUMENTS, 1999).

2.4 PAMPIUM

O PAMPIUM é um microcontrolador reconfigurável desenvolvido pelo Grupo de Arquitetura de Computadores e Microeletrônica (GAMA) da UNIPAMPA, campus Alegrete. Este projeto foi concebido através de uma arquitetura em SystemVerilog e trata-se de um microcontrolador de 16 bits RISC para uso geral (ENGROFF, 2014).

Esta arquitetura é composta por uma memória de programa, uma memória de dados, dois bancos de registradores, uma unidade lógica aritmética (ULA) e uma unidade de controle. A comunicação externa é feita utilizando duas portas do tipo IN/OUT. A arquitetura organizacional básica está expressa na Fig. 10.

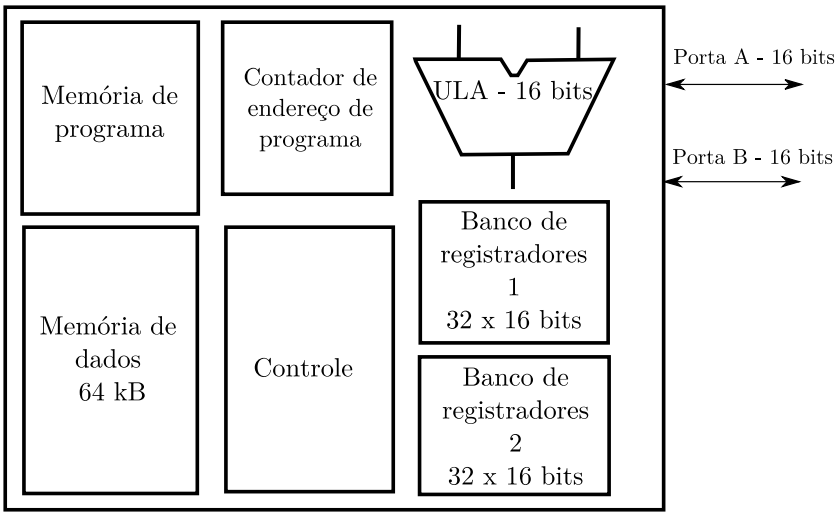


Fig. 10 – Estrutura organizacional básica do microcontrolador (ENGROFF, 2014).

A direcionalidade de cada pino de comunicação é definida por um bit específico contido no banco de registradores e o dado lido ou escrito na porta também está contido em um registrador específico. Todas as operações aritméticas e lógicas são realizadas entre registradores e seus resultados são salvos em registradores (ENGROFF, 2014).

O banco de registradores principal é composto por dois bancos secundários. Chamados $R0$ e $R1$, com 32 registradores cada, sendo que cada registrador possui 16 bits. Para acesso a um determinado registrador é necessário informar seu endereço e em qual banco está localizado. Alguns registradores possuem funcionalidades específicas para determinadas instruções ou configurações de portas.

Todas as instruções implementadas são justificadas e descritas em (ENGROFF, 2014). No total são contabilizadas 32 instruções, que podem ser vistas na tabela 2.

Tabela 2 – Conjutno de instruções do PAMPIUM.

OPCODE	Mnemônico	Descrição curta
000000b	**NOP**	Não faz nada
000001b	**END**	Paralisa programa
000010b	**COPY**	Cópia de valores entre registradores
000011b	**MOVL**	Atribui literal para *#REG_L*
000100b	**JUMP**	Pula instruções
000101b	**CALL**	Pula instruções e salva endereço de retorno
000110b	**RET**	Retorno para instrução
000111b	**RETL**	Retorno para instrução salvando literal
001000b	**RM**	Lê da memória de dados
001001b	**WM**	Escreve na memória de dados
001010b	**BL**	Pula se for maior
001011b	**BS**	Pula se for menor
001100b	**BLE**	Pula se for maior ou igual
001101b	**BSE**	Pula se for menor ou igual
001110b	**BE**	Pula se for igual
001111b	**BNE**	Pula se foi diferente
010000b	**BBCLEAR**	Pula se o bit for igual ”0”
010001b	**BBSET**	Pula se o bit for igual ”1”
010010b	**SHR**	Deslocamento a direita
010011b	**SHL**	Deslocamento a esqueda
010100b	**BSET**	Determina o bit como ”1”
010101b	**BCLEAR**	Determina o bit como ”0”
010110b	**MUL**	Multiplica o valor dos registradores
010111b	**DIV**	Divide o valor dos registradores
011000b	**REM**	Retorna o resto da divisão de dois registradores
011001b	**ADD**	Soma o valor dos registradores
011010b	**SUB**	Subtrai o valor dos registradores
011011b	**OR**	Faz a operação ”OU”lógica
011100b	**AND**	Faz a operação ”E”lógica
011101b	**XOR**	Faz a operação ”OU”exclusiva lógica
011110b	**NOT**	Faz a operação ”NOT”lógica
011111b	**ADDPC**	Pula instruções indicada por registrador

3 IMPLEMENTAÇÃO DAS INTERFACES E PROTÓTIPO

Para a confecção da luva de captação de gestos foram utilizados 6 sensores de movimento do tipo MPU 6050. Estes sensores foram alocados nas extremidades dos dedos e na parte superior da mão. Para a captação de dados dos sensores foram implementadas em SystemVerilog as interfaces de comunicação I2C e RS232. Estas interfaces foram inseridas no microcontrolador PAMPIUM e ligadas diretamente em seu banco de registradores. O PAMPIUM é então, responsável pela configuração e captação de dados dos sensores e pelo envio de dados ao computador.

Para a comunicação com os sensores foi implementada a interface I2C, visto que o sensor utiliza este protocolo de comunicação. E para a comunicação com o computador foi implementada a interface RS232. A Fig.11 mostra de forma simplificada como são feitas as ligações entre o PAMPIUM e as interfaces.

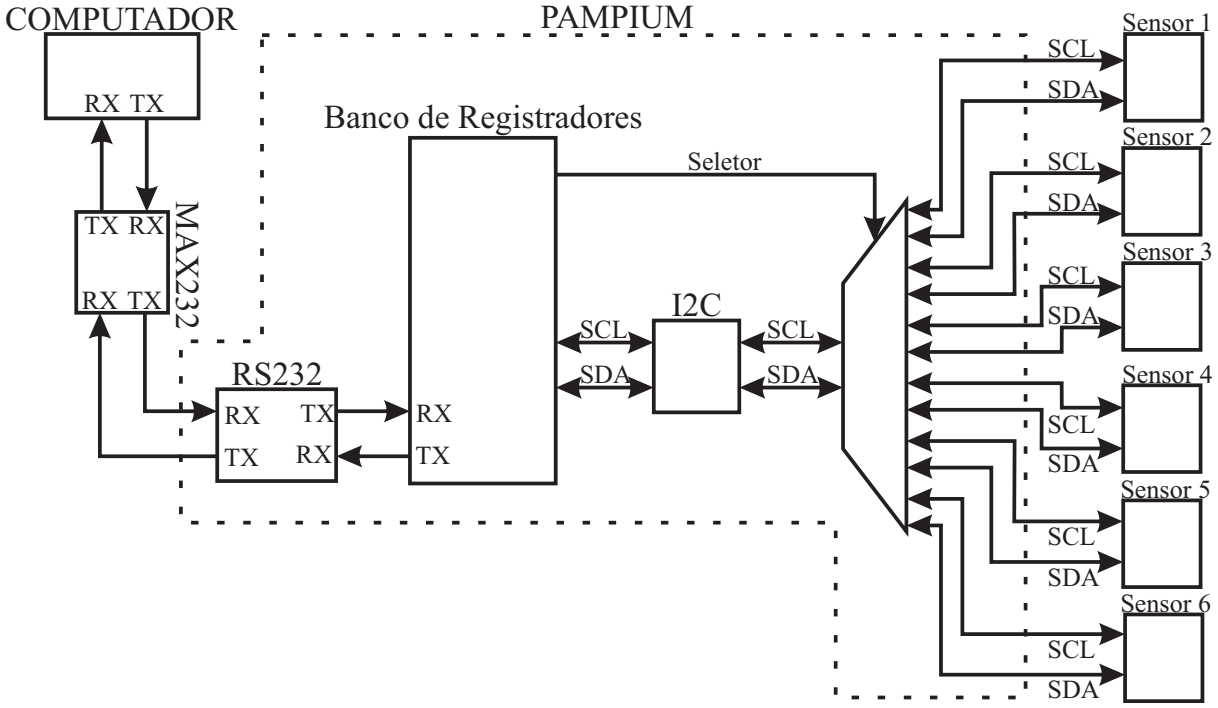


Fig. 11 – Diagrama de blocos das interfaces de comunicação.

Visto que foram utilizados 6 sensores para captação de dados, na interface de comunicação I2C utilizou-se um demultiplexador. Sua finalidade é permitir que controlando seu seletor o microcontrolador se comunique com todos os sensores.

Devido ao fato de todas as operações do PAMPIUM serem realizadas entre registradores, as interfaces de comunicação foram ligadas diretamente ao seu banco de registradores, proporcionando maior facilidade no controle das interfaces tanto para envio quanto para recebimento de dados.

O controle das interfaces foi feito pelo microcontrolador PAMPIUM, através da

sua programação em Assembly, utilizando seu próprio conjunto de instruções. O algoritmo utilizado pode ser visualizado no Apêndice A.

3.1 Implementação da interface I2C

A interface de comunicação I2C foi implementada de maneira que seu controle pudesse ser feito diretamente pelo banco de registradores do PAMPIUM. Foram alocados então alguns bits de um registrador específico para isto. A interface de comunicação tem 2 pinos de saída (**SCL** e **SDA**) que através do demultiplexador são ligados a todos os sensores da luva. A Fig.12 mostra o bloco de controle da interface e os pinos utilizados para configuração e comunicação.

