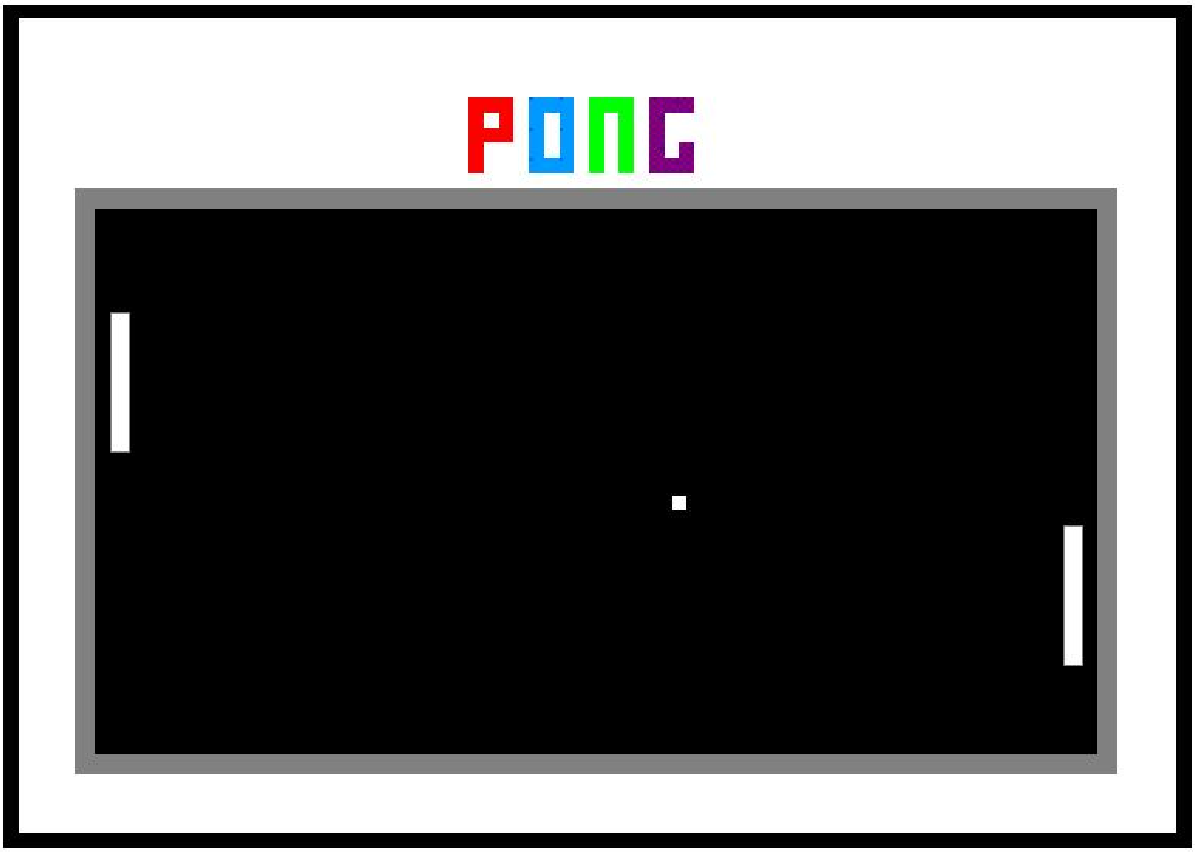
EE518 Advanced Microcontrollers

Pong Project



Group 3:

Maxime Cassan

Pauline Castets

Philipp Karagiannakis

Florent Rioult

Contents

Table of Figures

1 Introduction

1.1 Specification of Design

The game of PONG was developed in the 1970's and is an electronic version of table tennis. The game features two paddles placed at opposite ends of the screen. A ball is hit between these two paddles. The aim of the game is to defeat the opponent by hitting the ball past their paddle, when this occurs the player's score is incremented by one and the game starts again. The first player to score 9 is declared the winner after which the scores are reset and a new game begins.

The aim of this project was to create the PONG game utilising a M16C microcontroller; it would control all aspects of the game: game physics, VGA display and user controls. Some additional features such as game sounds, a computer controlled opponent, a menu system, colour display and a current score meter have been implemented. In order to achieve the aim of the project several key objectives were identified in the design process. These are listed below:

- Create signals to drive VGA display
- Define game physics
- Implement control method for paddles
- Integrate sounds
- Create computer opponent
- Mode select

1.2 Report Structure

This report begins by providing the reader with some background knowledge on creating an image on a VGA monitor. The game of Pong has been copied and recreated on many different occasions; this has led to various versions of game play. A brief description of the game physics used in the final design completes the background section. A bill of materials defines all the components used and these are described further in a hardware description section. The software developed to control the various aspects of game play, the display and additional features are clearly explained in the software description section. In both the hardware and software descriptions testing methods are defined which make it easier to find faults and repair them. A complete set of instructions for using the developed Pong game follow. Finally the report concludes with a discussion on problems occurred during construction and touches on future work which could be carried out in order to improve the game. A complete circuit diagram of the design can be found in Appendix A. A component layout diagram can be found in Appendix B and makes identifying the components easy.

