FPGA based wireless virtual reality system with 6-axis movement tracking (June 2016)

R. Fripon, M. Knoet, *IEEE*

***Abstract* — The goal of this project was to develop a virtual reality system consisting of a graphical software application and an embedded system, connected with each other via Wi-Fi to make it wireless and portable. The software application acts as a server and the embedded system acts as a client for that server. The client is FPGA based and uses the MPU6050, a 6-axis inertia sensor, to measure and calculate its own three-dimensional movements. It sends this information to the server, which in turn sends back an image stored in a local database. On the client side this image is displayed on a LCD-screen. By changing the image according to the movement, the client end user becomes a spectator in a virtual environment. The end result is a functional proof of concept which can be further improved by using the DMP^{TM} of the MPU6050 and adding a magnetometer for a more accurate tracking of the movements.**

***Index Terms* — Basys3, embedded system, FPGA, MPU6050, portable, server-client, SQL database, virtual reality, Wi-Fi**

I. Introduction

VIRTUAL REALITY is becoming increasingly more popular due to its applications in fields such as gaming. However, next to the highly commercialized entertainment industry, virtual reality can also be used for more practical applications. One such application could be a training program for firefighters, where they can practice in a virtual environment to eliminate any real dangers. With the use of an embedded system, wirelessly connected to a remote server, their position and viewing angle inside the virtual room can be calculated. Based on this information a certain part of the virtual environment can be displayed with special goggles, or on a screen held by the firefighter.

A virtual environment is described as a box of equidistant points, which represent the different coordinates. When walking around in this virtual environment, the user is basically moving from one coordinate to a consecutive one. To be able to 'look around' in this environment, each of the coordinates contains a 360° spherical panorama. This way the virtual scene is projected onto an orb with the coordinate at its center.

To achieve such a virtual reality system, following framework has been developed: the first part is a server application connected to a database, and the second part is an embedded system. The server software consists of only a few classes; the core is the main class which handles all graphical features of the application, and connected to it are a database class and a Wi-Fi class to handle the database connection and the wireless communication respectively. The client is FPGA based and uses the MPU6050 as a motion tracking device. A custom core is developed to operate this sensor. Furthermore, the embedded system consists of a LCD-screen to display the virtual environment and a Wi-Fi module to enable the wireless communication. For the communication between server and client, a lightweight custom UDP protocol is used.

II. Framework

The framework of this virtual reality (VR) system consists of two main parts: a software server application and an embedded system which is referred to as a client. Both are wirelessly connected via Wi-Fi, which has the advantage that the embedded system becomes a portable device. Also connected to the server is a database in which the virtual environments are stored. The client itself consists of a FPGA development board with connected to it: a Wi-Fi module for the wireless communication with the server, a motion tracking device to track the clients' three-dimensional movements, and a LCD-screen to display a certain part of the virtual environment. The complete framework is shown in Figure 1.

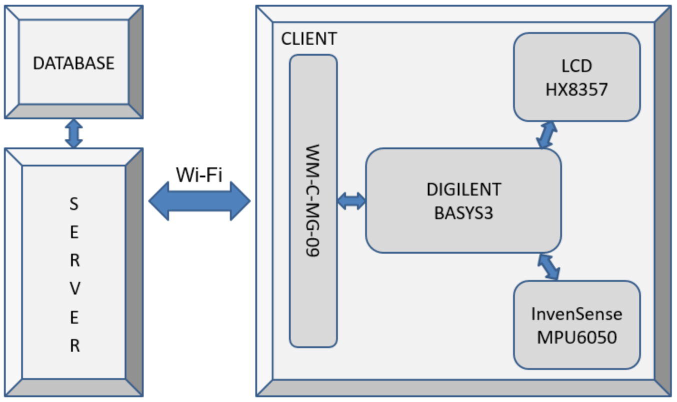


Figure 1. Framework of the embedded system.

Manuscript received June 28, 2016. R. Fripon and M. Knoet, Students electronics-ICT at the Hasselt University, Diepenbeek, B-3590 Belgium. E-mail: robin.fripon@student.uhasselt.be | marcus.knoet@student.uhasselt.be

III. Server side

The server side application is a graphical tool, developed with the Visual Studio 2015 software, in which the user gets to see all the connected clients and where he can assign a specific virtual environment to new clients that try to connect. It also provides tabs with: an overview of all the incoming data, a picture of the last sent image, and an interface for creating and displaying the virtual environments. Next to these graphical functionalities, the application also handles a database connection for the storage of the virtual environments, and it takes care of the wireless communication with all connected clients.

A. SQL database

The database used is a SQL local database and it has the sole purpose of storing all the images from the virtual environments. Each table in the database represents one virtual environment, and thus the name of this table is the name that one has chosen when creating that virtual environment. The structure of a table is very straight forward: there is one column for the coordinates, this is also the primary key, and there is a set of columns for every possible combination of the yaw and tilt.

The coordinate column is of type *varchar(50)* and contains all coordinates in the virtual environment. To improve the readability of the coordinates, a specific structure was used. The name of each coordinate is a concatenation of ‘x’, the value of the x-position, ‘y’, the value of the y-position, ‘z’, and the value of the z-position, e.g. x0y0z0.

The angle columns use the same structure for their names, namely y0t0, y18t0, etc., and they are of type *varbinary(max)* because they will contain the raw image data. Special about these names is that the values of the yaw and tilt are multiples of 18 and ranging from 0 to 342 for the yaw, and from -90 to 90 for the tilt. Discrete steps of eighteen degree angles were chosen to reduce the number of images while still maintaining a smooth transition between consecutive images.

B. Graphical user interface

The server side application has a simple and intuitive graphical user interface (GUI) which facilitates the interaction with the connected devices. The window is divided horizontally into a narrow panel on the left, which contains a tree view of all the connected clients, and a larger tabbed panel on the right. This latter has three different tabs: a Data-, a View-, and a Rooms-tab. On startup an empty tree view is shown and the Data-tab is selected (Figure 2).

The Data-tab contains a data grid with five columns for the x-, y- and z-position, yaw, and tilt. This grid gets filled with the values sent by the clients, hereby requesting the image corresponding to this unique set of values.

After sending back the requested image, this image will be displayed in the View-tab. This way one can see whatever the remote client is seeing. So in a way, the View-tab gives a visual representation of the movement of a client.

Lastly there is the Rooms-tab, in which one can create new virtual environments, referred to as ‘rooms’ in the GUI, by defining a three-dimensional box of coordinates and fill each of these coordinates with the images of a 360 degree spherical panorama. One can use the combo boxes to switch between the different rooms and between the coordinates within each room. There is the option to insert images one by one, or to insert multiple images at once.