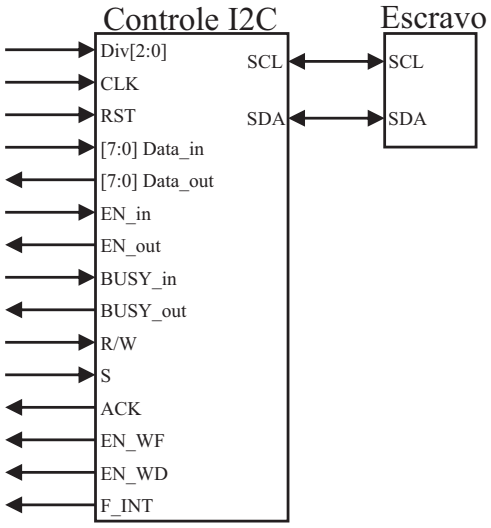


Fig. 12 – Bloco de controle da interface I2C.

A descrição de cada pino desta interface, é apresentada abaixo:

• **SLC** e **SDA** são pinos ligados diretamente com os pinos de saída do PAMPIUM e possuem acesso externo. São os pinos de comunicação da interface I2C e fazem a ligação com os sensores da luva;

• **Div** é um pino de configuração, responsável pela divisão interna do sinal de clock recebido. Sua finalidade é adequar a velocidade de comunicação do bloco de controle com o bloco escravo. São 3 bits de controle, o que resulta em 8 diferentes divisões do clock de entrada. As possíveis entradas e o valor pelo qual o clock será dividido estão expressos na tabela 3. Seu valor pode ser setado diretamente no registrador 0 do banco de registradores 1 ($[\#0]R1$) nos bits de 5 a 3;

• **Data_in** e **Data_out** são pinos de comunicação da interface com o microcontrolador. É por onde passam os dados à serem escritos no dispositivo escravo e também os dados lidos do escravo. A comunicação é feita em pacotes de 8 bits e o registrador onde

Tabela 3 – Divisões de clock da interface I2C.

Entrada	Valor da divisão
000b	0
001b	2
010b	4
011b	8
100b	16
101b	32
110b	64
111b	128

estes dados ficam alocados é o registrador 4 do banco de registradores 0 ($[\#4])R0$), nos bits de 7 a 0;

• **EN_in** e **EN_out** são pinos responsáveis pela habilitação da interface de comunicação I2C. Estes pinos estão alocados no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 0. Quando o nível lógico deste bit for "1"a interface está ativa;

• **BUSY_in** e **BUSY_out** é um pino de controle que indica quando deve ser iniciada uma rotina de escrita ou leitura com a interface I2C. Seu bit de controle está alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 1;

• **R/W** é o pino que indica se a ação executada será de escrita ou leitura no dispositivo escravo. Este pino está alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 2;

• **S** é um pino faz o controle da condição de início (S). Nas rotinas de leitura é necessária uma segunda condição de início, para que isto ocorra o nível lógico deste pino de ve ser "1". Este pino está alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 3;

• **ACK** é o pino que indica se o dispositivo escravo enviou a ação de reconhecimento (ACK). Este pino está alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 4;

• **EN_WF** é o pino que habilita a escrita da interface de comunicação no banco de registradores;

• **EN_WD** é o pino que habilita a escrita de dados vindos da interface de comunicação no banco de registradores;

• **CLK** é o pino de entrada do sinal de clock, que é dividido e adequado à velocidade de comunicação da interface;

• **RST** é o pino de entrada do sinal de reset, que limpa todos os dados dos

registradores e estabelece a interface de comunicação para sua configuração inicial;

• **F_INT** é um pino que indica quando a interface terminou de executar uma rotina e está pronta para receber um novo comando. Este pino está alocado no registrador 0 do banco de registradores 1 ($[\#0]R1$), no bit 2.

Para a ligação do PAMPIUM com os sensores foram utilizados pinos de conexão externos existentes na FPGA. Esta ligação pode ser feita de maneira direta, pois os níveis de tensão gerados pelos pinos da FPGA são iguais aos níveis requeridos pelo sensor MPU6050, $0V$ para nível lógico "0" e $3,3V$ para nível lógico "1" (ALTERA, 2006).

A Fig.13 mostra o fluxo de operação da interface I2C.

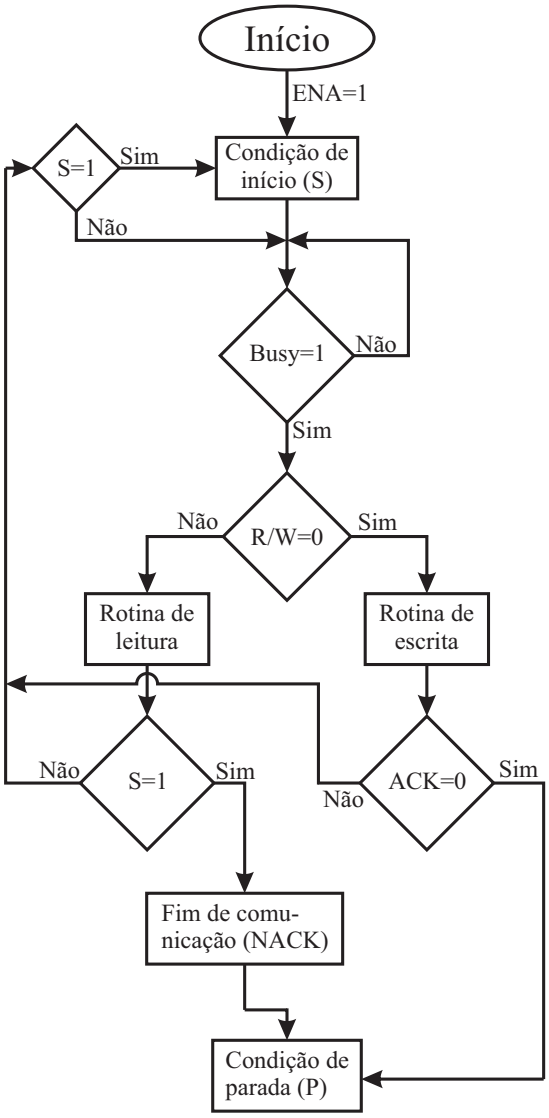


Fig. 13 – Fluxograma de operação da interface I2C.

Acompanhando o fluxograma é possível identificar como são feitas as rotinas de escrita e leitura no sensor. O início é dado quando o **ENA** vai pra nível lógico "1", logo em seguida a condição de início (S) é enviada pela interface que após isso entra em um laço esperando o Busy ir para nível lógico "1". A seguir é feita a análise do bit **R/W**,

que indica se a próxima ação é de escrita ou leitura. Caso R/W=0, a interface executará uma rotina de escrita e fará a verificação do bit de reconhecimento (ACK). Caso igual a 0 a condição de parada (P) é executada. No caso ACK=1 então é feita uma segunda verificação, desta vez do S, que indica se uma nova condição de início deve ser executada. No caso de R/W=1, então a interface executa a rotina de leitura e em seguida faz uma verificação do S. Se S=1 é executada a condição de fim de transmissão (NACK) e em seguida a condição de parada. Para S=0 a interface retorna para o laço onde fica esperando Busy=1 para executar uma nova ação. Este ciclo é repetido até completar a rotina de escrita ou leitura de um dado no sensor. No Apêndice B é possível visualizar a descrição em SystemVerilog da interface I2C.

3.2 Implementação da interface RS232

A interface RS232 foi implementada de maneira similar à interface I2C, onde todos os pinos de controle foram ligados diretamente ao banco de registradores. Foram também alocados alguns bits do banco de registrador para o controle desta interface. Foram utilizados apenas 2 pinos pra comunicação **RX** e **TX**; a pinagem da interface implementada pode ser vista na Fig.14.

O sinal gerado pelo microcontrolador é de 0 V para nível lógico "0" e 5 V para nível lógico "1", mas os níveis de comunicação da interface RS232 variam de -12 V a 12 V. Portanto, foi utilizado um conversor de níveis de tensão, que é o MAX232 para fazer a adequação dos sinais de entrada e saída (INTRUMENTS, 2006) . O MAX232 eleva a tensão para a transmissão e reduz a tensão no recebimento de dados. A Fig.14 mostra como é feita a ligação da interface de RS232 com o conversor de tensão MAX232.

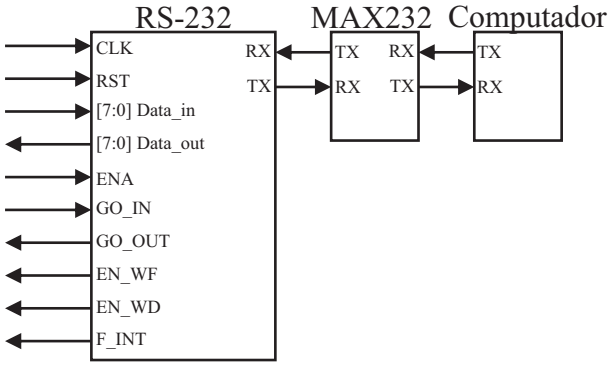


Fig. 14 – Bloco de controle da interface RS232.

A descrição de cada pino desta interface, é apresentada abaixo:

• **CLK** é o pino de entrada do sinal de clock no circuito; este sinal é utilizado para controlar a velocidade de recebimento e transmissão de dados;

• **RST** é o pino de entrada do sinal de reset que limpa todos os dados dos registradores e estabelece a interface de comunicação para sua configuração inicial;

• **Data_in** é o pino de transferência de dados do microcontrolador para a interface de comunicação. Os dados que devem ser transmitidos são enviados através deste pino, que está alocado no registrador 16 do banco de registradores 0 ($[\#16]R0$), nos bits de 7 à 0;

• **Data_out** é o pino que faz a ligação do recebimento de dados da interface de comunicação com o microcontrolador. Este pino está alocado no registrador 17 do banco de registradores 0 ($[\#16]R0$), nos bits de 7 a 0;

• **EN** é o pino responsável pela habilitação da interface de comunicação RS232. Este pino é alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 5. Quando nível lógico deste bit for "1" a interface está ativa;

• **GO_IN** e **GO_OUT** são pinos responsáveis pelo comando de envio de dados. Quando este pino é ativado a interface envia o dado presente no registrador 16 do banco de registradores 0. Este pino está alocado no registrador 0 do banco de registradores 0 ($[\#0]R0$), no bit 6;

• **EN_WF** é o pino que habilita a escrita da interface de comunicação no banco de registradores.