2 Background

2.1 Creating VGA images

2.1.1 VGA Timing signals

A single dot of colour on a video monitor does not impart much information. A horizontal line of pixels begins to carry something meaningful. However, it is when we combine these horizontal lines into a frame that we are able to create an image on a screen. In VGA terms a frame typically consists of 480 horizontal lines each made up of 640 pixels. To paint a frame there are deflection circuits in the monitor that allow the screen to be painted from left to right and from top to bottom. These deflection circuits require two synchronisation signals which start and stop the deflection circuits at the correct times. This ensures that an image is formed by painting a line of pixels across the monitor and that these lines are stacked up from top to bottom. The timing for the horizontal VGA synchronisation signals is shown below in figure 1.

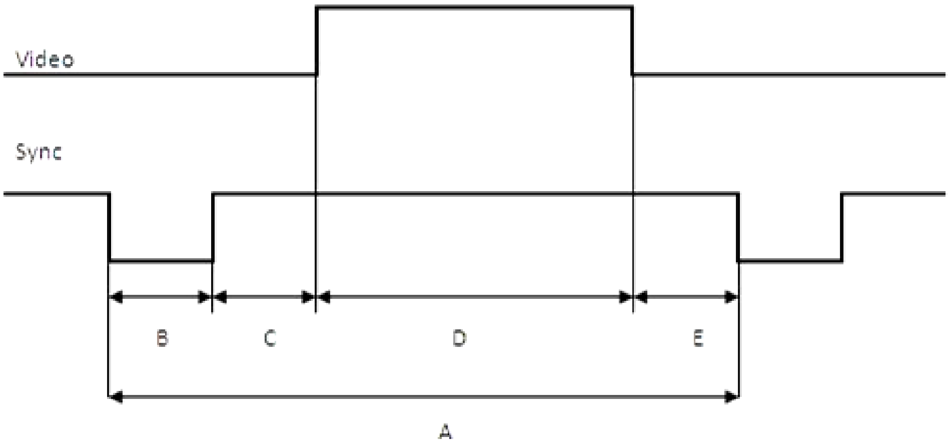


Figure 1: VGA Horizontal synchronisation timing

The low pulses on the horizontal sync signal indicate the start and end of a line and ensure that the monitor displays the pixels between the right and left edges of the visible screen. The actual pixels are sent to the monitor within a 25.17 µs (D) window. The horizontal signal drops low a minimum of 0.94 µs (E) after the last pixel and stays low for 3.77 µs (B). A minimum of 1.89 µs(C) after the horizontal signal goes high again a new line of pixels can begin. A single line of pixels occupies 25.17 µs of a 31.77 µs (A) interval. The other 6.6µs of each line is the horizontal blanking period during which the screen is dark.

In a similar fashion the negative pulses on the vertical sync mark the start and end of a frame. Each frame is made up of 480 video lines as well as some additional blanking periods. This ensures that the image is displayed between the top and the bottom edges of the visible screen. The lines are sent to the monitor within a 15.25 ms (R) window. After which the vertical sync signal goes low a minimum of 0.35 ms (S) after the last line. It stays low a total of 0.06 ms (P); this indicates to the monitor that a new frame is about to begin and instructs it to start at the top left corner again. A minimum of 1.02 ms later the first new line can be sent to the monitor. So a single frame occupies

15.25 ms of a 16.68 ms interval. The rest of the time is used for the vertical blanking interval during which the screen is dark. The timings for the VGA vertical synchronisation pulse and relative video output are shown in Figure2.