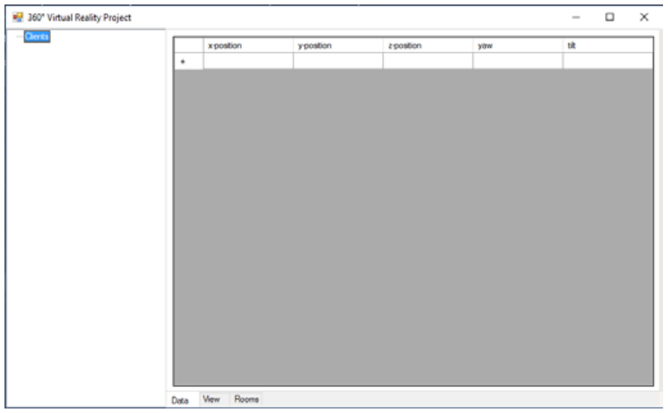


Figure 2. Graphical user interface on startup.

C. Framework

The server has a lightweight framework with only three major classes: the main class, the database class and the Wi-Fi class. Both the database and the Wi-Fi class have a subclass, called image class and client class respectively. The main class provides all functionalities for the GUI, which runs on the main thread. Furthermore, the main class can access the local database via the functions in the database class, and it can start and stop the Wi-Fi communication. This communication is handled in the Wi-Fi class, which runs on a secondary thread. The use of a secondary thread is needed to prevent the GUI to become slow and irresponsive as a consequence of the intensive process of the Wi-Fi communication. The structure of this framework is shown in Figure 3. The arrows in this structure indicate that a class can call functions of, or make changes in another class.

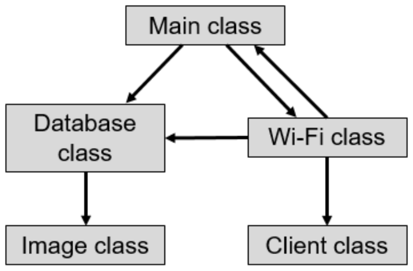


Figure 3. Structure of the server side framework.

1) Main class

The main purpose of this class is to take care of the GUI. This includes updating the tree view when a new client connects, filling the data grid with all incoming data, displaying the last sent image in the View-tab, displaying all images of the selected room and coordinate in the Rooms-tab… Whenever an image needs to be displayed, functions of the database class will be called to retrieve it from the local database. Another function of the main class, next to keeping the GUI up to date, is to use the BackgroundWorker class to start a new thread on which the Wi-Fi class can run.

As mentioned previously, in the Rooms-tab one can create new rooms, and select a room and one of its coordinates. And if selected, one can add images to this coordinate. When adding a single image, one has to specify the yaw and tilt manually for this image. When adding multiple images at once, the software will assign the correct angles to the images automatically. However, this will only work if the filenames are of a the format discussed in III.A, e.g. *'y90t18.png'*.

2) *Database class*

The database class contains all functions to interact with the SQL local database. One function will dynamically, i.e. during runtime, create a new table in the database. This happens when one adds a new room. Two other functions are for retrieving all rooms, and for retrieving all coordinates of one specific room. Furthermore, there are functions to insert an image into a table or to delete one, and functions to retrieve one image, or to retrieve all images at one coordinate.

When retrieving one specific image at a specific coordinate in one of the rooms, all the parameters, i.e. room, x-, y- and z-position, yaw and tilt, are known. However, for displaying all contents of a selected coordinate, all images will be retrieved from the database at once. To display them in the correct order the angles of each image are needed, but the database only contains the raw image data. Therefore an object of the image class will be created for each retrieved image. Every object of type image contains a bitmap image, a value for the yaw, and a value for the tilt. Using these values for yaw and tilt, the images can be displayed in the correct order in the Rooms-tab.

3) *Wi-Fi class*

The Wi-Fi class handles the wireless communication with clients. System.Net.Sockets is used to enable the sending and receiving of messages with the connectionless UDP protocol. After the Wi-Fi class is initialized, it starts to continuously listen for incoming messages. This is a very intensive process which would cause the GUI to become very slow and irresponsive, therefore the Wi-Fi class has to run on a second thread.

Different threads cannot interfere with each other, which means that no changes can be made in the GUI from within the Wi-Fi class. Nonetheless this is needed e.g. when newly received data has to be displayed in the data grid. To make this possible, the graphical elements that need to be updated are passed on from the main class to the Wi-Fi class and used in callback delegates. The Wi-Fi class contains callback functions for updating the tree view, the data grid in the Data-tab, and the picture box in the View-tab.

When receiving a message from one of the clients, this message is analyzed for its content. This content is either a connection request, or motion data to request a new image. In the case of a connection request, it will be checked if the connecting client has already connected once, or not. If not, a new object of type client is created, which takes the clients' IP endpoint and the room that it has been assigned to as its parameters. This object is then added to the servers' list of client objects. If the received message contains motion data, then the requested image is retrieved from the database. After sending some information about the image first, the image itself is sent to the client as a stream of 260 byte packets. The first two bytes are for indicating that the packet contains image data, the two consecutive bytes contain the packet number, and the remaining 256 bytes contain the actual image data. This is equivalent to 127 pixels, as every pixel is of the RGB565 format and thus 2 bytes in size.

IV. CLIENT SIDE

The client side is a wireless embedded system that measures accelerations and rotations to calculate its three-dimensional position and angles and then sends this information to the server. In return the server sends an image which is displayed on the LCD-screen of the client. The embedded system consists of several parts. Physical hardware, a FPGA hardware design with an integrated softcore.

A FPGA hardware design handles the communication protocols for the LCD-screen, Wi-Fi module and a motion tracking device. These modules are part of the physical hardware. A custom IP core is written to handle the initiation of the motion tracking device, data acquisition and generation of an interrupt when the movement exceeds a certain threshold. A softcore is integrated in the FPGA to compute the software.

Software is written to convert the raw data from the motion tracking device to useable data and calculate angles to send to the server. This will be done when an interrupt occurs. The LCD-screen driver and the communication protocol between server and client are another important part of the software.

A. *Physical hardware*

The hardware used for this system consists of four major parts. For the main part of the system a Digilent development board is chosen, the Basys3.

A next important part is the sensor to measure the movement of the system. The InvenSense MPU6050 Six-Axis MEMS MotionTracking™ Device combines an accelerometer and gyroscope to measure the change in movement. To use the MPU a custom IP core is written that connects to an "AXI GPIO" block which connects to the MicroBlaze. The custom core is also connected to the I^2C bus of the sensor.