• **EN_WD** é o pino que habilita a escrita de dados vindos da interface de comunicação no banco de registradores.

• **F_INT** é um pino que indica quando a interface terminou de executar uma rotina de envio ou recebimento de dados e está pronta para receber um novo comando. Este pino está alocado no registrador 0 do banco de registradores 1 ($[\#0]R1$), no bit 6.

A Fig.15 mostra de maneira simplificada como é a operação da interface RS232.

Acompanhando o fluxograma é possível identificar as principais etapas na operação da interface RS232. A interface inicia quando o **ENA** for para nível lógico 1 e em seguida faz o ajuste da velocidade de transmissão (autobaud), esperando receber um byte do dispositivo a ela ligado. Este byte foi padronizado como sendo **'10101010'b**, a interface controla o tempo de recebimento deste byte e divide o tempo por 8 obtendo a velocidade de transmissão. Após o ajuste da velocidade uma rotina de testes é iniciada monitorando se a interface vai receber ou enviar o próximo dado. Primeiramente é feita a verificação do pino **RX**, se ele for pra nível lógico "0" inicia-se uma rotina de leitura de dados e ao final retorna-se a etapa de verificações. Se o pino **RX** se manter em nível lógico "1", então é feito o teste do pino **GO_IN** se ele for pra nível lógico "1" é iniciada uma rotina de escrita, ao final desta rotina o pino **GO_IN** é zerado novamente e retorna-se para a etapa de verificações. Caso o pino **GO_IN** se mantiver em nível lógico "0", a interface permanece em um laço de verificação. No Apêndice C é possível visualizar a descrição em SystemVerilog da interface RS232.

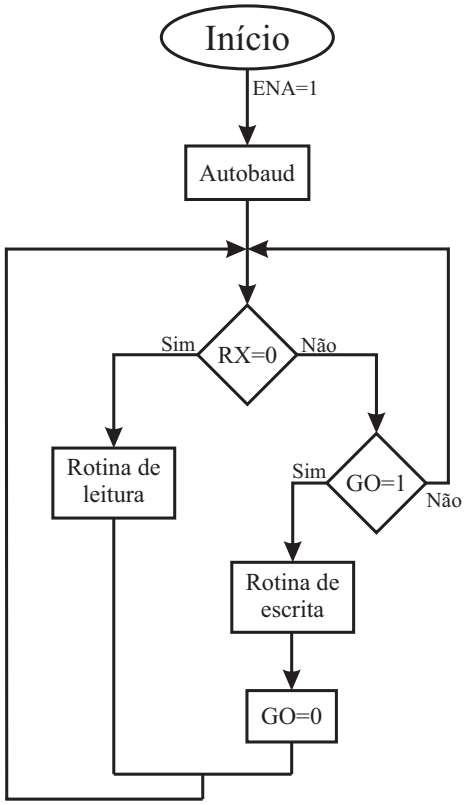


Fig. 15 – Fluxograma de operação da interface RS232.

3.3 Protótipo

O primeiro protótipo da luva para a captura de movimentos foi implementado. Para sua confecção foi utilizada uma luva de material não condutor, 6 sensores do tipo MPU6050, fios e conexões. Os sensores foram costurados à luva na extremidade de cada dedo e na parte superior da mão. Para conectar cada sensor foram necessários 4 fios, 2 para alimentação (VDD e GND) e 2 para comunicação, seguindo o protocolo I2C (SCL e SDA). A Fig.16 mostra a parte superior da luva com os sensores e as ligações.

Para comunicação com a FPGA foram utilizados pares de fios trançados, onde um fio é responsável pela comunicação e o outro é ligado ao terra (GND). Esta forma de ligação diminui as interferências e permite que o circuito opere a grandes frequências. No caso da luva foram utilizados 6 sensores com 2 fios responsáveis pela comunicação de cada sensor. Portanto, nesta ligação foram utilizados 12 pares de fios de comunicação. A Fig.17 mostra com mais detalhes como foi feita esta ligação. Para a comunicação da FPGA com o computador foi utilizado um cabo com conectores padrão DB-9 de 9 pinos, que é o cabo comumente utilizado no protocolo de comunicação RS232.

Fig. 16 – Protótipo da luva de captura de dados.

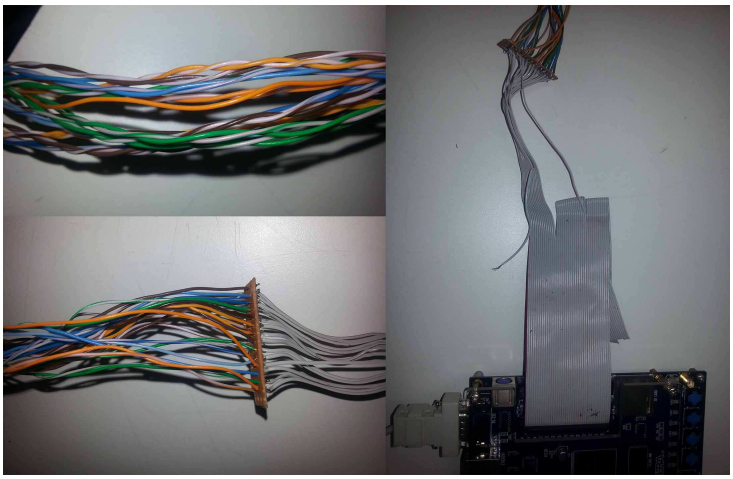


Fig. 17 – Detalhes da ligação da luva com a FPGA.

3.4 Interface em Matlab

Em Matlab foi criada uma interface para o controle de recebimento de dados. Nesta interface é possível inserir dados pessoais do indivíduo que executará o movimento, dados como: nome, idade, sexo e mão que possui maior habilidade. Dados referentes ao movimento e data também podem ser salvos.

Para a captura de um movimento deve ser estabelecido um tempo de execução e no momento que o usuário ativar o botão de início, a luva começa a enviar os dados referentes à posição e aceleração de cada sensor. Estes dados são armazenados em variáveis internas e ao final existe a possibilidade de salvar todos os dados captados em um arquivo. Os dados

são plotados em tempo real nos campos "Acelerômetros"e "Giroscópios", sendo possível assim acompanhar a captura de dados. A Fig.18 mostra a interface criada em Matlab.

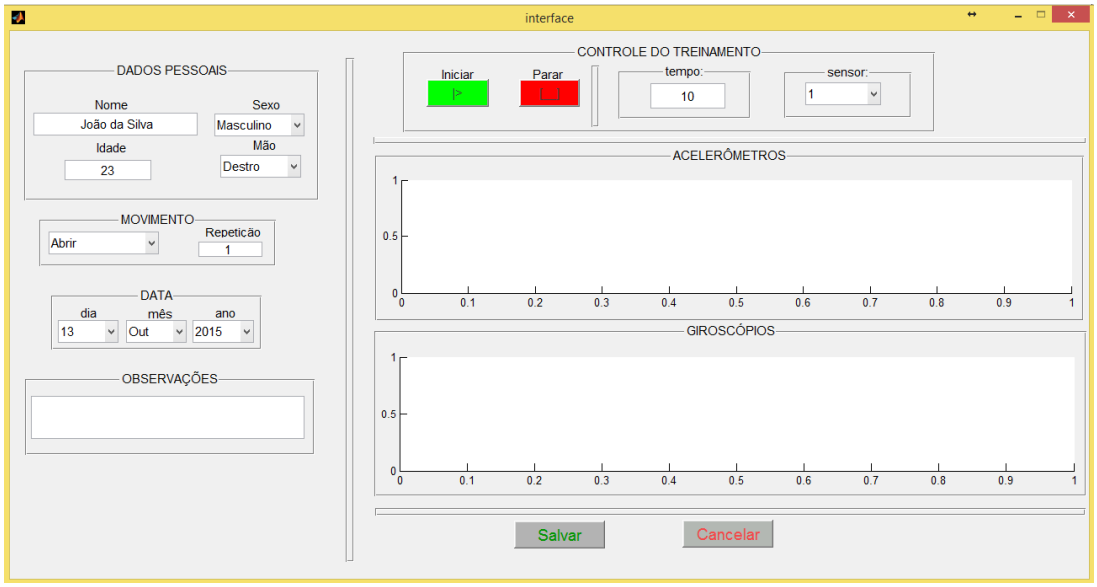


Fig. 18 – Interface de controle de dados em Matlab.

Um algoritmo de carregamento e leitura de dados também foi implementado, para que se consiga analisar cada sensor individualmente e também em conjunto. Com estas funcionalidades esta interface estabelece a possibilidade da criação de um banco de dados com diversos movimentos, onde diferentes usuários repitam um determinado movimento. Desta forma será possível comparar as diferenças entre pessoas na execução de um mesmo movimento.

4 RESULTADOS OBTIDOS

Concluídas as etapas de implementação das interfaces I2C, RS232 e com o protótipo pronto, inicia a fase de captura e análise de dados. As escalas de precisão dos sensores foram todas configuradas igualmente para que os dados captados tivessem na mesma proporção. A escala configurada nos giroscópios foi de $\pm 250\ graus/seg$ e nos acelerômetros foi configurada a aceleração máxima de $2G$.

Para facilitar a visualização dos dados foi estabelecida uma numeração para identificar cada sensor individualmente. Esta numeração pode ser vista na Fig.19.

Fig. 19 – Luva com numeração de cada sensor.

Os números definidos foram:
1-Polegar;
2-Indicador;
3-Médio.
4-Parte superior da mão
5-Anelar;
6- Mínimo.

4.1 Captura de gestos

Para captura de dados foram executados 2 gestos distintos. O primeiro foi de erguer e baixar a mão. Durante este gesto toda a mão se movimenta de maneira igual, primeiro para cima e depois para baixo. A Fig. 20 mostra cada etapa de sua execução.