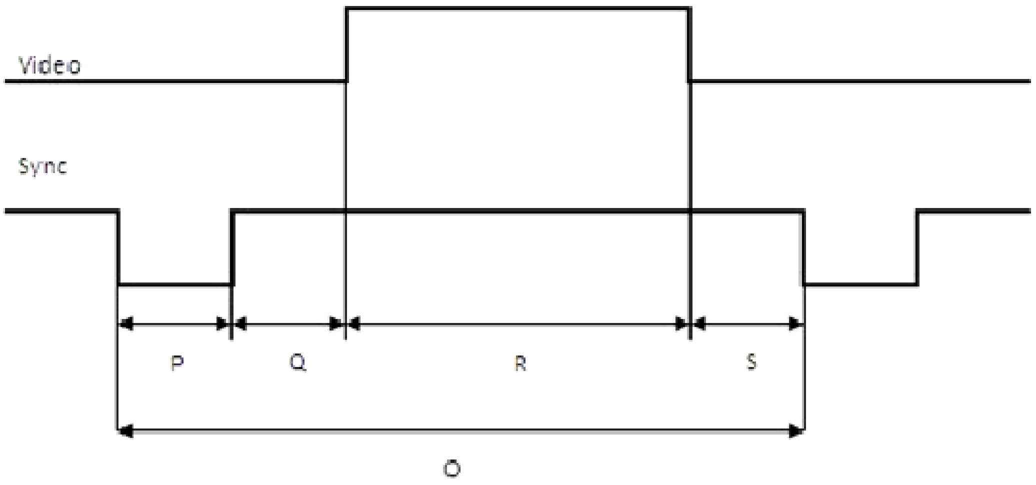


Figure 2: VGA Vertical synchronisation signal timing

2.1.2 VGA Colour Signals

Three signals are used to send colour information to a VGA monitor – Red, Green and Blue. On a CRT screen these three signals each drive an electron gun that emits electrons. Each gun paints one primary colour at a point on the monitor screen. Analogue levels between 0 V (completely dark) and 0.7 V (maximum brightness) on the control lines tell the monitor what intensities of these three primary colours to combine to make the colour of a dot (or pixel) on the monitor's screen.

2.2 Game Physics

The simple game of Pong requires that a ball be hit between two paddles placed at opposite sides of a screen. In order to achieve this goal a set of rules which govern the movement of the paddles and ball must be adhered to. The paddles are required to move horizontally up and down on the sides of the screen and must not be able to move past the borders of the screen – created by the top and bottom of the screen.

The ball is free to move in any direction. However, when it comes into contact with either player's paddle or the top or bottom border it must bounce off. In the case of the ball hitting the wall, the ball is simply reflected at an equal but opposite angle and ball speed is unaffected. When the ball hits one of the paddles, it experiences a change in direction and speed. The angle of reflection and the change of speed are determined by where the ball makes contact with the paddle. Figure 3 below illustrates the results of the ball making contact with different parts of a player paddle.

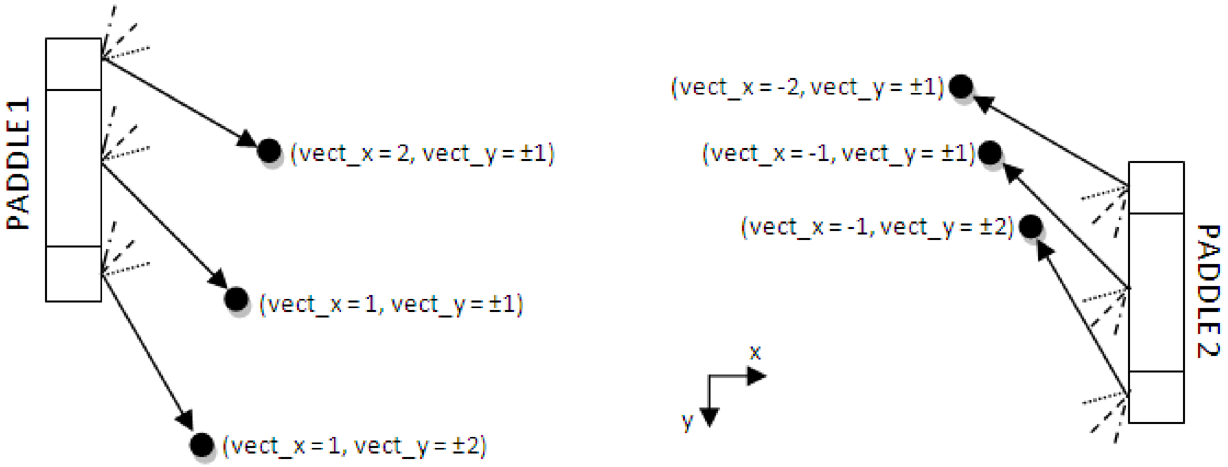

PADDLE 1
(vect_x = 2, vect_y = ±1)
(vect_x = 1, vect_y = ±1)
(vect_x = 1, vect_y = ±2)
(vect_x = -2, vect_y = ±1)
(vect_x = -1, vect_y = ±1)
(vect_x = -1, vect_y = ±2)
x
y
PADDLE 2

Figure 3: The various figures illustrate the movement of the ball once it has made contact with one of the player paddles.

