Faculdade de Engenharia da Universidade do Porto
Mestrado Integrado em Engenharia Eletrótécnica e de computadores

Electronic Systems final project

Workbench 1

André Filipe Ávila Silva
Francisco Gil Cerqueira Correia

2019

Conteúdo

1 Introdução

In this document it is described both systems, developed to fulfill the requirements listed on both tasks of the final Project, the JTAG controller for PIC32 microprocessor and the Pong game on Arduino which required developing I2C and SPI libraries.

2 PIC32 microprocessor JTAG controller

2.1 Description

In this system it was programmed an Arduino board as a JTAG controller to make use of the PIC32 microprocessor JTAG capabilities on the ChipKIT UNO32 board.

It was used Serial communication to choose which functionality we want to execute and read the JTAG output when needed.

Also, it was implemented reading the ID CODE, turn ON or OFF a specific LED and read a pin state to which we attached a button.

2.2 Firmware implementation

`void JTAGPort::begin():`

- Setup TMS, TCK, TDO and TDI pins.
- Reset JTAG state machine.
- Doesn't return.

`void JTAGPort::reset():`

- set TMS pin to HIGH.
- Do 5 clock Cycles to reach reset state on Jtag state machine.
- Doesn't return.

`void JTAGPort::clockPulse():`

- Set TCK pin to HIGH.
- Wait 3 microseconds.
- Set TCK pin to LOW.
- Wait 3 microseconds.
- Doesn't return.

`void JTAGPort::sendBits(uint8_t tms, uint8_t tdo):`

- Set TMS pin to TMS.
- Set TDO pin to TDO.
- Do a clock cycle.
- Doesn't return.

`void JTAGPort::sendBits(uint8_t tms, uint8_t tdo, uint8_t size):`

- Sends TMS and TDO bits up to position size.

`void JTAGPort::setDR():`

- Set JTAG state machine to DATA SHIFT state.

`void JTAGPort::setIR():`

- Reset JATG state machine.
- Set JTAG state machine to INTRUCTION SHIFT.

`void JTAGPort::sendInst(uint8_t instruction):`
- Set JTAG state machine to INSTRUCTION SHIFT.
- Write chosen instruction.
- Exit into JTAG state machine ready state.

`void JTAGPort::sendData(uint8_t tdo_pos, uint8_t value):`
- Set JTAG state machine to DATA SHIFT state.
- Bit shift 0 into chosen pin Input register.
- Bit shift value into chosen pin Output register.
- Bit shift 1 into chosen pin Control register.
- Bit shift 1 into Reset.
- Exit into JTAG state machine ready state.

`uint8_t JTAGPort::readData():`
- Read from TDI pin.
- Shift data bit.
- return 1 if TDI pin is HIGH or 0 if LOW.

`uint8_t JTAGPort::readData(uint8_t tdo_pos):`
- Set JTAG state machine to DATA SHIFT state.
- Send bits until reaching tdo_pos.
- Read TDI pin.
- Exit into JTAG state machine ready state.
- return 1 if TDI pin is HIGH or 0 if LOW.

`uint32_t JTAGPort::getID():`
- Set intruction to IDCODE.
- Set JTAG state machine to DATA SHIFT state.
- Read TDI and bitshift readings into value.
- Exit into JTAG state machine ready state.
- return value.

`void JTAGPort::setLED(uint8_t value):`
- Set Intruction to EXTERNAL TEST.
- Write 1 to LED pin register.

`uint8_t JTAGPort::checkButton():`
- Set Instruction to SAMPLE AND PRELOAD.
- Read button pin register.
- return value read.

`void setup():`
- Start JTAG library.
- Start Serial port.
- Print commands.

`void loop():`

- Checks if theres any command available in the Serial port:
  - If command 'd', get and print ID code in hexadecimal.
  - If command '1', turn on LED.
  - If command '0', turn off LED.
  - If command 'b', check button state.

2.3 Firmware test

- Clone this repository with `git clone https://git.fe.up.pt/up201808899/sele_a2_b01_jtag.git`.
- Compile the *main.cpp* file with PlatformIO to get *.hex* file.
- Flash the *.hex* file to the master and slaves.
- Run Putty with baud rate 9600;

3 Pong with I2C sonar SRF08 and SPI display PCD8544

3.1 Description

In this system, it was recreated the well know game Pong using the sonar SRF08 to control a paddle and the display PCD8544 to display the board.

