Game application:	Points:	____ / 7
Bluetooth/UART:	Points:	____ / 5
Music Playback (MP3 and Volume):	Points:	____ / 4
GLCD:	Points:	____ / 5
Rand,ADC:	Points:	____ / 4
Overall:	Points:	____ / 25

Microcontroller VU

Application Protocol

Max Geiselbrechtinger, MatrNr. 01609418
e1609418@student.tuwien.ac.at

December 2, 2018

Contents

1 Overview **3**

1.1 Connections, External Pullups/Pulldowns 3

1.2 Design Decisions 4

1.3 Specialties 5

2 Main Application **6**

2.1 Main 6

3 Music Playback **6**

3.1 SPI 7

4 ADC **7**

4.1 RAND 7

5 LC–Display **7**

5.1 GLCD 7

5.2 HAL GLCD 7

6 WII MOTE **8**

6.1 HAL WT41 8

7 MENU **8**

8 Problems **9**

9 Work **9**

1 Overview

1.1 Connections, External Pullups/Pulldowns

Pin Assignment

bigAVR6	smartMP3
	J1 connected to 5V J5 connected to SP OUT1+ J6 connected to SP OUT1-
PB0 PB1 PB2 PB3 PB4 PB5 PD0 PG1	Connected to MP3_CS Connected to SCK Connected to MOSI Connected to MISO Connected to MP3_RST Connected to BSYNC Connected to DREQ Connected to MMC_CS
VCC GND	Connected to VCC Connected to GND
bigAVR6	WT41_Bluetooth
	JP RTS/CTS on
PJ0 PJ1 PJ2 PJ3 PJ5	Connected to TX Connected to RX Connected to RTS Connected to CTS Connected to RST
VCC GND	Connected to VCC Connected to GND
bigAVR6	ADC
J15 connected to PF0 P5 adjust volume	
bigAVR6	GLCD
SW14 LCD-BGLIGHT to VCC P3 adjust brightness	

All other DIP-Switches, that were not mentioned in this section, have to be switched off. Also every Jumper, that has not been mentioned, can be set arbitrarily.

If multiple resets of the bigAVR6 board occur, it is recommended to use an external power supply.

The smartMP3 extension board has to be equipped with a valid SD-card, otherwise initialization will fail.

1.2 Design Decisions

- **SD-Card:** A sdcard has to be present during the whole runtime of the application. Removing the sdcard after the initialization should not affect the gameplay, in any way other than stopping music playback. But it is not recommended to do this.

- **GLCD:** The glcd's hal filters all input addresses, through a modulo operation. This results in the effect, that drawing or writing text above any of the screens border will continue on the opposite edge of the screen.
  The text writing function will handle newlines according to the GNU style. Any other format of newlines will result in an undefined behavior. Also the program memory text writing function is limited to a string length of 32 characters, including the null terminator.

- **Wiimote:** The wii_user.c file of the libwiimote library has been modified to include disabling of the Wii's accelerometers.

- **Music:** The music playback is implemented as background task, which shovels junks of 32 bytes from the SD-card to the mp3-decoder as long as the mp3-decoder is not busy. It then remains IDLE, until the mp3-decoder requests more data. Volume will be adjusted if a new volume value has been generated by the ADC and no data transmission to the mp3-decoder is currently active.

- **ADC:** The adc issues a conversion every 5ms, and afterwards it switches the conversion mode. The volume is sampled through single conversion mode, where as the noise, used to seed the rand module, is generated through differential conversion mode of two floating input pins.

- **Rand:** The rand's linear feedback shift register is implemented through inline assembler to ensure true and efficient operation. The module also includes a rand8 function, which outputs random 8bit values.

- **Menu:** The menu is conceived as background task, but execution is restricted to 20Hz, to provide consistent game updates. It consists of a top-level state machine which handles connection to the Wiimote, the menu itself and the gameplay. All of the states contain state machines themselves, to remove redundancy and ensure progress of other background tasks.

- **Gameplay:** The gameplay is located in the gameloop menu state. To move platforms from bottom to top the RAM-shift of the glcd controllers is used. It starts by shifting every 6 game ticks until a threshold of 30 points is reached, then it will decrement the game tick threshold for the next shift by one and multiply the threshold by two. This happens until shifting occurs at each game tick. Every 14 shifts, a new platform, which is randomly chosen out of a set of platforms, gets inserted on the bottom of the screen. The ball can only move one pixel per game tick in x-direction but will fall two pixels per game tick downwards if no platform is beneath it. To manage the collision detection a ringbuffer holds all platforms under the ball. The platform right beneath the ball is checked for gaps to let the ball pass, otherwise the ball will be shifted up with the platform. The score is incremented every second and consists of a 16bit unsigned integer, to ensure that no overflow will occur during a long period of gameplay.