If the ball hits the middle of the paddle, its new direction of travel is diagonal (vect_x = vect_y), making it easy for the opponent to return. On the other hand, if the ball hits the top or bottom of the paddle, the speed is increased, either in the x-axis, or in the y-axis, making it harder for the opponent to return. Thus, if the player wants to win, he would have more success trying to hit the ball with either the top or bottom edges of his paddle. However, this is not without its dangers as it is easy for the player to miss the ball completely and lose a point. These basic rules make the game more challenging and they enhance the quality of game play.

3 Description of Hardware

A bill of materials is listed below and some of the hardware structures which are used in the design are described further in this section:

- M16C62P Microcontroller development board (x1)
- 75 Ω (x3)
- 470 Ω (x3)
- Piezo-electric transducer (Buzzer) (x1)
- 10k Ω Slide potentiometer, linear taper (x2)
- 10k Ω Slide potentiometer, logarithmic taper (x1)
- Push to make switch: two way (x2), one way (x1)
- Blue LED (x1)
- VGA connector PCB with test points (x1)

3.1 Renesas RSKM16C62P Starter Kit

The Renesas development starter kit provides a platform from which many aspects of the M16C microcontroller can be tested. It operates with a 6 MHz clock which through the use of a phase lock loop can be increased to 24 MHz which makes it suitable for use in the design. It has 512 kbytes of read only memory (ROM) and 31 kbytes random access memory (RAM). All relevant technical documentation can be found on the Renesas website [1].

3.2 Piezoelectric transducer (buzzer)

The Piezoelectric transducer generates a range of audible tones and frequencies when energised by a square wave. It is driven direct from the microcontroller board using a pulse width modulated (PWM) signal. The creation of the signal is described in the software description section and the physical setup is shown in Figure 4. A variable resistor with a logarithmic taper was implemented as volume control.

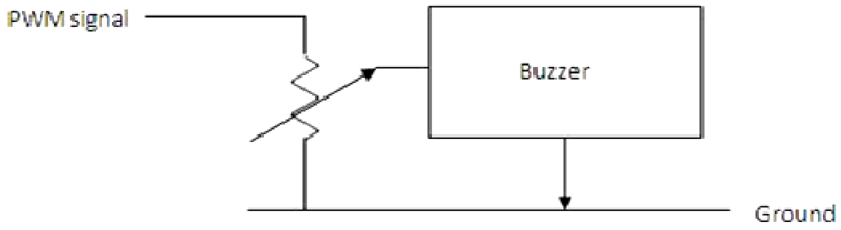


Figure 4: Figure shows how the slide potentiometer is used as volume control

3.3 Slide potentiometer

There are two types of slide potentiometers used in the design: logarithmic taper and linear taper; they are used for volume control and player paddle control respectively. The two different types of potentiometers are setup in a similar fashion. They each have three pins: Vcc, ground and Vref. In the case of the player paddle slide control, the Vcc and ground are connected to the Vcc and ground of the microcontroller board. The Vref pin is connected to an input pin for one of the Analogue to

Digital Converters (ADC) found on the microcontroller board. As the slide moves up and down the voltage across the variable resistor is measured and converted into a digital form which can be used by the microcontroller.

The sliding potentiometer used to control the volume is connected a little bit differently. The Vcc input is now connected directly to the PWM output pin coming from the board. Ground is once again connected to ground and Vref is now connected to the input for the buzzer. This effectively controls the voltage level of the square wave used to excite the buzzer and thus the volume can be controlled. The way in which these potentiometers are connected is shown in Figure 5 below.

It is important to identify the potentiometer's Vcc and ground pins and wire them correctly. This is to prevent a short circuit occurring when the slide is at zero resistance. A simple method to identify the pins of the potentiometer is to measure the resistance between pins. The resistance between the Vcc and ground pins of the potentiometers is always constant. When testing Vcc should read +5V and the output to the ADC input pin varies between 0V to +4.5 V.