Este gesto foi analisado com o intuito de verificar se todos os 6 sensores respondem de maneira igual perante um movimento uniforme. Na Fig. 22 é possível observar o

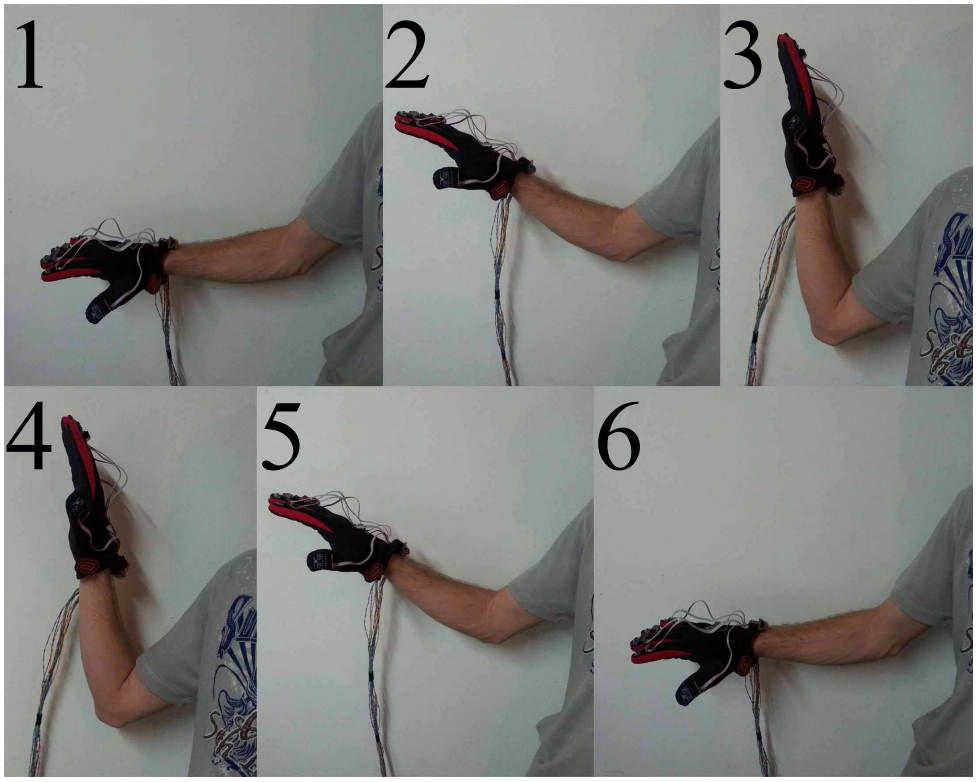


Fig. 20 – Movimento de erguer e baixar a mão, realizado para captação de dados.

comportamento dos acelerômetros nos 3 eixos, em cada sensor presente na luva. A Fig.23 mostra o comportamento dos giroscópios, também nos 3 eixos e em todos sensores da luva.

O segundo gesto executado foi de fechar e abrir a mão. Durante este gesto a mão se manteve parada e apenas os dedos se movimentaram, primeiro fechando a mão e depois abrindo. A Fig.21 mostra as etapas da execução deste movimento.

Fig. 21 – Movimento de fechar e abrir amão, realizado para captação de dados.

Na Fig.24 é possível verificar o comportamento dos 6 acelerômetros presentes na luva nos 3 eixos. Na Fig.25 é possível verificar o comportamento dos 6 giroscópios presentes na luva nos 3 eixos.

4.2 Análise de resultados

Analisando os gráficos gerados pela captura de dados dos acelerômetros e giroscópios pode-se verificar facilmente as etapas da execução do gesto. No gesto de erguer e baixar a

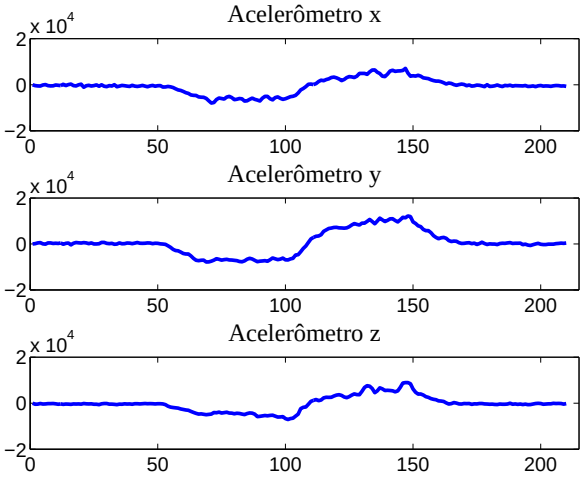
mão representado nas Fig. 22 e 23 pôde ser notado um comportamento uniforme de todos os sensores, visto que foi executado um gesto onde toda a mão se moveu igualmente. No segundo gesto analisado, que foi de fechar e abrir a mão, representado nas Fig.24 e 24 pôde-se notar um movimento bastante similar nos sensores alocados nos dedos, no entanto o sensor presente nas parte superior da mão mostrado nas Fig.24d e 25d permaneceu praticamente constante.

Utilizando o sensor alocado na parte superior da mão é possível então, identificar quando é executado um movimento envolvendo toda a mão ou somente os dedos. Podemos utilizar este sensor como referência, onde os dados gerados por este sensor podem ser utilizados para filtrar movimentos realizados em todos os sensores de maneira uniforme

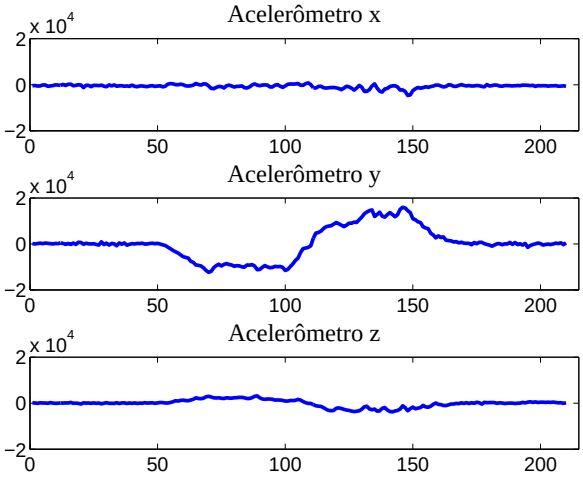
O movimento de erguer e baixar a mão foi executado em aproximadamente 6,97 segundos e possui um total de 210 amostras. O movimento de abrir e fechar a mão foi executado em 5,21 segundos e possui um total de 157 amostras. A taxa de amostragem é então, de 30,1 amostras por segundo.

A velocidade de transmissão da interface I2C é em torno de 400K bits por segundo e a taxa de transmissão da interface RS232 ficou em no máximo 28800 bits por segundo.

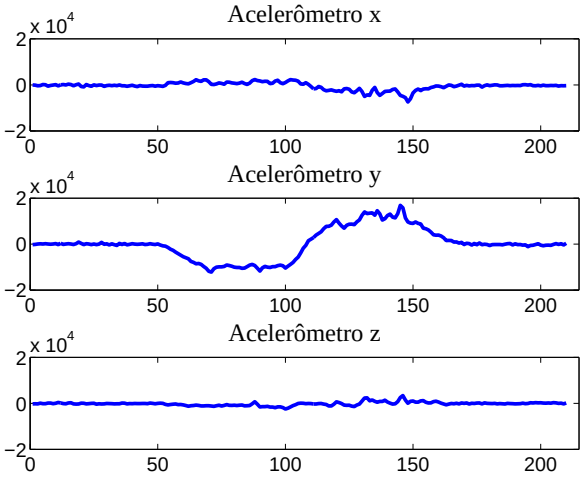
Analisando visualmente os gráficos gerados pelos sensores, pode-se dizer que a precisão alcançada é suficiente para capturar gestos. Desde que não sejam demasiadamente rápidos, pois cada etapa da execução do gesto foi facilmente identificável nos gráficos. Uma análise computacional poderá informar se com esta precisão é possível também identificar os gestos.



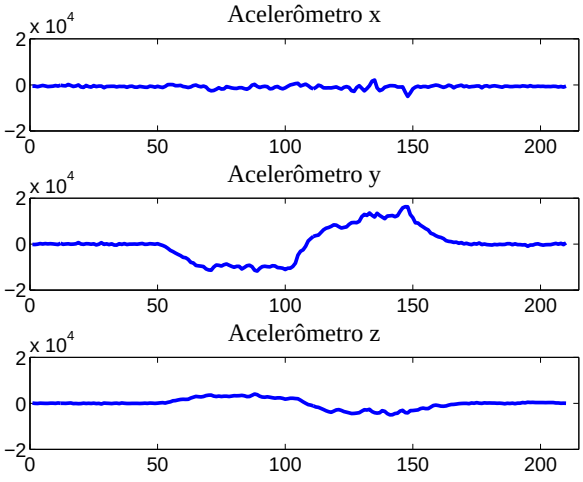
(a) Acelerômetros do sensor 1.



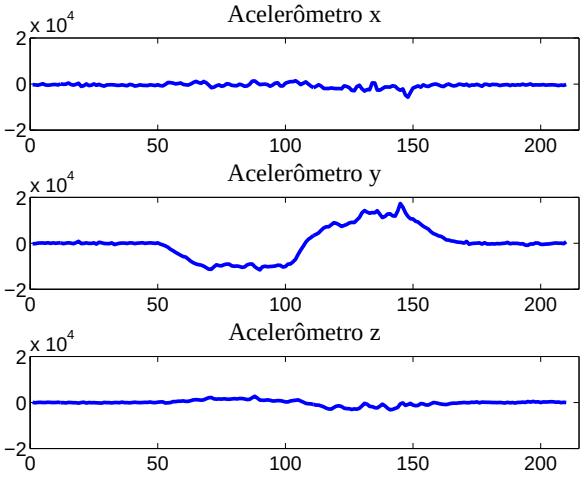
(b) Acelerômetros do sensor 2.