To make the system wireless, Wi-Fi is added by using a WM-C-MR-09 Wi-Fi module. The module connects through the AXI Quad SPI core to the MicroBlaze. The flash of the Basys3 is necessary to store the firmware for the Wi-Fi module.

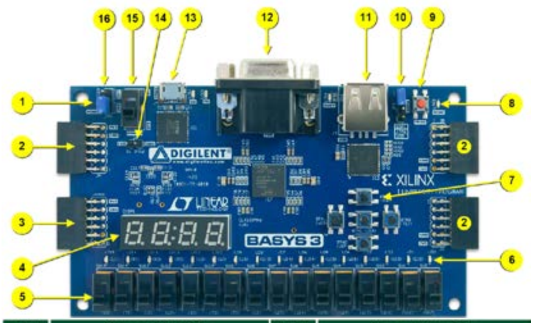
The final part of the system is a LCD-screen to display the pictures sent by the server. The Himax's HX8357-C chipset drives the screen and communicates with the MicroBlaze using an AXI Quad SPI core.

1) *Development board: Digilent Basys3*

The Basys3 is a FPGA development board designed for the Vivado® Design Suite featuring the Xilinx® Artix®-7-FPGA architecture. "The Artix-7 FPGA is optimized for high performance logic [1]." To connect the different modules to the FPGA the Pmod ports are used, port A connects the Wi-Fi module, port B the LCD-screen and port C the motion tracking device. The leds are used to display data and the switches are used to control the system. Pushbutton C (the middle button) functions as the system reset. The usb port functions as the

JTAG programming and UART port. The flash memory of the Basys3 is used to store the firmware of the Wi-Fi module.

In Figure 4 an overview is shown of all the components of the Basys3 development board, and in Figure 5 the physical hardware is displayed.



Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure 4. Basys3 overview.

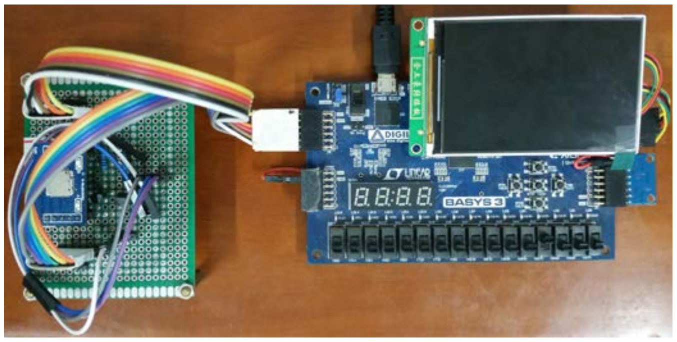


Figure 5. Physical hardware.

2) Motion tracking device: InvenSense MPU6050

To be able to measure motion the InvenSense MPU6050 sensor (Figure 6) is used. The sensor combines two MEMS sensors and a Digital Motion Processor™ (DMP™) hardware accelerator engine in one package. The sensor offers the possibility to interface with third party digital sensors such as magnetometers. When connected to a 3-axis magnetometer, the MPU-6050 delivers a complete 9-axis MotionFusion output to its primary I²C port. The DMP™ can use the data from the accelerometer, gyroscope and magnetometer to accurately compute position, angles, etc. [2].

Figure 6. InvenSense MPU6050.

The use of the magnetometer and DMP™ are not yet implemented due to time constraints. To improve the system these components should be used.

3) Wi-Fi module: WM-C-MR-09

The WM-C-MR-09 Wi-Fi module is based on the Marvell 88W8686 chipset to make wireless communication possible. The module is connected to the Basys3 by a custom designed board to control the power to ensure a correct startup sequence. This is shown in Figure 7. The module is able to communicate through the SDIO protocol which could work faster than the SPI protocol, but unfortunately a license has to be payed to make use of the SDIO protocol, therefore SPI is used. The module complies with the latest IEEE802.11b/g wireless LAN Physical Layer Specification and has a data rate up to 54Mbps by incorporating Direct Sequence Spread Spectrum [3][4].

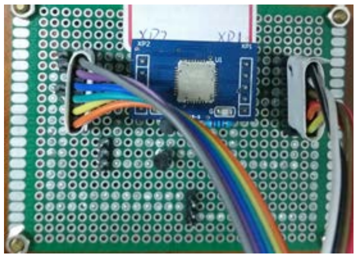


Figure 7. Custom designed board with Wi-Fi module.

4) LCD-screen: Himax's HX8357-C chipset

Himax's HX8357-C chipset controls a TFT-LCD panel with a maximum of 320x480 RGB dots. The chip is designed to provide a single-chip solution to drive a panel. With several interface modes the chip is versatile and it supports HVGA resolution. Using the "DBI TYPE-C Option 3" interface, a serial communication by SPI is achieved to write commands and data to the LCD-screen. Several color modes are available, to decrease the size of data packets the "RGB565" mode is chosen. It is a sixteen bit mode where red is represented by the five most significant bits, green the six following and blue the five less significant bits.

Another feature is the integrated graphical random access memory (GRAM) which stores eighteen bit pixels and therefore consists of 2,764,800 bits (320x18x480bits). When writing to the GRAM the sixteen bit color data is converted to eighteen bit pixel data. The GRAM will automatically adjust the pixel address within the direction and window preferred. A continues data write action is possible which speeds up the communication and the refresh rate [5].

B. FPGA hardware design

When using the Basys3 development board the Vivado® Design Suite is required. The software offers a schematic approach for designing the programmable hardware. With the "IP integrator" tool a block design is made which represents the hardware. The design consists of a MicroBlaze block with a "Processor System Reset", local memory, "MicroBlaze Debug Module (MDM)", "AXI Interconnect" and "AXI Interrupt Controller".

Connected to the "AXI Interconnect" are an "AXI BRAM Controller", three "AXI Quad SPI" blocks, three "AXI GPIO" blocks and an “AXI Uartlite” block. In the design is a custom IP core integrated, the “MPU6050 Controller”. The core connects to one of the “AXI GPIO” blocks, the other signals from the core are connected to external ports in the wrapper. First the custom core will be explained and afterwards the connections of all the “AXI GPIO” blocks.

1) IP core: MPU6050 Controller

To communicate by I²C with the MPU the core integrates a wishbone slave: the “I2C controller core” from Richard Herveille, which can be downloaded at opencores.org [6].

In the wishbone master the “I2C controller core” is initiated, the I²C communication will run at 200kHz and is directly enabled when the hardware is loaded into the FPGA. To control the wishbone master several connections are made to one of the “AXI GPIO”.