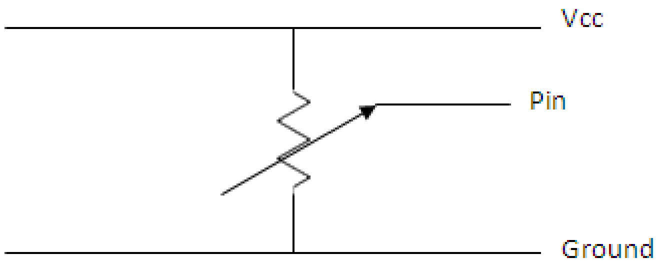


Figure 5: Wiring configuration for sliding potentiometers

3.4 Push to make switch

Both one and two way push to make switches are used in the design. They are both set up in a similar fashion and operate in the same way. The pin of the microcontroller is connected between the ground and one end of switch. The other side of the switch is directly connected to Vcc. The pin reads 0 V until a switch is depressed at which point it goes high to Vcc. This is registered at the microcontroller input and the appropriate action is taken. Their physical setup is shown below in Figure 6.

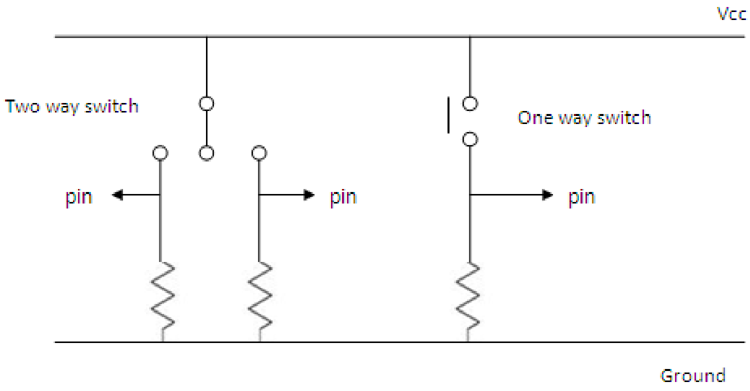


Figure 6: Push to make switch wiring method

3.5 VGA Digital to Analogue Converter

Each analogue colour input can be set to one of two levels by the digital output from the microcontroller using a simple two bit digital to analogue converter (DAC). The DAC is made up from a simple voltage divider circuit created by the two different value resistors mentioned in the bill of materials; this is shown in Figure 7. The voltage level of the digital signals is a Transistor Transistor Logic (TTL) high of 5 V and once it passes through the voltage divider circuit it is attenuated to the max of 0.7 V as referred to in the background section. The two possible levels on each analogue input are combined by the monitor to create a pixel with one of 2x2x2 = 8 different colours. Therefore the 3 digital control lines allow us to choose from a palette of 8 colours.

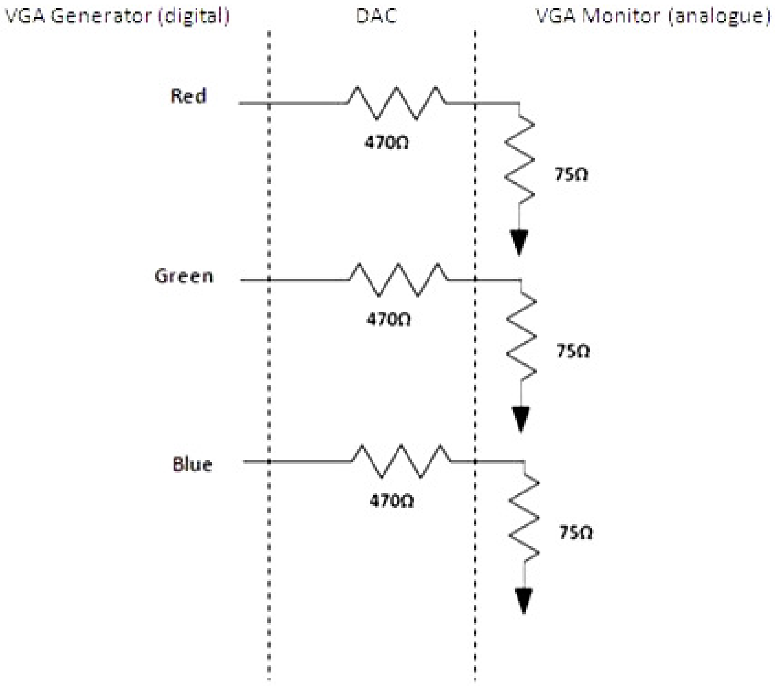


Figure 7: Two bit digital to analogue conversion (DAC)

A printed circuit board which represents the above DAC was made and the layout of the PCB can be found in Appendix C. The PCB was made to allow for easier testing of the VGA signals with the test points clearly labelled on the schematics of the circuit board.

4 Description of Software

The software description is broken down into two main sections: Device Drivers and Game Code. The device driver section explains how both synchronisation signals are created, how the screen resolution is defined as a two dimensional array and how an interrupt assembler routine uses this array to output the desired image. The PWM signal used to create the sounds for the game is created using one of the timers on the microcontroller and the methods used to set it up as well as how to adjust the output frequency and pitch completes the device driver section. The game code section describes the flow of the game physics code and how the rules of play are were implemented. A full listing of the code is available on the CD-ROM which is attached to this report.