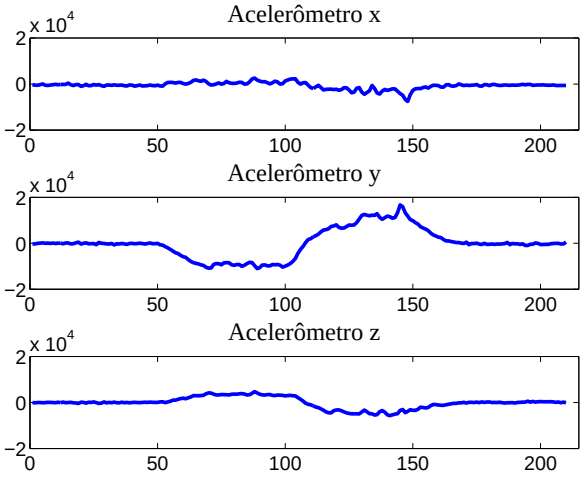
(c) Acelerômetros do sensor 3.



(d) Acelerômetros do sensor 4.

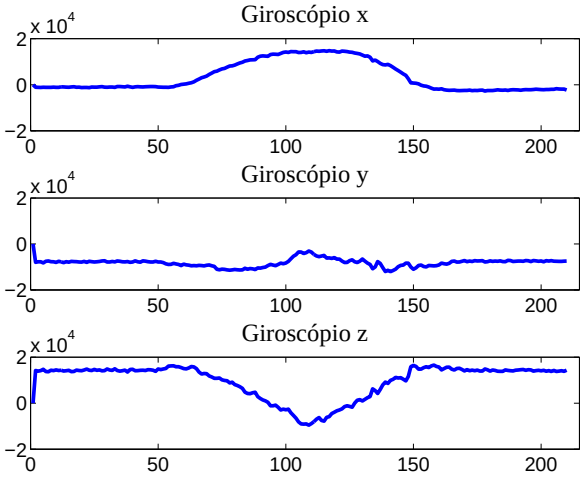


(e) Acelerômetros do sensor 5.

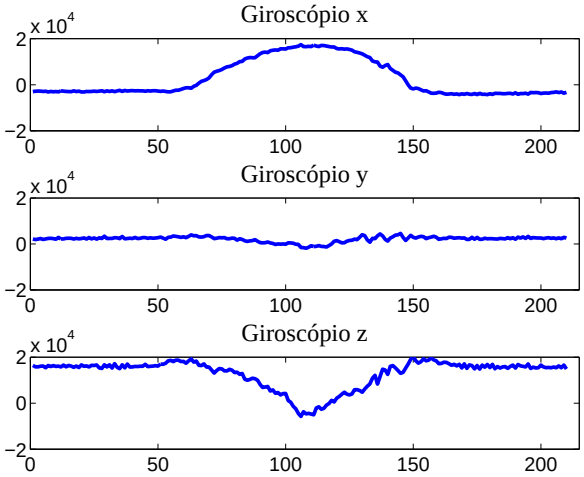


(f) Acelerômetros do sensor 6.

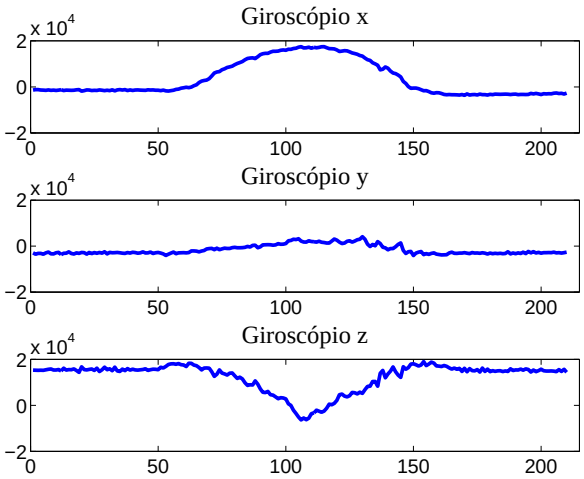
Figura 22 – Gráficos gerados pelos acelerômetros durante o movimento de erguer e baixar a mão.



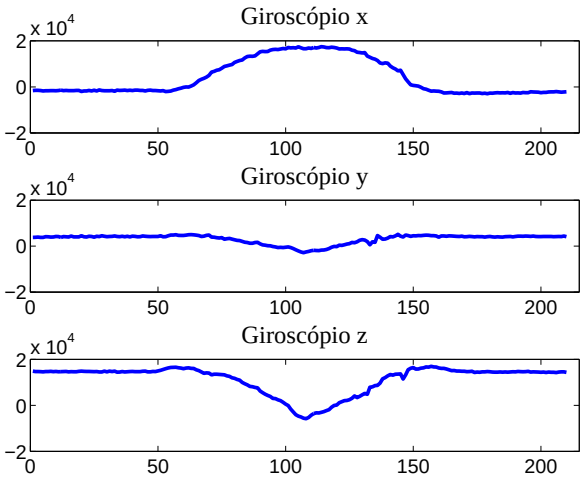
(a) Giroscópios do sensor 1.



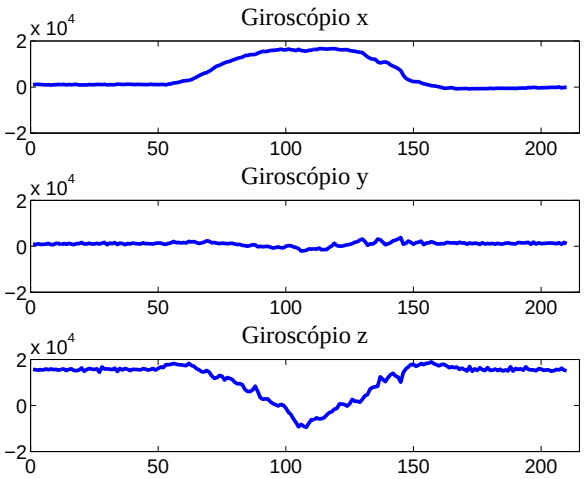
(b) Giroscópios do sensor 2.



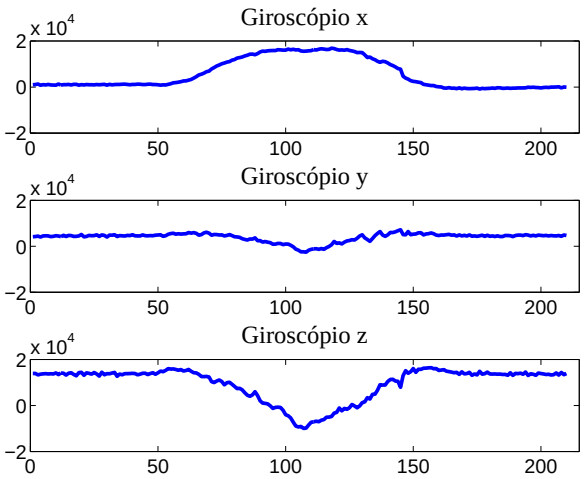
(c) Giroscópios do sensor 3.



(d) Giroscópios do sensor 4.

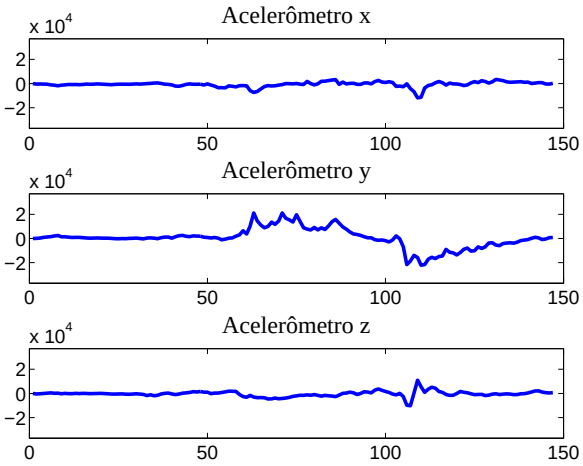


(e) Giroscópios do sensor 5.

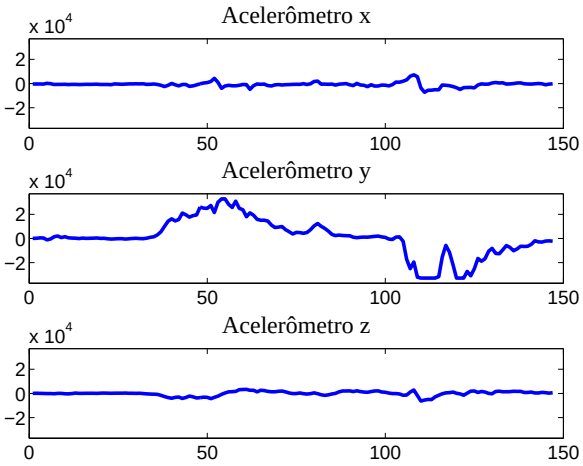


(f) Giroscópios do sensor 6.

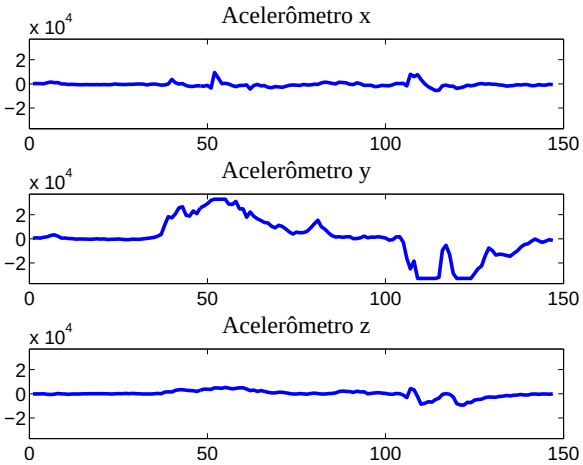
Figura 23 – Gráficos gerados pelos giroscópios durante o movimento de erguer e baixar a mão.



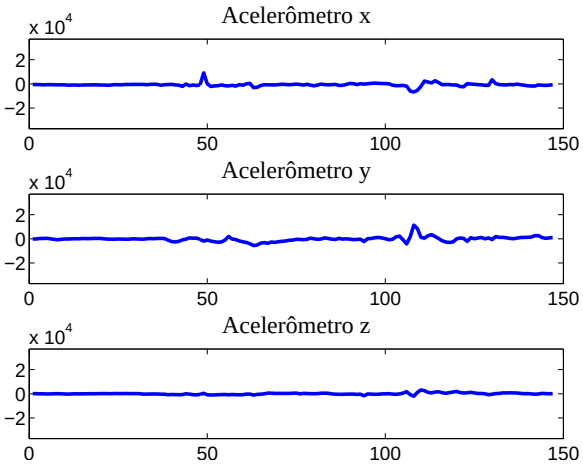
(a) Acelerômetros do sensor 1.