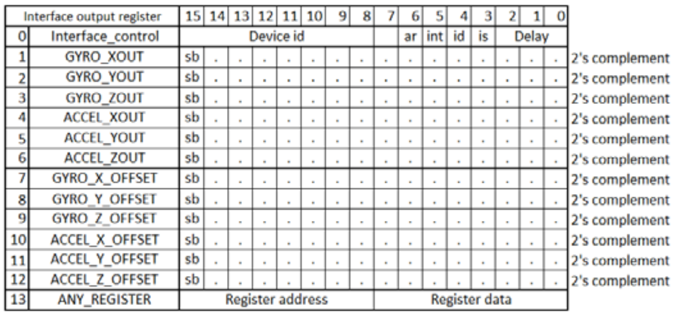
Figure 9 shows an overview of the IP core, and Figure 10 shows all the connections to control the wishbone master.

The control connections are divided in several parts, a first part is the “addr_in”, second and third are the “data_in” and “data_out” busses, then an interrupt output pin “intpin” and a “wr_e” write enable check output. The most significant bit of “addr_in” is the “Write enable” bit, by setting this bit a core register can be written with the sixteen bit “data_in” bus. When the bit is zero a core register can be read on the sixteen bit “data_out” bus. The four remaining bits select the core register to be written or read.

a) Core registers

The core registers are explained in IV.B.1)a), it shows which registers are readable, writeable or both and the format of the registers. IV.B.1)b) explains the core structure; when the different functions are possible and when they are executed. Lastly IV.B.1)c) shows the calculation method of the interrupt.

An overview of the registers is shown in Figure 8. There are four major groups. The first register, register zero, is the “Interface_control” register. Next is a group of output only registers containing the raw data from the six measurements: 3-axis from the gyroscope and 3-axis from the accelerometer. The third group of registers reads or sets the MPU offset registers. The last register, register thirteen, functions as a pass-through register. When reading or writing the “ANY_REGISTER” a certain format and sequence is required.



Interface output register		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	Interface_control	Device id									ar	int	id	is	Delay			
1	GYRO_XOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
2	GYRO_YOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
3	GYRO_ZOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
4	ACCEL_XOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
5	ACCEL_YOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
6	ACCEL_ZOUT	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
7	GYRO_X_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
8	GYRO_Y_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
9	GYRO_Z_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
10	ACCEL_X_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
11	ACCEL_Y_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
12	ACCEL_Z_OFFSET	sb	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2's complement
13	ANY_REGISTER	Register address								Register data								

Figure 8. Core registers overview.

Register “Interface_control” is a sixteen bit control register. The four least significant bits are read- and writeable, the remaining bits only readable. The delay bits set the interval for reading the measurements from the MPU and updating the core registers. In Table 1 the different delay settings are shown.

Read every	
Setting	**Delay (ms)**
0	10
1	50
2	100
3	200
4	250
5	500
6	750
7	1000

Table 1. Delay settings MPU6050 Controller.

When bit “init_start” (is) is set the core starts the initiation process of the MPU. After this initiation is finished and the measurements are ready to read the “init_done” (id) bit is set. It is not possible to reset the “init_start” bit with a write action.

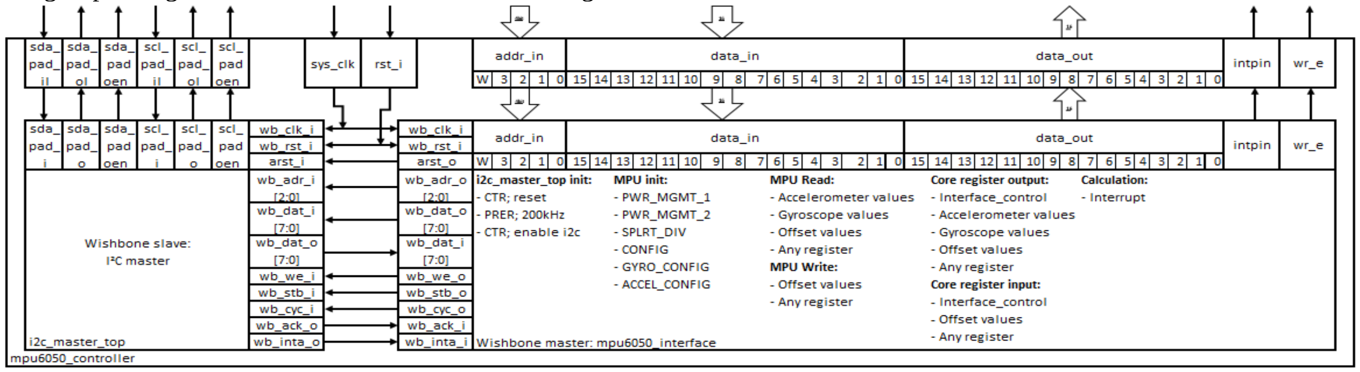


Figure 9. MPU6050 Controller overview.

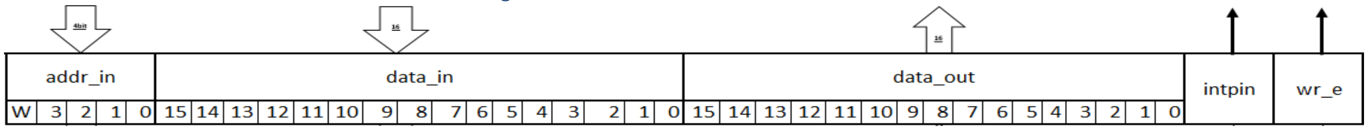

4bit
16
16
addr_in
data_in
data_out
intpin
wr_e
W 3 2 1 0 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 10. Connections to control.

When noticeable movement is detected the “interrupt” (int) bit is set for 1ms, the same pulse interrupt available on the “intpin” output of the control signals. When no interrupts are generated when moving, the system needs to be reset.

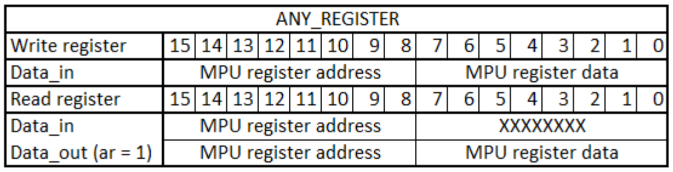
Bit “any_ready” (ar) will be set when the “Register address”, set in register “ANY_REGISTER”, has been read from the MPU and the corresponding data is available in “Register data” of register thirteen. The correct sequence of using register thirteen will be explained later on.

The eight most significant bits are the readout of the “WHO_AM_I” register of the MPU. When this represents “0x68” [7] the communication is up and running, if not there is a fault in the communication or connection of the MPU. When this occurs the system needs to be reset and initiation restarted.