To interface with the sonar, it was written a simple generic I2C driver for devices acting as Masters and a lib specifically to control and configure the sonar to read small distances.

To interface the LCD, it was developed a small SPI library to control the Arduino SPI port.

3.2 Firmware implementation

`void setupI2C():`

- Initialize SDA and SCL lines to HIGH.
- Set Two Wire Interface Prescaler.
- Initialize I2Cstate.

`void void attachErrorHandler(void (*function)(uint8_t,uint8_t):`

- Attach a function to be run when an unsupported command is detected.

`void i2cstop():`

- Set TWCR to stop current I2C communication with STOP bit.

`void writeI2C(uint8_t address, uint8_t value[], uint8_t size):`

- Wait until I2Cstate is I2CREADY.
- Set I2Cstate to I2CBUSY.
- If error is detected run error_handler.
- Reset error flag.
- Copy from value[] to internal write buffer.

- Set I2Caddress to write from address.
- Send START signal through the Two Wire Interface.

`void readI2C(uint8_t address, uint8_t values[], uint8_t size):`

- Wait until I2Cstate is I2CREADY.
- Set I2Cstate to I2CBUSY.
- If error is detected run error_handler.
- Reset error flag.
- Set I2Caddress to read from address.
- Send START signal through the Two Wire Interface.
- Wait until I2Cstate is I2CREADY signalling end of read from device.
- Write from internal read buffer to values[].

`ISR(TWI_vect):`

- Reads from Two Wire Interface Status register.
- Wait for aknowledged START.
- While write buffer is not empty Write to Two Wire interface Register.
- While read buffer is not full read from Two Wire interface Register.
- Set STOP at the end of operation
- If not supported status is detected send STOP and set error flag.

i2csonar

Library to interface the sonar with I2C.

`setupSonar():`

- Set up I2C.
- Set I2C error handler.

`setSonarRange():`

- Write Sonar Range address into the write buffer.
- Write range into write buffer.
- Send buffer to be written through I2C on address.

`setSonarRange():`

- Write Sonar Gain address into the write buffer.
- Write gain into write buffer.
- Send buffer to be written through I2C on address.

`void InitMeasure(uint8_t sonar_mode, uint8_t address):`

- Write Sonar command address into buffer.
- Write Initiate Readings command with sonar_mode offset to choose reading mode.
- Send buffer to be written through I2C on address.

`uint16_t readSonarValue(uint8_t address):`

- Write to buffer the first sonar reading address
- Send buffer to be written through I2C on address.
- Read 2 values from I2C on address.

- Return value from joining the 2 values.

ponglib

Library used for Pong game logic.

`void pongInit(uint16_t widthT, uint16_t heightT, uint8_t playersT,uint16_t paddleSize):`
- Initialize pong board with heightT and widthT.
- Initialize player stats and size.
- Use analogRead to get a random seed for initial ball direction.
- Reset board state.

`void pongReset():`
- Set ball to screen center.
- Set initial ball speed and random direction.

`void pongProc():`
- Calculate next ball position.
- Detect ball collision with paddles or board limits.
- Calculate new ball Speed and speed Multiplier if collision detected.
- Award points and reset board game state if Ball reached either end of the board.
- Update Ball position.

`uint8_t setPlayerPos(uint8_t player, uint16_t pos):`
- Get player struct from player.
- Update player paddle position with pos.
- return new position.

`void getBallPosInt(uint16_t *eBall):`
- Updates eBall with Ball position casted to uint16_t.

`uint16_t getPlayerPosition(uint16_t nplayer):`

- Get player struct from player.
- return player position.

`uint16_t getPlayerSize(uint16_t nplayer):`

- Get player struct from player.
- return player Size.

`uint16_t getPlayerPoints(uint16_t nplayer):`
- Get player struct from player.
- return player points.

spi

Library used to control the Arduino SPI port.

`void SPIClass::begin(uint8_t data_order, uint8_t master_slave, uint8_t mode, uint8_t fosc):`

- Setup SPI pins direction.
- Set SPI control register parameters.
- Set Slave Select pin High.
- Doesn't return.

`uint8_t SPIClass::transmit(char data):`

- Select slave.
- Put data to SPI data regiter.
- Wait for transmission to be completed.
- Return SPI data register which contains received data.