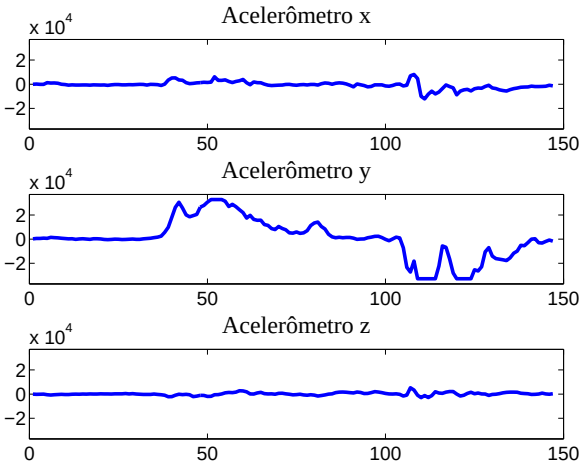
(b) Acelerômetros do sensor 2.



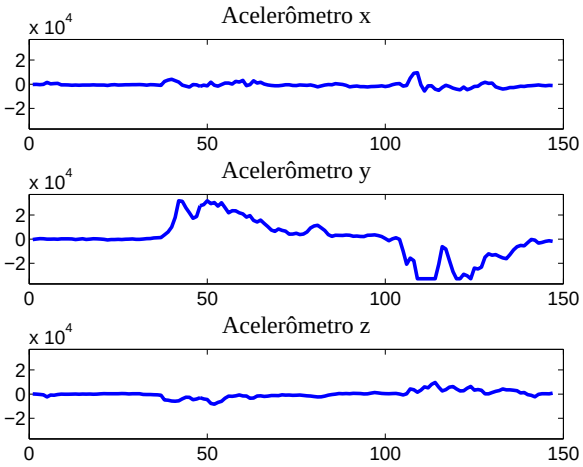
(c) Acelerômetros do sensor 3.



(d) Acelerômetros do sensor 4.

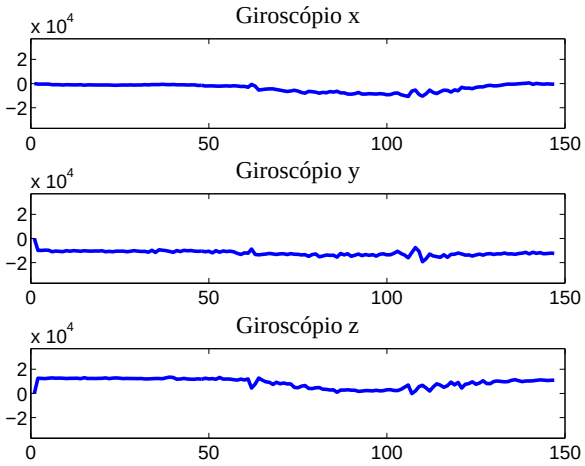


(e) Acelerômetros do sensor 5.

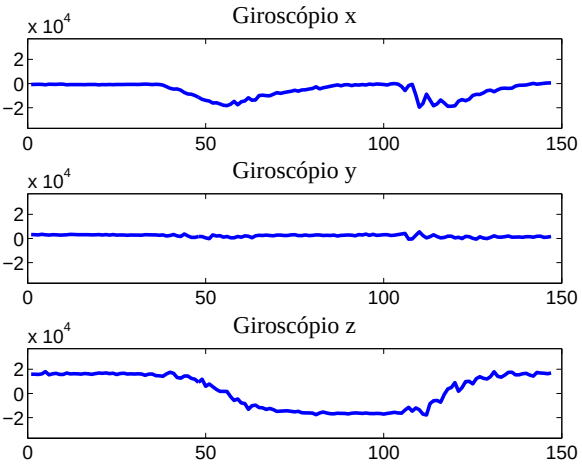


(f) Acelerômetros do sensor 6.

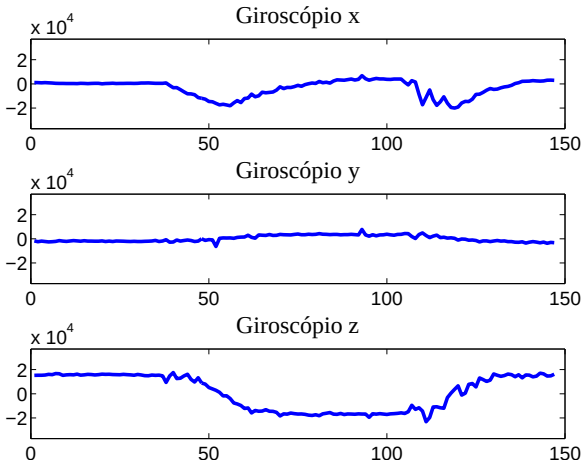
Figura 24 – Gráficos gerados pelos acelerômetros durante o movimento de fechar e abrir a mão.



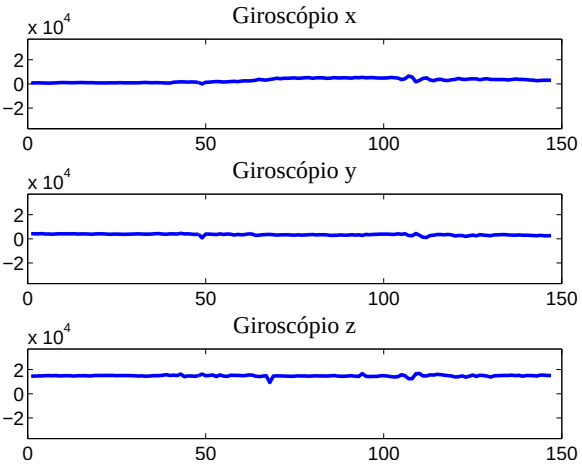
(a) Giroscópios do sensor 1.



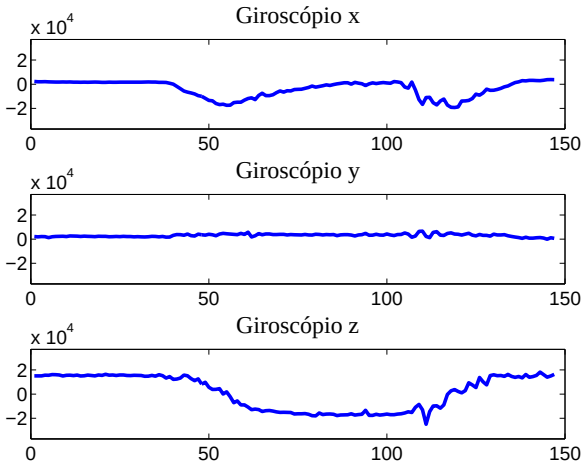
(b) Giroscópios do sensor 2.



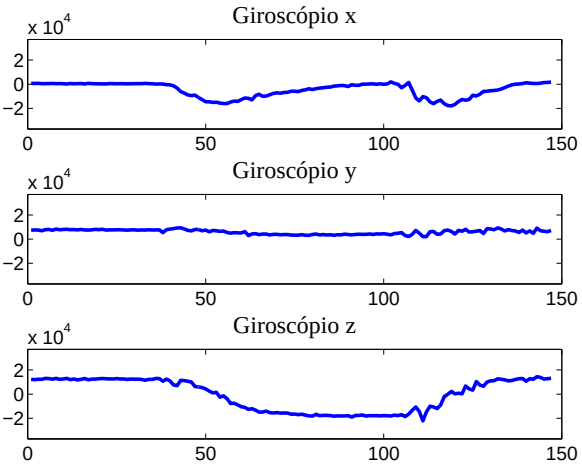
(c) Giroscópios do sensor 3.



(d) Giroscópios do sensor 4.



(e) Giroscópios do sensor 5.



(f) Giroscópios do sensor 6.

Figura 25 – Gráficos gerados pelos giroscópios durante o movimento de fechar e abrir a mão.

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de uma luva para captura de gestos. Toda a parte de comunicação foi implementada no microcontrolador PAMPIUM e descrita em SystemVerilog. Na fase atual, o projeto demostrou resultados positivos, sendo capaz de captar e enviar para um computador dados referentes a aceleração e inclinação de 6 sensores.

A utilização do PAMPIUM facilitou bastante a implementação do projeto, pois por se tratar de um microcontrolador reconfigurável e descrito em SystemVerilog proporcionou a possibilidade de ligar as interfaces de comunicação diretamente ao seu banco de registradores. Com o uso dos sensores MPU6050 foi possível captar e enviar os dados diretamente para o computador. Isto pelo fato dos sensores possuírem conversores internos Analógico/Digital com uma precisão de 16 bits.

A comunicação do PAMPIUM com os sensores foi de 400 kbits/segundo e a comunicação com o computador foi de 28800 bit/segundo. Proporcionando assim, uma taxa de amostragem de 30,1 amostras por segundo. Esta taxa de amostragem se mostrou suficiente para a captação dos gestos executados.

Trabalhos Futuros

Este trabalho faz parte de um projeto, onde o objetivo é reconhecer os gestos captados pela luva. Portanto inúmeras aplicações podem ser dadas a esta luva, uma delas é a substituição de alguns dispositivos de controle, tais como: teclado, mouse, passador de slides, etc. Algumas sugestões de trabalhos futuros são apresentadas abaixo:

- Implementação de comunicação sem fio com o computador, para substituir a comunicação RS232, atualmente utilizada;
- Aumento da taxa de amostragem, por meio da substituição do demultiplexador de saída, para 6 blocos de comunicação I2C independentes;
- Desenvolvimento de técnicas para o reconhecimento de movimentos, a partir dos dados captados;
- Análise da necessidade de filtragem do sinal;
- Desenvolvimento e novas aplicações para a luva.

REFERÊNCIAS

ALTERA. **DE2, development and education board - User Manual**. [S.l.], 2006.

AXELSON, J. **Serial Port Complete: COM Ports, USB Virtual COM Ports and Ports for Embedded Systems**. [S.l.]: Lakeview Research LLC, 2007.