The next group represents the MPU measurement registers of the two MEMS sensors. These registers contain the raw data in a sixteen bit two’s complement format, a value in the range of negative 32768 and positive 32767. The sixteen bit two’s complement format consists of a sign bit (most significant bit) and the remaining bits are the numerical value. To get the numerical negative value a simple conversion is necessary; when the sign bit is set the numerical bits are inverted and incremented by one. The registers are read only and will not be updated when the “Write enable” bit is set.

For setting the offset registers of the MPU another group of six registers is available. The offset registers are not documented in the datasheets, the information is gathered from different MPU6050 Arduino libraries. The most advanced library is from I²Cdevlib. Information about the library, a download link and a complete register map can be found at the I²Cdevlib webpage [8]. Just like the measurement registers, the offset registers are in two’s complement format. The values in these registers will be subtracted from the measurements inside the MPU, correcting the raw output data. It is crucial to calculate the offset when the sensor is level, stable and facing upwards, an example program of an automated offset calculation is found on the I²Cdevlib forum [9].

To write an offset register, which is selected with the “addr_in” bus, the data needs to be presented to the “data_in” bus before setting the “Write enable” bit. When setting the bit, the write action happens exactly one time, and the address of the last written register is stored. After a change in address or a read action of the same address a new write action is possible. The core offset registers will only change by reading the MPU registers, while performing the write action the MPU register is directly written. This way it is possible to check if the write action was successful.



ANY_REGISTER																
Write register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data_in	MPU register address								MPU register data							
Read register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data_in	MPU register address								XXXXXXXX							
Data_out (ar = 1)	MPU register address								MPU register data							

Figure 11. ANY_REGISTER read/write structure.

The last core register makes it possible to read or write every register in the MPU. Figure 11 shows the “data_in” and “data_out” structure when reading or writing the register.

When reading a certain register the “data_in” bus needs to be set before choosing register thirteen. The eight most significant bits represent the MPU register to read. When the “any_ready” bit is set in the “Interface_control” register, the “data_out” bus will hold the corresponding data. A correct sequence to read a MPU register is the following: first set the “data_in” with the MPU register address, select address thirteen, and check the “any_ready” bit in the “Interface_control” register. When this bit is set, read the “data_out” bus after selecting register thirteen. Repeat the sequence to perform a next read.

For writing a MPU register the “data_in” bus needs to be set before selecting register thirteen. The eight most significant bits represent the MPU register to write and the eight least significant bits represent the data to write into the register. Before selecting register thirteen the write bit needs to be enabled. The correct sequence is as follows: select a not writeable core register, set “data_in” with the MPU register address and data, set the “Write enable” bit and finally select register thirteen. The write action will only happen once, to write another register the sequence needs to be repeated.

b) Core structure

To explain the hardware design Figure 12 shows a simplified structure. Keep in mind that hardware runs continuously and parallel, so every action is dependent on a specific bit that is either set by an external connection or a different part of the design. The bit needs to be reset after the action is completed.

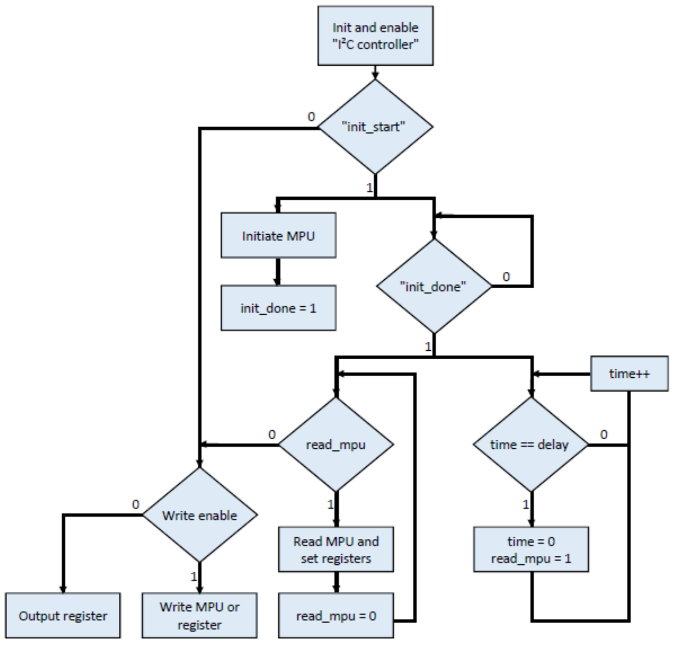


Figure 12. MPU6050 Controller structure.

The first block in the design initiates and enables the “I2C controller core” from Richard Herveille. This happens directly after the bit stream is loaded or the hardware is reset.

The following action depends on the “init_start” bit of the “Interface_control” register. When the bit is not set, the hardware keeps performing the “check_addr_in” function. It is the left side of the structure. When the “Write enable” bit is not set, the register selected on the “addr_in” bus is set on the “data_out” bus. When the bit is set only the “Interface_control” writeable bits can be written. After setting the “init_start” bit the core will initiate the MPU.

The “initiate MPU” block will perform following instructions: set the internal 8MHz oscillator and disable the temperature sensor in the “PWR_MGMT_1”register, fill the “PWR_MGMT_2” register with zeros, sets the digital low pass filter to 42Hz in the “CONFIG” register. Also the ranges for the gyroscope and accelerometer are set to 500°/s and 2g in the “GYRO_CONFIG” and “ACCEL_CONFIG” registers, a sample rate of 100Hz is chosen by setting the “SMPLRT_DIV” register to nine, and to confirm a proper connection the “WHO_AM_I” register is read. The “init_done” bit is set in the “Interface_control” core register when the initiation is finished.

When the “init_done” bit is set the main part of the core is activated. A counter is started, which increments every millisecond, and when the counter equals the selected delay the “read_mpu” bit will be set and the counter is reset zero.

While the “read_mpu” bit is not set the “check_addr_in” function is executed with all the functionality. It still depends on the “Write enable” bit; when not set the interrupt will be calculated and the chosen register outputted on the “data_out” bus. When the “Write enable” bit is set all the writeable registers can be written by selecting the address of a register. When writing a register other than the “Interface _control” register a bit is set to prevent a simultaneous read and write action of the MPU registers.