4.1 Device Drivers

In order to create the synchronisation signals and send the image information to the VGA monitor some simplifications were made. The VGA standard defines a resolution which is not possible to display with the available hardware, thus the resolution was reduced and is defined in the display array section. The sync signals were both created using a single timer interrupt and this is explained further in the VGA timing signals section. To aid future projects the assembler routine used to output the display array is clearly explained. Finally the setup of the PWM signal is discussed.

4.1.1 VGA Timing Signals

The total time for the horizontal sync signal is 31.77 µs, of which the signal is low for 3.77 µs. This leaves 28 µs which is used to output the pixels and to create a border for the screen where nothing is drawn. The total time for the vertical sync signal is 16.68 ms and it can be defined in terms of horizontal lines, see Appendix A for more detailed information of VGA timing signals. This property is used to help reduce the processing demands placed on the microcontroller by using the horizontal sync signal to create the vertical sync signal.

The main c-file initialises all the variables used and sets up all the registers so that the various timers and ADCs all work as desired. It then enters an infinite while loop which is interrupted every 31.77 µs by an assembler interrupt routine. The horizontal sync is created using a combination of software commands and the timer. Outputting the pixels requires a defined amount of time and this forms part of the horizontal timing. The vertical sync is created by tracking the number of horizontal lines and using this number to switch the vertical sync from high to low. The flow diagram in Figure 8 shows the normal operation of the code.

The horizontal and vertical syncs require precise timing and changing the code in anyway could affect these timings and cause the screen not to sync onto the signals. Particular points to be careful of are: increasing the screen resolution by adding in more pixels or visible lines and adding too many extra commands to the game physics code. Not only must the display array size be increased but the appropriate amount of pixels should be added to the draw_pix routine in the assembler interrupt.