ENGROFF, A. M. **PAMPIUM I - PROJETO E IMPLEMENTA ¸CAO DE UM˜ MICROCONTROLADOR DE 16 BITS EM ARQUITETURA RISC**. Dissertação (Mestrado) — Universidade Federal do Pampa – Unipampa, 2014.

HOROWITZ, P.; HILL, W. **The Art Of Electronics**. [S.l.]: Press Syndicate of the University of Cambridge, 1989.

INFO, I. I2c bus,interface and protocol. **Disponível em: <http://i2c.info/ >**, 12 de Setembro 2015.

INSTRUMENTS, N. **GPIB-232/485CT-A User Manual**. [S.l.], 1999.

INTRUMENTS, T. **MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS**. [S.l.], 2006.

INVENSENSE. **MPU-6000 and MPU-6050 Product Specification Revision 3.4**. 1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A., 2013. Disponível em: <www.invensense.com>.

INVENSENSE. **MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2**. 1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A., 2013. Disponível em: <www.invensense.com>.

JIN, S. et al. Interaction and control with the auxiliary of hand gesture. In: **Information Science and Technology (ICIST), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 1362–1366.

KIM, J.-H.; THANG, N. D.; KIM, T.-S. 3-d hand motion tracking and gesture recognition using a data glove. In: **Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on**. [S.l.: s.n.], 2009. p. 1013–1018.

KUMAR, P.; RAUTARAY, S.; AGRAWAL, A. Hand data glove: A new generation real-time mouse for human-computer interaction. In: **Recent Advances in Information Technology (RAIT), 2012 1st International Conference on**. [S.l.: s.n.], 2012. p. 750–755.

PHILIPS. The i2c bus specification version 2.1. **Philips Semiconductors**, 2000.

VENKATESWARAN, P. et al. Design and implementation of fpga based interface model for scale-free network using i2c bus protocol on quartus ii 6.0. **International Conference on Computers and Devices for Communication**, 2009.

Apêndices

APÊNDICE A – PROGRAMA DE LEITURA E ENVIO DE DADOS

```
NOP;
BSET 0(1),3;
CALL ACORDA;
BSET 0(0),7;
CALL ACORDA;
BSET 0(0),8;
BCLEAR 0(0),7;
CALL ACORDA;
BSET 0(0),7;
CALL ACORDA;
BSET 0(0),9;
BCLEAR 0(0),7;
BCLEAR 0(0),8;
CALL ACORDA;
BSET 0(0),7;
CALL ACORDA;
BCLEAR 0(0),9;
BCLEAR 0(0),7;
BSET 0(0),5;
CALL VERIFICA_F_INT;
FUNCTION MAIN;
CALL LEITURA_SENSOR;
BSET 0(0),7;
CALL LEITURA_SENSOR;
BSET 0(0),8;
BCLEAR 0(0),7;
CALL LEITURA_SENSOR;
BSET 0(0),7;
CALL LEITURA_SENSOR;
BSET 0(0),9;
BCLEAR 0(0),7;
BCLEAR 0(0),8;
CALL LEITURA_SENSOR;
BSET 0(0),7;
CALL LEITURA_SENSOR;
BCLEAR 0(0),9;
BCLEAR 0(0),7;
CALL MAIN;
END;
FUNCTION LEITURA_SENSOR;
CALL ACELEROMETRO_X;
COPY 6(0), 11(0);
CALL ENVIA_RS232;
CALL ACELEROMETRO_Y;
COPY 7(0), 11(0);
CALL ENVIA_RS232;
CALL ACELEROMETRO_Z;
COPY 8(0), 11(0);
CALL ENVIA_RS232;
CALL GIROSCOPIO_X;
COPY 9(0), 11(0);
CALL ENVIA_RS232;
CALL GIROSCOPIO_Y;
COPY A(0), 11(0);
CALL ENVIA_RS232;
CALL GIROSCOPIO_Z;
COPY B(0), 11(0);
CALL ENVIA_RS232;
RET;
FUNCTION ENVIA_RS232;
COPY 11(0),F(0);
BSET 0(0),6;
CALL VERIFICA_F_INT;
SHR 11(0),8,F(0);
BSET 0(0),6;
CALL VERIFICA_F_INT;
RET;
FUNCTION VERIFICA_F_INT;
MOVL 2(0);
FUNCTION VERIFICA;
BBSET 0(1),6,1(0);
JUMP VERIFICA;
BCLEAR 0(1),6;
RET;
FUNCTION ACELEROMETRO_X;
MOVL 3B(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,6(0);
MOVL 3C(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD 6(0),4(0),6(0);
RET;
FUNCTION ACELEROMETRO_Y ;
MOVL 3D(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,7(0);
MOVL 3E(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD 7(0),4(0),7(0);
RET;
FUNCTION ACELEROMETRO_Z;
MOVL 3F(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,8(0);
MOVL 40(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD 8(0),4(0),8(0);
RET;
FUNCTION GIROSCOPIO_X;
MOVL 43(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,9(0);
MOVL 44(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD 9(0),4(0),9(0);
RET;
FUNCTION GIROSCOPIO_Y;
MOVL 45(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,A(0);
MOVL 46(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD A(0),4(0),A(0);
RET;
FUNCTION GIROSCOPIO_Z;
MOVL 47(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
SHL 4(0),8,B(0);
MOVL 48(0);
COPY 1(0),5(0);
CALL LEITURA_REGISTRADOR;
NOP;
ADD B(0),4(0),B(0);
RET;
FUNCTION LEITURA_REGISTRADOR;
MOVL D0(0);
COPY 1(0),4(0);
BSET 0(0),0;
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_1;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_1;
COPY 5(0), 4(0);
BSET 0(0),3;
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_2;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_2;
```

```
MOVL D1(0);
COPY 1(0),4(0);
BCLEAR 0(0),3;
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_3;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_3;
BSET 0(0),3;
BSET 0(0),2;
BSET 0(0),1;
MOVL 2(0);
FUNCTION LEITURA_1;
BBCLEAR 0(0),1,1(0);
JUMP LEITURA_1;
BCLEAR 0(0),2;
BCLEAR 0(0),3;
RET;
FUNCTION ACORDA;
MOVL D0(0);
COPY 1(0),4(0);
BSET 0(0),0;
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_10;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_10;
MOVL 6B(0);
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_20;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_20;
MOVL 0(0);
COPY 1(0),4(0);
BSET 0(0),4;
BSET 0(0),1;
MOVL 2(0);
FUNCTION ESCRITA_30;
BBCLEAR 0(0),1,1(0);
JUMP ESCRITA_30;
BCLEAR 0(0),2;
BCLEAR 0(0),3;
RET;
```