When the “read_mpu” bit is set, MPU registers are read (if the above mentioned write bit is not set). The following registers are read in this order: a burst read of the gyroscope out register, a burst read of the accelerometer out registers, a burst read of the gyroscope offset registers, a burst read of the accelerometer offset registers and if necessary the register requested by reading core register thirteen. If all these registers are read from the MPU the “read_mpu” is reset.

c) Interrupt generation

An interrupt is generated to prevent the loss of useable samples or reading the same sample several times. The next code snippet shows the calculation within the “check_addr_in” while the “Write enable” bit is zero.

```
if(init_done)
begin
    // if sensor doesn't rotate the gryo's values stay 0 (SB + 5MSB's 1 or 0)
    if((GYRO_XOUT[15:9] != 7'h7F && GYRO_XOUT[15:9] != 7'h0) ||
       (GYRO_YOUT[15:9] != 7'h7F && GYRO_YOUT[15:9] != 7'h0) ||
       (GYRO_ZOUT[15:9] != 7'h7F && GYRO_ZOUT[15:9] != 7'h0))
        interrupt <= 1'b1;
    if(interrupt == 1'b1)
    begin
        if(!Count2)
            is2En <= 1;
        else if(Count2 == T1MS )
        begin
            is2En <= 0;
            interrupt <= 1'b0;
        end
    end
end
```

Code snippet 1. Interrupt calculation.

The interrupt generates a pulse of one millisecond so the software can perform a single read operation from the registers. To calculate the interrupt the core depends on the gyroscope data. The gyroscope outputs the change in degrees over time, and when there is no change the measurements are zero. Because there is always noise in the measurements a low pass filter is applied by checking the six most significant bits of every axis. The data is in two’s complement therefore all the bits need to be zero or one when the device is stable. If the bits are not all zero or one the device is moving.

2) AXI GPIO connections

To be able to control external signals with the MicroBlaze “AXI GPIO” blocks are used. In the hardware design three blocks are connected to several signals or cores. Multiple channels are possible for every block, where channel one is always an output and channel two is always an input. All three blocks are explained below.

a) axi_gpio_led_sw

This block connects the 16 bit leds and 16 bit slide switches of the Basys3. The leds are used to output the register status selected with the five most significant bits. The three least significant bits are to set the delay. The function described is not set by hardware so it can be changed in the software.

axi_gpio_led_sw																															
15	Channel 1 output														0	15	Channel 2 input														0
Leds																Slide switches															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 13. axi_gpio_led_sw connections.

b) axi_gpio_lcd_wifi

To control the LCD-screen and Wi-Fi module this block connects to three signals of the LCD-screen and two of the Wi-Fi module. Figure 14 shows the bits corresponding to the signals.

axi_gpio_lcd_wifi				
4	Channel 1 output			0
LCD			Wi-Fi	
BL	RS	SS	RST	SS

Figure 14. axi_gpio_lcd_wifi connections.

The “BL” connection controls the backlight of the LCD-screen. “RS” controls the type of data sent to the LCD; when set to zero a command is sent and when set to one data is sent. “SS” is the SPI slave select to select the LCD-screen. For the Wi-Fi module the “RST” controls the power on state and the “SS” is the SPI slave select of the Wi-Fi module.

c) Axi_gpio_mpu6050

The third block connects to the MPU6050 Controller core, using Figure 15 and Figure 16 the connections are explained. The input channel contains eighteen bits and the output channel twenty-one.

axi_gpio_mpu6050																				
20	Channel 1 output																			0
addr_in					data_in															
W	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 15. axi_gpio_mpu6050 output connections.

The output channel is used to select the core register address to read or write and to send data to the controller. The five most significant bits are the “addr_in” bits and the remaining the “data_in” bus.

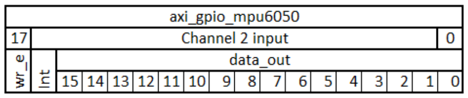


Figure 16. axi_gpio_mpu6050 input connections.

The most significant bit “wr_e” of the input channel is the write enable check output, bit sixteen is the generated interrupt and the remaining bits are from the “data_out” bus.

C. Software

When making an embedded system software is needed to run within the controller core. The “Xilinx Software Development Kit” or SDK is used for the software on the MicroBlaze. The hardware is exported from Vivado and SDK is started to create a new application project. The project is written in C language.

As shown in Figure 17 the source of the project is divided in several folders, this will keep a good overview. When a change is needed in the program the files are found easily.

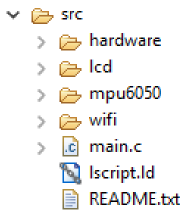


Figure 17. Application project overview.

1) Hardware

The hardware folder contains the settings of the hardware and the specific functions to initiate and control the hardware. There are three files as shown in Figure 18.

Figure 18. Hardware folder.

In “hardware_config.h” all the hardware connections are defined, this way the program can be adjusted quickly if there are any hardware changes in the future.

In “hardware.h” the several libraries are included to work with the hardware, by doing this all the needed libraries are included by including “hardware.h” in a source file. Another function of this file is to make all the instances, necessary to work with the functions, global. By adding “extern” in front of the instance it can be accessed from every source file where “hardware.h” is included and the instance is instantiated. When using UART to debug or output useful data, several defines can help to determine what the output must be. The defines “debug”, “output_data” or “debug_wifi” can be chosen or put in comment.

The file “hardware.c” contains all the functions to initiate and control most of the hardware.

2) LCD-screen

The files in the lcd folder are used to control the LCD-screen. As shown in Figure 19 only two files are in the folder.

Figure 19. Lcd folder.

The same method is applied as with the “hardware.h”, all the includes and defines are in the “lcd.h” file. The used registers of the Himax’s HX8357-C are defined along with several sixteen bit colors.

The functions to communicate with the LCD-screen, initiate and control it are in “lcd.c”. The functions are based on the “Universal TFT and other display device library for the chipKIT and PIC32 based boards [10].“ from Majenko Technologies. This library was deprecated and instead the “DisplayCore” library [11], [12] can be used as a reference in the future.

3) MPU6050

To control the MPU6050 Controller core two files in the mpu6050 folder are used. The files are shown in Figure 20.

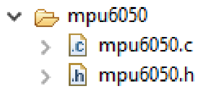


Figure 20. Mpu6050 folder.

For good practice the same method is used in the “mpu6050.h” file, all the previous files are written specifically for this project therefore the same methods are applied.

Some of the functions in “mpu6050.c” are explained in what follows: IV.C.3)a) explains the conversion from the raw data to useable data and IV.C.3)b) the motion calculation.

a) Raw data conversion

To use the data from the MPU it needs to be converted because the data outputted by the MPU is data from the internal analog to digital converter. Depending on the full scale range used for the application the raw data needs to be divided by a sensitivity level. For the gyroscope in a full scale range of 500°/s the sensitivity level is 65.5 and for the accelerometer in the 2g range 16384.

As mentioned before there will be an offset on the data. Example code [9] to set the offset in the MPU registers was implemented, however by experimentation this method turned out to be not useful due to a long calculation time and low success rate. If it was working, there was no need to calculate the offset in software and subtract it from the raw data.