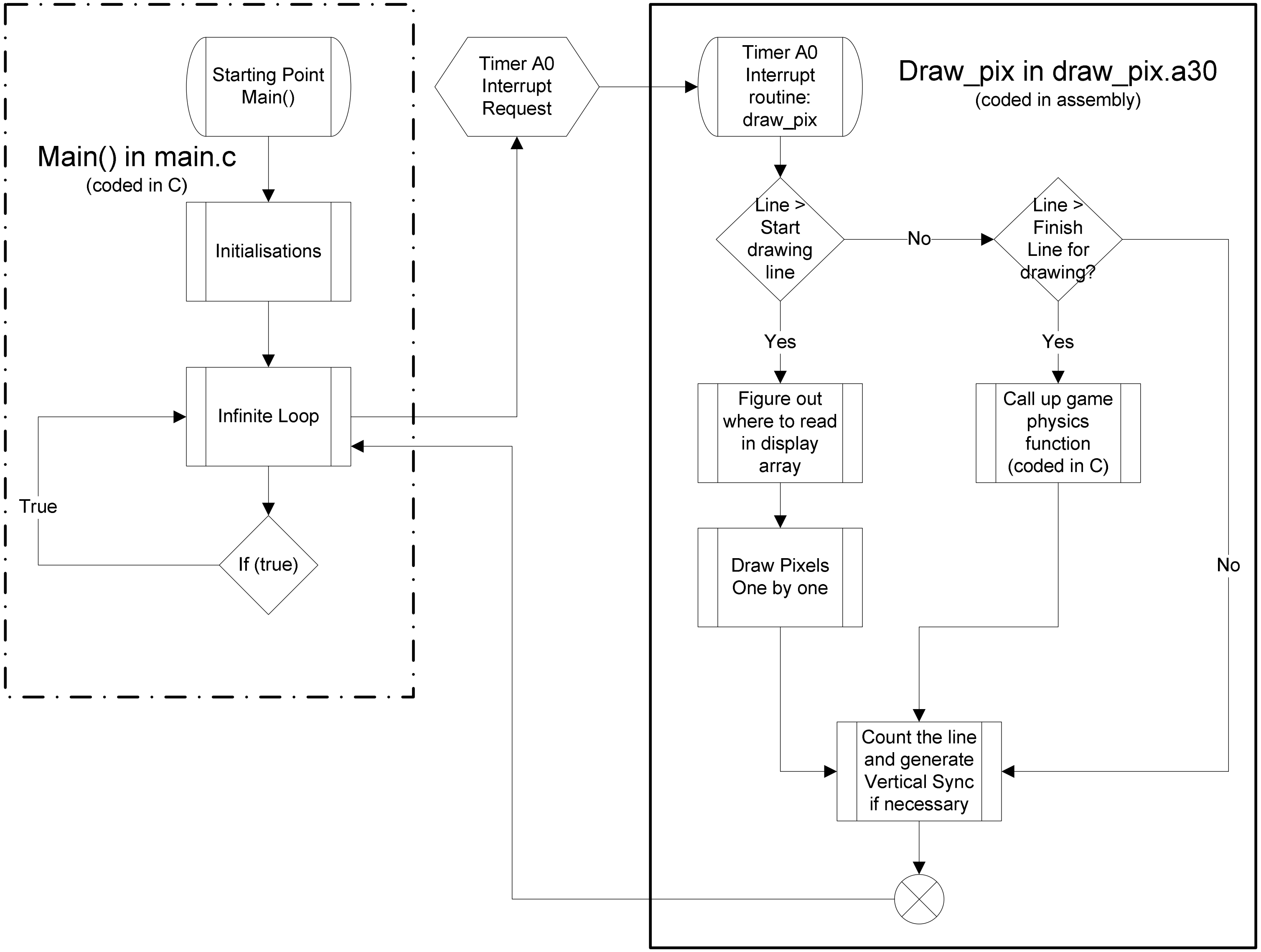

Starting Point Main()
Main() in main.c
(coded in C)
Initialisations
Infinite Loop
True
If (true)
Timer A0 Interrupt Request
Timer A0 Interrupt routine: draw_pix
Draw_pix in draw_pix.a30
(coded in assembly)
Line > Start drawing line
No
Line > Finish Line for drawing?
Yes
Yes
Figure out where to read in display array
Call up game physics function (coded in C)
Draw Pixels One by one
No
Count the line and generate Vertical Sync if necessary

Figure 8: Flow chart showing main flow of code

4.1.2 Display Array

A VGA screen is defined by its resolution and the industry standard is 640x480. This means that there are 480 lines each consisting of 640 pixels. Describing each pixel in a two dimensional array would represent a large quantity in RAM (300 kBytes) which exceed the capacity of our M16C. In order to reduce the space occupied in RAM a smaller two dimensional array is defined. Each line of this array will represent a fixed number of lines on the screen. In a similar fashion, a pixel in the display array represents several pixels on the screen. The grouping of the lines and pixels is illustrated in Figure 9. The colour for each pixel is encoded as a character in the software; this means that it is represented by 8 bits which are simply turned on or off depending on the desired colour.

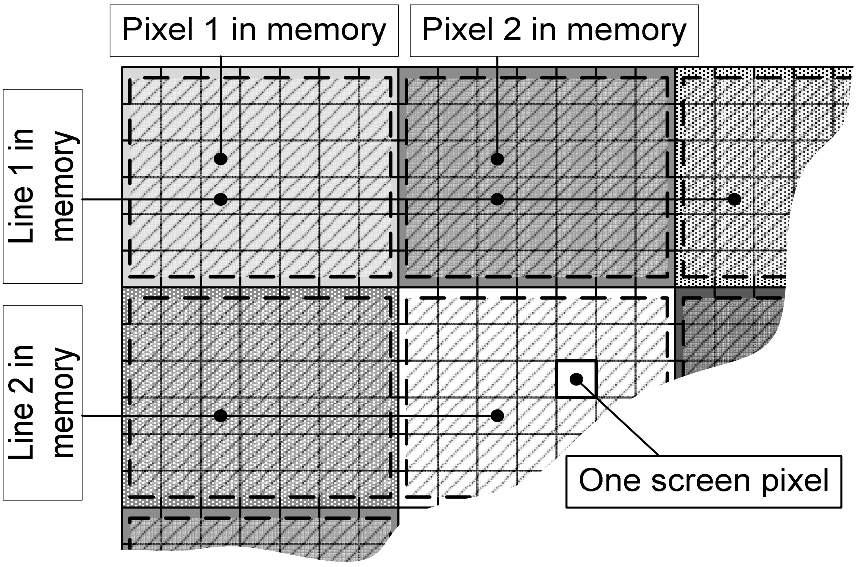


Figure 9: Representation of how lines and pixels are grouped together in the display array.

4.1.3 Drawing the pixel

The software developed to output an image on a VGA monitor consists of an interrupt assembler routine (draw_pix.a30) which is used to create both the horizontal and vertical syncs as well as to output the pixels to the screen at the correct time. This assembler routine calls a C sub-routine (physics) which is used to control all aspects of game play, user input and sounds.