1.3 Specialties

- **Positiv:** I am quite satisfied with the way I implemented the menu. Particularly the player select came out really nice. Also the collision detection is managed in an efficient way I think.
- **Negativ:** There is definitely space to advance the game experience, such as faster shifts, different ball speeds or more random platforms. Also I would prefer if the gameplay would be outsourced in an external module, to keep the menu's code clearer.

2 Main Application

To control the application, a Wiimote has to be connected. To do so, you have to set the MAC-address in the file *mac.h*, inside the top-level directory, corresponding to the Wiimote's MAC-address. Note, issue a *make clean install* to make sure that the new MAC-address gets applied.

Then you will be asked to connect the Wiimote to the board. Do so, by pressing any button and wait until the led's of the controller stop flashing. If Led 1 lights up, the connection has successfully been established and you will see the homescreen of the game on the GLCD. If not, try pressing the sync button behind the battery cover.

In the game menu you can switch to the highscore board or the user select by issuing button 1 or 2 respectively. To get back to the home screen simply press button B.

The highscore board shows the five highest achieved scores since the last reset. Scores are sorted in ascending order, from top to bottom.

In the player select menu you can choose a player number though using the up and down arrow buttons. After selecting your player with button A the game will start right away.

During the game random platforms will stream from bottom to top. The ball should be moved left or right, by tilting the Wiimote left or right respectively, to fall through the gaps. The game ends if the ball hits the top of the screen or the home button is pressed. After every game the highscore table is shown. From there on you can move through the menu again as explained before.

2.1 Main

The main function is in charge of calling all initialization functions required for the application. It also calls the music playback and the game play functions. After returning from these functions it puts the CPU into a power saving IDLE mode. The main function depends on all modules of the following sections.

3 Music Playback

While the application is running a fancy song, stored on the sd-card, is played over the speakers of the mp3 extension board. The volume of the music can be adjusted through the potentiometer P5 on the bigAVR6 board. This module relies on the libmp3 and libsdcard library, provided by the LVA-team.

3.1 SPI

The SPI module provides an extra layer of abstraction to the mp3 and sd-card user application. After the SPI has been initialized the module can be used to send or receive data bytes over the spi bus. This module is required to ensure operation of the libmp3 and libsdcard libraries.

4 ADC

The ADC module is serving two tasks. One is to provide a seed for the PRNG and the other is to read the volume setting of potentiometer P5.

4.1 RAND

The rand module consist of a pseudo random number generator which is seeded though noise of the ADC to increase randomness. The random numbers are used to index an array of platforms to generate a pseudo random level. The random number generator relies on the ADC module.

5 LC–Display

The menu and game play is visualized on the 128 by 64 pixel GLCD.

5.1 GLCD

The glcd user module provides an extra layer of abstraction to the user application. It depends on the glcd hal module and consists of basic pixel manipulation functions, as well as more advanced drawing and text writing functions (eg. draw circle, fill rectangle or write text).

5.2 HAL GLCD

The hardware abstraction layer of the glcd handles the communication with the two glcd controller. It allows access to byte-sized pages of the y-axis, for every x-coordinate. The module has to memorize the latest address to handle the post increment of x-coordinate after read or write commands and to handle requests to the y-shift's state. It also manages a fluent switch between the controllers after a post increment. Through changing the controllers RAM start address a efficient screen shift, in the y direction, can be implemented.

6 WII MOTE

The libwiimote library presents a high level interface to the user application to communicate with the Wiimote. The library is provided by the LVA-team but requires the wt41 hal module to be present.

6.1 HAL WT41

The hardware abstraction layer of the wt41 module handles the UART connection to the bluetooth extension board. To ensure proper functionality the module includes a 32 byte ringbuffer to store received data.

7 MENU

The menu module handles the game menu and game play. It receives input from the Wiimote and issues output to the GLCD. It depends on the GLCD and Wiimote module.

8 Problems

After a revision of the specification I did not notice that the version of the libsdcard was updated and therefore wasn't able to get my music playback running. I also encountered problems while compiling the glcd module with advanced optimization modes. This led to drastically slowing down my module.

9 Work

The assumption column was left free, because I hadn't really worked on microcontrollers yet, nor had I programmed a project of this size before. Therefore I had no estimations on how long something should take me. I also did not protocol my working time precisely, therefore the times in the table are estimated values. Nevertheless I have spent quite a lot of time debugging correct code because of other bugs, like depreciated libraries or wrong compiler flags.

Task	Assumption (IP)	Reality
reading manuals, datasheets		15 h
program design		20 h
programming		10 h
debugging		80 h
protocol		5 h
Total		130 h