To calculate the offset in the software the raw data is read for a certain number of times, the first hundred times are discarded then it will read the data again for the chosen number of times and calculates the average. For the three gyroscope axis and the accelerometer x- and y-axis the raw data needs to be zero after subtraction of the offset. For the accelerometer z-axis the value of the raw data needs to be equal to the sensitivity level because the gravity is working only on the z-axis when level and stable.

Although it is not possible to fully level a sensor, the user can set a tolerance for every axis.

Code snippet 2 shows the code that is eventually used, the useable data for the gyroscope is in degrees per seconds and for the accelerometer in terms of gravitation which equals to 9.81m/s².

```
void ConvertAllToDegreesPerSec()
{
    degree_s_x = (float)(mpu_gyro_x_raw - offset_gyro_x) / sensitivity_gyro;
    degree_s_y = (float)(mpu_gyro_y_raw - offset_gyro_y) / sensitivity_gyro;
    degree_s_z = (float)(mpu_gyro_z_raw - offset_gyro_z) / sensitivity_gyro;
}

void ConvertAllToGravity()
{
    gravity_x = (float)(mpu_accel_x_raw - offset_accel_x) / sensitivity_accel;
    gravity_y = (float)(mpu_accel_y_raw - offset_accel_y) / sensitivity_accel;
    gravity_z = (float)(mpu_accel_z_raw - offset_accel_z) / sensitivity_accel;
}
```

Code snippet 2. Raw data conversion.

b) Motion calculation

To calculate proper values which can be used to acquire pictures some algorithms are needed. Data presented by the MPU will not be accurate enough, drift on the gyroscope and noise on the accelerometer are the cause. The calculations consist of two parts: the calculation of angles and calculation of coordinates. Due to time constraints the coordinates calculation is not implemented and needs more research for further expansion of the system. Theoretically a double integration of the acceleration presents the distance travelled, however because of the noise it is not that simple.

The calculation of angles is not fully implemented due to memory constraints of the Basys3, additional BRAM is already used for the software. To do the proper calculations the library "Math.h" needs to be included which is too large, a simplified calculation is used and is not as accurate as the preferred method that is explained below. Only the data from the gyroscope is used, it is not the best option because of the drift from the gyroscope.

There are several options for calculating angles, an algorithm which can be used is the Kalman Filter. This algorithm calculates an estimate based on the presented samples, which is more accurate when not all samples are correct. A drawback of this algorithm is the high processor load and difficult implementation because of the complex mathematics. Another algorithm is the Complementary filter (Figure 21), it uses the data from both sensors and cancels the high frequency noise of the accelerometer and the low frequency drift of the gyroscope.

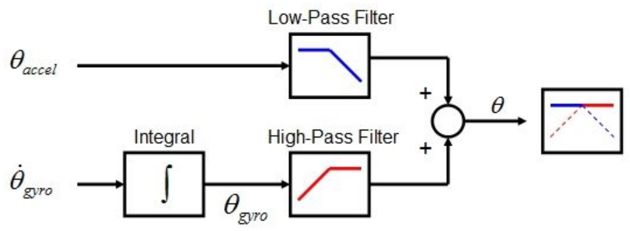


Figure 21. Complementary filter.

The algorithm uses a low-pass filter for the angles calculated from the accelerometer data to cancel the noise. A high-pass filter is used for the angles calculated from the gyroscope. This removes the drift. By adding the two calculated and filtered angles an accurate angle is calculated.

Using some trigonometry the angles are calculated from the accelerometer, to calculate the tilt and roll the code from Code snippet 3 will be executed.

```
tiltAccel = atan(gravity_x/sqrt(pow(gravity_y,2) + pow(gravity_z,2)));
rollAccel = atan(gravity_y/sqrt(pow(gravity_x,2) + pow(gravity_z,2)));
```

Code snippet 3. Calculating angles from accelerometer data.

The yaw cannot be calculated from accelerometer data, because no change in gravity is measured when the device turns in the horizontal plane. To use an additional measurement for the yaw a magnetometer needs to be used.

To calculate angles from the gyroscope data, the degrees per second need to be converted to degrees. This is done with an integral, in code it is nothing more than adding the new measurement to the previous measurements keeping in mind it is not measured during one second but only one sample period of ten milliseconds. Code snippet 4 shows the code used.

```
tiltGyro += degree_s_y * samplerate;
rollGyro += degree_s_x * samplerate;
yawGyro  += degree_s_z * samplerate;
```

Code snippet 4. Calculating angles from gyroscope data.

To apply the low-pass and high-pass filters the two angles will be added, each with a multiplication factor. The angle of the gyroscope has the highest factor because of the high-pass filter, for example 0.98. The sum of the factors need to be one. The yaw is only calculated with the angle of the gyroscope and a high-pass filter. Code snippet 5 shows the calculation of the tilt, roll and yaw.

```
tilt += factor*(tiltGyro)+(1-factor)*tiltAccel;
roll += factor*(rollGyro)+(1-factor)*rollAccel;
yaw  += factor*(yawGyro);
```

Code snippet 5. Calculation of tilt, roll and yaw.

The server expects certain angle values therefore a last calculation is needed. Code snippet 6 shows the calculation of the expected values.

```
if(tilt<-90)
{
    tilt = -90;
}
if(tilt>90)
{
    tilt = 90;
}
if(yaw < 0)
{
    yaw += 360;
}

if(yaw > 359)
{
    yaw -= 360;
}

tilt_out = tilt/18;
tilt_out *= 18;
yaw_out = yaw/18;
yaw_out *= 18;
```

Code snippet 6. Calculation expected angle values.

4) Wi-Fi

This part of the code is developed in previous projects of the Yangzhou University research team, however the file "v360_protocol.c" is added to implement the communication protocol and display the images on the LCD-screen. The received images are displayed in the center of the screen, the server can send an image with a variable size up to the size of the LCD-screen. When packets with raw image data are received, the data is written directly into the GRAM of the LCD-screen. The server and client work in a peer-to-peer connection because the Wi-Fi module is working in AdHoc mode secured by WEP.

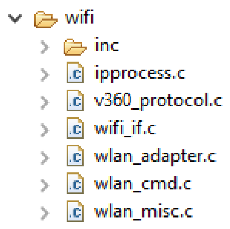

```
wifi
  inc
  ipprocess.c
  v360_protocol.c
  wifi_if.c
  wlan_adapter.c
  wlan_cmd.c
  wlan_misc.c
```

Figure 22. Wifi folder.