`uint8_t SPIClass::read():`

- Sends zeros in transmit method in order to get data.
- Return received data.

pcd8544

Library used to control the LCD screen PCD8544.

`void PCD8544::begin():`

- Initialize SPI communication.
- Set pin directions.
- Set reset pulse during 100 miliseconds.
- Setup LCD screen througn a command list.
- Doesn't return.

`void PCD8544::setCursor(uint8_t x, uint8_t y):`

- Set cursor in X, LINE postion.
- Doesn't return.

`void PCD8544::clear():`

- Writes zeros all over the screen to clear.
- Doesn't return.

`void PCD8544::drawBitmap():`

- Like the clear method, but this one prints given data.
- Doesn't return.

`void PCD8544::write(uint8_t data, uint8_t dc):`

- Send data or command.
- Doesn't return.

`void PCD8544::print(const char c):`

- Prints a given character from a matrix present in `pcd8544_ascii.h`.
- Doesn't return.

`void PCD8544::print(const char *c):`

- Prints a string by printing each character.
- Doesn't return.

pong_graphics

Library used to print Pong game graphics.

`void PongGraphics::drawFrame():`
- Draws upper, lower and side lines in order to create a frame.
- Returns nothing.

`void PongGraphics::drawPaddle(uint8_t lrPaddle, uint8_t yPos):`
- Draws left or rigth paddle according to a Y coordinate that corresponds to the coordinate of the top of the paddle.
- The paddle is 9 bits tall, so it can occupy at most two lines.
- Calculate both lines pixels.
- Doesn't return.

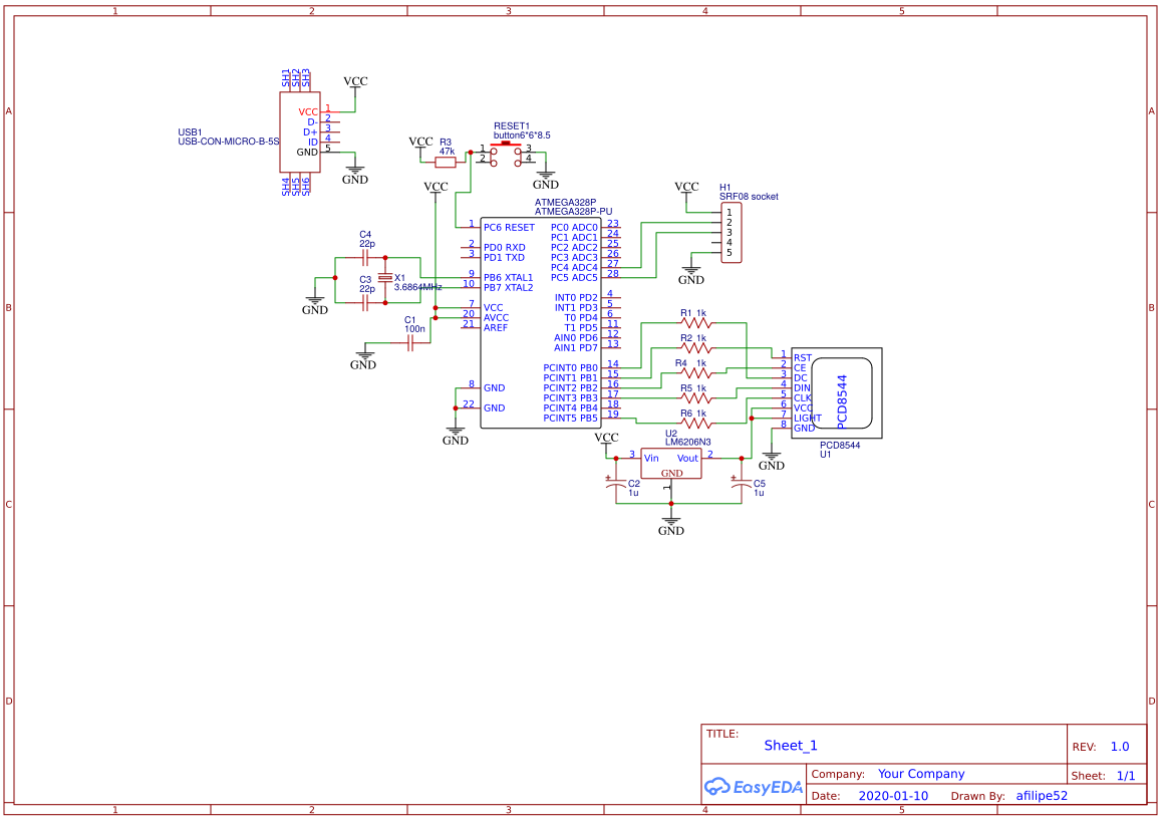
`void PongGraphics::erasePaddle(uint8_t lrPaddle):`
- Erase paddle so it can print another one.
- Down't return.