APÊNDICE B – DESCRIÇÃO EM SYSTEMVERILOG DA INTERFACE I2C

```
module Master(
input logic clk,
rst, rw, S, ena_in,
busy_in, output logic
ack, ena_wf, ena_wd,
f_int, busy_out,
ena_out, input logic
scl_in, sda_in,
input logic[0:7]data_in,
output reg[0:7]data_out,
output reg scl_out,
sda_out, c_sda,
input logic ack_in);

reg [0:4]state, next;
reg sp;
reg [0:3]cont;
reg [0:7]intd;
reg r_ack;
reg [0:2]cont2;


always_ff@(posedge clk or
negedge rst)begin

if (!rst)begin
state<=0;
end
else begin
state<=next;
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
next=0;
end
else begin
case (state)

0:begin
if (busy_in & ena_in)
begin
next=2;
end
end

2:begin
if(busy_in) begin
next=3;
end
end

3:begin
next=4;
end

4:begin
next=5;
end

5:begin
next=6;
end

6:begin
if(busy_in) begin
next=7;
end
end

7:begin
if(rw)begin
next=19;
end
else begin
next=8;
end
end

8:begin
next=9;
end

9:begin
next=10;
end

10:begin
next=11;
end

11:begin
next=12;
end

12:begin
if (cont<7) begin
next=9;
end
else begin
next=13;
end
end

13:begin
next=14;
end

14:begin
next=15;
end

15:begin
next=16;
end

16:begin
if(!ack_in) begin
if(!ack) begin
next=17;
end
else begin
if (cont<4) begin
next=13;
end
else begin
next=17;
end
end
end
else begin
next=28;
end
end


17:begin
if(!ack) begin
next=18;
end
else begin
next=28;
end
end

19:begin
next=20;
end

20:begin
next=21;
end

21:begin
next=22;
end

22:begin
```

```
if (cont<7)begin
next=19;
end
else begin
next=23;
end
end

23:begin
next=24;
end

24:begin
next=25;
end

25:begin
next=26;
end

26:begin
next=27;
end

27:begin
if(!S)begin
next=18;
end
else begin
next=28;
end
end

18:begin
if(S) begin
next=2;
end
else begin
next=6;
end
end

28:begin
next=29;
end

29:begin
next=30;
end

30:begin
next=0;
end

default: begin
next=0;
end
endcase
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
sda_out=0;
end
else begin
case (state)

default: begin
sda_out=0;
end

3:begin
sda_out=1;
end

4:begin
sda_out=1;
end


9:begin

sda_out=intd[cont];
end

10:begin
sda_out=sda_out;
end

11:begin
sda_out=sda_out;
end

12:begin
sda_out=sda_out;
end

23:begin
sda_out=S;
end

24:begin
sda_out=sda_out;
end

25:begin
sda_out=sda_out;
end

26:begin
sda_out=sda_out;
end


30:begin
sda_out=1;

end
endcase
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
scl_out=0;
end
else begin
case (state)

default: begin
scl_out=0;
end

4:begin
scl_out=1;
end

5:begin
scl_out=1;
end

10:begin
scl_out=1;
end

11:begin
scl_out=1;
end

14:begin
scl_out=1;
end

15:begin
scl_out=1;
end

20:begin
scl_out=1;
end

21:begin
scl_out=1;
```

```
end

24:begin
scl_out=1;
end

25:begin
scl_out=1;
end

29:begin
scl_out=1;
end

30:begin
scl_out=1;
end
endcase
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
intd=0;
end
else begin
case (state)

default: begin
intd=0;
end

8:begin
intd[7]=data_in[7];
intd[6]=data_in[6];
intd[5]=data_in[5];
intd[4]=data_in[4];
intd[3]=data_in[3];
intd[2]=data_in[2];
intd[1]=data_in[1];
intd[0]=data_in[0];

end

9:begin
intd=intd;
end

10:begin
intd=intd;
end

11:begin
intd=intd;
end

12:begin
intd=intd;
end

19:begin
intd=intd;
end

20:begin
intd=intd;
end

21:begin
intd[cont]=sda_in;
end

22:begin
intd=intd;
end

23:begin
intd=intd;
end

endcase
end


end


always@(negedge clk or
 negedge rst) begin

if(!rst) begin
data_out=0;
end
else begin
case (state)

default: begin
data_out=0;
end

23:begin
data_out=intd;
end

24:begin
data_out=data_out;
end

25:begin
data_out=data_out;
end

endcase
end
end


always@(posedge clk or
negedge rst) begin

if(!rst) begin
busy_out=0;
ena_out=0;
end
else begin
case (next)

default: begin
busy_out=1;
ena_out=1;
end

2:begin
busy_out=0;
ena_out=1;
end

6:begin
busy_out=0;
ena_out=1;
end

17:begin
busy_out=1;
ena_out=1;
end

18:begin
busy_out=0;
ena_out=1;
end

30:begin
busy_out=0;
ena_out=0;
end


endcase
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
r_ack=0;
```

```
ack=0;
end
else begin
case (state)

default: begin
r_ack=0;
ack=0;
end


15:begin
r_ack=sda_in;
ack=0;
end

16:begin
r_ack=r_ack;
ack=0;
end

17:begin
r_ack=r_ack;
ack=r_ack;
end


endcase
end
end


always@(negedge clk or
negedge rst) begin

if(!rst) begin
ena_wf=0;
ena_wd=0;
f_int=0;
end
else begin
case (state)

default: begin
ena_wf=0;
ena_wd=0;
f_int=0;
end

17:begin
ena_wf=1;
ena_wd=0;
f_int=0;
end

18:begin
ena_wf=1;
ena_wd=0;
f_int=1;
end

23:begin
ena_wf=0;
ena_wd=1;
f_int=0;
end

24:begin
ena_wf=0;
ena_wd=1;
f_int=0;
end

28:begin
ena_wf=0;
ena_wd=0;
f_int=0;
end

29:begin
ena_wf=1;
ena_wd=0;
f_int=1;
end

30:begin
ena_wf=1;
ena_wd=0;
f_int=1;
end

endcase
end
end


always@(negedge clk or
 negedge rst) begin

if(!rst) begin
cont=0;
cont2=0;
end
else begin
case (state)

default: begin
cont=0;
cont2=0;
end

9:begin
cont=cont;
cont2=0;
end

10:begin
cont=cont;
cont2=0;
end

11:begin
cont=cont;
cont2=0;
end

12:begin
cont=cont+1'b1;
cont2=0;
end

13:begin
cont=0;
cont2=cont2;
end

14:begin
cont=0;
cont2=cont2;
end

15:begin
cont=0;
cont2=cont2;
end

16:begin
cont=0;
cont2=cont2+1'b1;
end


19:begin
cont=cont;
cont2=0;
end

20:begin
cont=cont;
cont2=0;
end

21:begin
cont=cont;
cont2=0;
end

22:begin
cont=cont+1'b1;
cont2=0;
end
```

```
                                c_sda=1;                        end
                                end
endcase                                                         25:begin
end                             7:begin                         c_sda=1;
end                             c_sda=1;                        end
                                end
                                                                26:begin
always@(negedge clk or          8:begin                         c_sda=1;
negedge rst)begin               c_sda=1;                        end
                                end
if (!rst)begin                                                  27:begin
c_sda=0;                        9:begin                         c_sda=1;
end                             c_sda=1;                        end
else begin                      end
case (state)                                                    28:begin
                                10:begin                        c_sda=1;
default: begin                  c_sda=1;                        end
c_sda=0;                        end
end                                                             29:begin
                                11:begin                        c_sda=1;
3:begin                         c_sda=1;                        end
c_sda=1;                        end
end                                                             30:begin
                                12:begin                        c_sda=1;
4:begin                         c_sda=1;                        end
c_sda=1;                        end
end                                                             endcase
                                23:begin                        end
5:begin                         c_sda=1;                        end
c_sda=1;                        end
end
                                24:begin                        endmodule
6:begin                         c_sda=1;
```

APÊNDICE C – DESCRIÇÃO EM SYSTEMVERILOG DA INTERFACE RS232

```
module RS232(
input clk,rst,ena,go_in,
output logic tx,f_int_rs,
output logic go_out,
input logic rx,
output logic
[7:0]data_out_rs,
input logic
[7:0]data_in_rs,
output logic en_wf_rs,
output logic en_wd_rs);

reg [9:0] data;
reg [3:0] state, next;
reg [2:0] cont2;
reg [20:0] begin contbr,
contbr2,cont;
end
reg [17:0] br;


always@(posedge clk or
negedge rst)begin
if (!rst)begin
state<=0;
end
else begin
state<=next;
end
end

always@(negedge clk or
negedge rst)begin
if (!rst)begin
next=0;
end
else begin
if(ena) begin
case(state)

default:begin
next=0;
end

0:begin
if(!rx & ena)begin
next=1;
end
end

1:begin
if (rx) begin
next=2;
end
end

2:begin
if (!rx)begin
next=3;
end
end

3:begin
if (rx)begin
next=4;
end
end

4:begin
if(cont2<3)begin
next=2;
end
else
next=5;
end

5:begin
next=6;
end

6:begin
if(!rx)begin
next=11;
end
else begin
if(go_in)begin
next=7;
end
end
end

7:begin
next=8;
end

8:begin
next=9;
end

9:begin
if(contbr!=br)begin
next=9;
end
else begin
if (cont<10)begin
next=8;
end
else
next=10;
end
end

10:begin
next=15;
end

11:begin
if(contbr2!=br[17:1])begin
next=11;
end
else begin
next=12;
end
end

12:begin
if(contbr!=br) begin
next=12;
end
else begin
next=13;
end
end

13:begin
if(cont<7) begin
next=12;
end
else begin
next=14;
end
end

14:begin
if(rx)begin
next=15;
end
end

15:begin
next=6;
end

endcase
end
else begin
next=0;
end
end
end

always@(posedge clk or
```

```
negedge rst)begin
if(!rst)begin
go_out=0;
f_int_rs=0;
end
else begin
case (state)

default: begin
go_out=0;
f_int_rs=0;
end

5:begin
go_out=0;
f_int_rs=1;
end

7:begin
go_out=1;
f_int_rs=0;
end

8:begin
go_out=1;
f_int_rs=0;
end

9:begin
go_out=1;
f_int_rs=0;
end

10:begin
go_out=0;
f_int_rs=1;
end

14:begin
f_int_rs=1;
go_out=0;
end

15:begin
f_int_rs=1;
go_out=0;
end

endcase
end
end

always@(posedge clk or
negedge rst)begin
if (!rst)begin
cont=0;
cont2=0;
end

else begin
case(state)

default: begin
cont=0;
cont2=0;
end

2:begin
cont=cont+1'b1;
cont2=cont2;
end

3:begin
cont=cont+1'b1;
cont2=cont2;
end

4:begin
cont=cont+1'b1;
cont2=cont2+1'b1;
end

5:begin
cont=cont;
cont2=cont2;
end

8:begin
cont=cont+1'b1;
cont2=0;
end

9:begin
cont=cont;
cont2=0;
end

12:begin
cont=cont;
cont2=0;
end

13:begin
cont=cont+1'b1;
cont2=0;
end

endcase
end
end

always@(posedge clk or
negedge rst)begin
if (!rst)begin
data=0;
data_out_rs=0;
end

else begin
case(state)

default: begin
data=0;
data_out_rs=0;
end

7:begin
data[0]=0;
data[8:1]=data_in_rs[7:0];
data[9]=1;
data_out_rs=0;
end

8:begin
data=data;
data_out_rs=0;
end

9:begin
data=data;
data_out_rs=0;
end

12:begin
data=data;
data_out_rs=0;
end

13:begin
data[cont]=rx;
data_out_rs=0;
end

14:begin
data=data;
data_out_rs[7:0]=data[7:0];
end

15:begin
data=data;
data_out_rs=data_out_rs;
end

endcase
end
end

always@(posedge clk or
negedge rst)begin
if (!rst)begin
br<=0;
```

```
end

else begin
case(state)

default: begin
br<=br;
end

0:begin
br<=0;
end

5:begin
br<=cont[17:3];
end

endcase
end
end

always@(posedge clk or
 negedge rst)begin
if (!rst)begin
contbr=0;
contbr2=0;
end

else begin
case(state)

default: begin
contbr=0;
contbr2=0;
end

9:begin
contbr=contbr+1'b1;
contbr2=0;
end

11:begin
contbr=0;
contbr2=contbr2+1'b1;
end

12:begin
contbr=contbr+1'b1;
contbr2=0;
end

endcase
end
end

always@(posedge clk or
negedge rst)begin
if (!rst)begin
tx=1;
end

else begin
case(state)

default: begin
tx=1;
end

8:begin
tx=data[cont];
end

9:begin
tx=tx;
end

endcase
end
end

always@(posedge clk or
negedge rst)begin
if (!rst)begin
en_wd_rs=0;
en_wf_rs=0;
end

else begin
case(state)

default: begin
en_wd_rs=0;
en_wf_rs=0;
end

4:begin
if(cont2<3)begin
en_wd_rs=0;
en_wf_rs=0;
end
else begin
en_wd_rs=0;
en_wf_rs=1;
end
end

5:begin
en_wd_rs=0;
en_wf_rs=1;
end

10:begin
en_wd_rs=0;
en_wf_rs=1;
end

14:begin
en_wd_rs=1;
en_wf_rs=1;
end

15:begin
en_wd_rs=en_wd_rs;
en_wf_rs=1;
end

endcase
end
end


endmodule
```