5) *Main*

When the program starts, the hardware is initiated and when successful the LCD-screen is initiated followed by the Wi-Fi module. During the initiation of the Wi-Fi module the interrupt handler starts to receive and process data from the module. The next part sets up the AdHoc network with the ssid "VR360System" and keyascii "1234567890123". When the server is detected it makes the connection and the program continues. To start the MPU6050 Controller core switch three is set high. When the MPU is started a "init_done" check is performed, if successful the interrupt handler for the MPU is activated. It reads the core registers and calculates the angles. After the activation of all the hardware and the connection to the server is made the systems starts requesting images when movement is detected.

V. Wi-Fi Communication Protocol

The communication between server and clients follows a lightweight protocol (Figure 23) with only a few steps.

For starters, after a client has initialized its Wi-Fi module, it will send an ARP message to request the servers' MAC-address. Once the client has received the MAC-address the communication can begin, so the client sends a message containing the word "connect". On receiving this connection request, the server replies with "connok" to tell the client that the connection is ok, and adds the client to its list of clients if it is a new client. The client can now start sending its coordinate and angles, hereby requesting the corresponding image. Next the server retrieves the correct image from the database and sends back some information about this image. This information message comprises the size of the image, the number of packets needed to send to entire image, the width of the image, and the height.

If the reception of this information has been acknowledged by the client, the server starts sending a stream of packets holding the raw image data. The client displays the comprised data directly on the LCD-screen. The last packet of the stream contains a unique string of cyphers indicating the end of the image transmission. When this end-of-file (EOF) string has been received, the client knows that the server has finished sending image data so it will check if a new image needs to be requested.

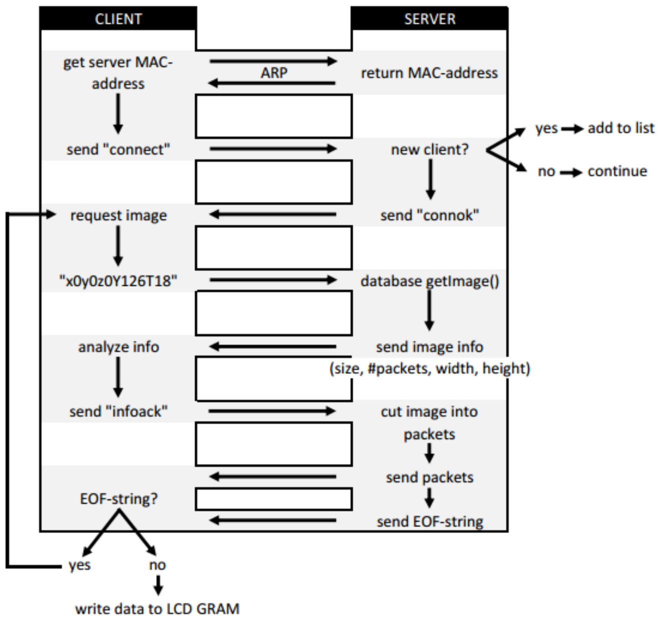


Figure 23. Wi-Fi communication protocol.

VI. Future Work

The created system is a functional proof of concept which can be improved on multiple areas. On the server side some improvements can be made by creating a full screen functionality of the GUI and adding several skins, for example by using Gnome. A function to change the dimensions of an existing virtual environment is another improvement for the server. Improvements on the client side can be made by changing the network type with the purpose to connect several clients to one server and improve the Wi-Fi stability. When the network type is changed the method of writing image data to the GRAM also needs to be changed. The current method writes the data in the order it is received, however with a larger network the packets can be in a mixed order, therefore the GRAM needs to be written according the packet number. To improve the angle accuracy a magnetometer must be added and the DMP^{TM} must be used to calculate the angles, this can help to improve the overall speed of the system and free some memory. Another way to improve the speed is by speeding up the reading process of the MPU. The last thing which must be done is the calculation of the position. With all these improvements the system can function as a smooth virtual reality system.

VII. Conclusion

This wireless 360° virtual reality system is a functional proof of concept consisting of a simple and lightweight server framework and an embedded client system using a FPGA.

The server uses multithreading to simultaneously update the GUI when communicating over Wi-Fi with the client. This gives a smooth experience on the server side. The server has three important classes and the protocol used to communicate is lightweight and easy to understand.

The client uses programmable hardware to communicate with the MPU6050 and integrate a MicroBlaze controller to run software. When working with a MEMS accelerometer and gyroscope algorithms are required to use the inaccurate data of the sensors and send accurate angles to the server. Because of the complexity of the algorithms and communication over Wi-Fi the accessible memory on the hardware was too low. The communication of the LCD-screen and overall speed are too slow for a smooth experience one the client side.

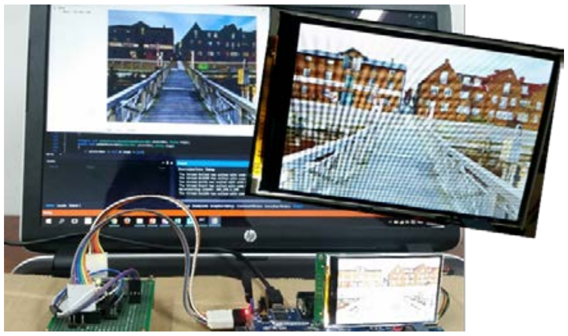


Figure 24. Wireless 360° virtual reality system.

REFERENCES

[1] Digilent Inc, "Basys 3 TM FPGA Board Reference Manual Power Supplies," vol. C, pp. 1–19, 2016.

[2] InvenSense Inc, "MPU-6000C and MPU-6050C Product Specification," vol. 1, no. 408. 2012.

[3] C. Chen, "Preliminary Product Specification of WM-G-MR-09," vol. 1.7, no. 00001, p. 24, 2007.

[4] Marvell, "Marvell 88W8686 datasheet," vol. D, p. 190, 2007.

[5] Himax, "HX8357-C_DS," 2012, pp. 1–254.

[6] R. Herveille, "opencores.org/project,i2cslave," p. Webpage, 2013.

[7] InvenSense Inc, "MPU-6000 and MPU-6050 Register Map and Descriptions," vol. 1, no. 408, pp. 1–50, 2011.

[8] J. Rowberg, "http://www.i2cdevlib.com/devices/mpu6050#source," p. Webpage, 2014.

[9] L. Ródenas, "http://www.i2cdevlib.com/forums/topic/96-arduino-sketch-to-automatically-calculate-mpu6050-offsets/," 2014.

[10] Majenko Technologies, "https://github.com/TFTLibraries/TFT," p. Webpage, 2015.

[11] Majenko Technologies, "https://github.com/MajenkoLibraries/DisplayCore," p. Webpage, 2016.

[12] Majenko Technologies, "http://displaycore.org/," p. Webpage, 2015.