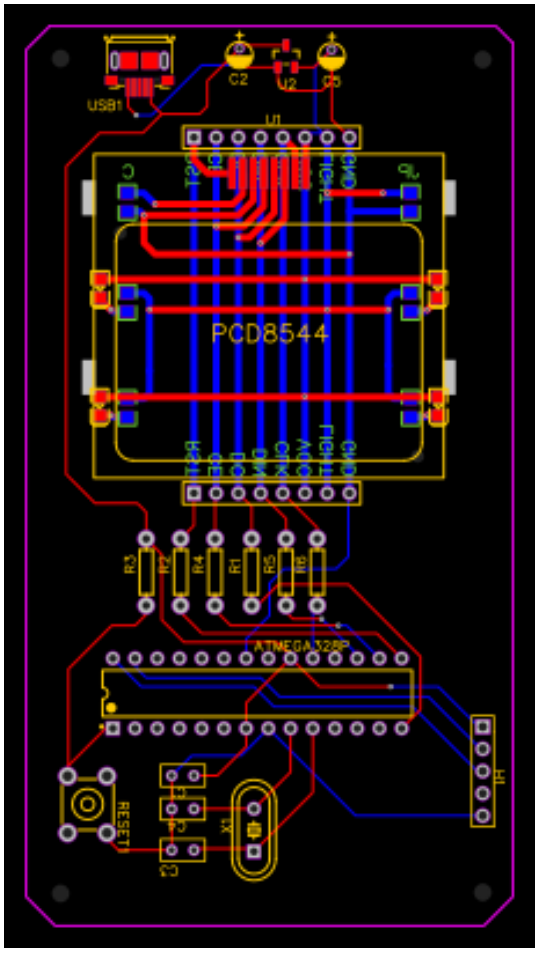
`void PongGraphics::drawBall(uint8_t x, uint8_t y):`
- Draw ball in the given X and Y coordinates.
- Doesn't return.

`void PongGraphics::eraseBall():`
- Erase ball so it can print the next one.
- Doesn't return.

3.3 Firmware test

- Clone this repository with `https://git.fe.up.pt/up201808899/sele_a2_b01_sync.git`.
- Compile the *main.cpp* file with PlatformIO to get *.hex* file.
- Flash the *.hex* file to the Arduino board.
- Enjoy the game :).


USB1
USB-CON-MICRO-B-5S
VCC
GND
R3
47k
RESET1
button6*6*8.5
ATMEGA328P
ATMEGA328P-PU
PC6 RESET
PD0 RXD
PD1 TXD
PB6 XTAL1
PB7 XTAL2
VCC
AVCC
AREF
GND
PC0 ADC0
PC1 ADC1
PC2 ADC2
PC3 ADC3
PC4 ADC4
PC5 ADC5
INT0 PD2
INT1 PD3
T0 PD4
T1 PD5
AIN0 PD6
AIN1 PD7
PCINT0 PB0
PCINT1 PB1
PCINT2 PB2
PCINT3 PB3
PCINT4 PB4
PCINT5 PB5
C4
22p
C3
22p
X1
3.6864MHz
C1
100n
H1
SRF08 socket
R1 1k
R2 1k
R4 1k
R5 1k
R6 1k
RST
CE
DC
DIN
CLK
VCC
LIGHT
GND
PCD8544
U1
U2
LM6206N3
Vin
Vout
C2
1u
C5
1u
TITLE:
Sheet_1
REV: 1.0
Company: Your Company
Sheet: 1/1
EasyEDA
Date: 2020-01-10
Drawn By: afilipe52


USB1
C2
U2
C3
U1
PCD8544
R3
R2
R4
R1
R5
R6
ATMEGA328P
RESET
C1
C2
X1
H1