According to the industry standard VGA timing, to be able to control a single pixel it is necessary to output pixel values at approximately 25 MHz. The M16C development board can only achieve a CPU clock of 24 MHz. Hence one “software pixel” as encoded in the display array will be displayed on several pixels across. The other constraint is that the interrupt must be exited before the timer A0 elapses again. This is the main limiting factor for the number of pixels that can be displayed across the screen.

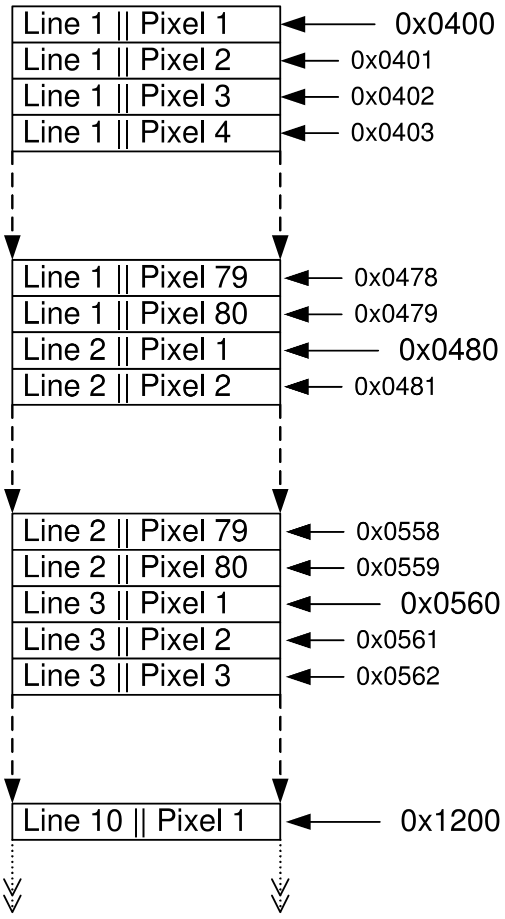
From a programming point of view, what is needed to output a pixel is only to put its value in the VIDEO port register (port 0). In assembly it will be performed by a MOV:

```
        MOV.B #pixel_color, VIDEO
```

To find the right pixel colour to output we will have to look for it at the right address in the display array in memory. This address will be stored in the register A1 so the instruction will be this one:

```
        MOV.B [A1], VIDEO ;Content of address A1 --> VIDEO
```

Hence we need to figure out in which address our pixel value is located.



First of all we have the array representing our screen in memory, this two dimensional table is called display.

We also have a global variable linemod, which is used to know which line of the display array has to be outputted during the current screen line.

A line of the display array is as long as the number of pixels we have to display across. The maximum (empiric) value we found was around 80 pixels.

The procedure followed to get to the right address is clarified here through use of an example. Let's say that the starting address of our two dimensional array display is 0x0400 (in hexadecimal).

In order to display the correct pixel we need to read the value contained in memory at the correct address. Each memory "slot" contains one byte and each pixel value is coded using just one byte. To go from one pixel to the next we just need to add 1 to the address value.

For example if we have the register A1 = 0x400, then the following instruction will display the first pixel of the first line and then the second pixel (of this same line).

```
        MOV.B [A1], VIDEO       ; Display Pixel one of the first line (located at 0x0400)
        INC.W A1                ; Value in A1 = 0x0400 + 0x0001 = 0x0401
        MOV.B [A1], VIDEO       ; Display Pixel two of the first line (located at 0x0401)
```

We also need to be positioned on the right address each time we start drawing a new line. The formula is quite simple: we need to offset the starting address by the line we want to display multiplied by the number of pixel in each line.

```
     1) MOV.W _linemod, R0        ;Put the screen matrix map line we are in into R0
     2) MUL.W #RES_PIXEL, R0      ;Multiply the number of line by the number of pixel in each line
     3) MOV.W #_display, A1       ;Put the start address of the screen matrix into A1
     4) ADD.W R0, A1              ;Offset the address with (linemod * RES_PIXEL) to get to the start
                                  ;of a the current line in the screen matrix
```

For example if we want to display line 3 of the display matrix, then we have linemod = 2 (because linemod = 0 correspond to the first line of our screen). In step 1 we have R0 = 2, after step 2, R0 = 2 * 80 = 160. Then in step three we set A1 = 0x400. Finally in step 4 we perform the operation A1 = A1 + R0 = 0x0400 + 0x00A0 = **0x4A0**.

4.1.4 Sound Generation

The buzzer is excited by a square wave and by varying the frequency and pulse width of the wave various notes and pitches can be achieved. The required square wave can be created using the PWM mode of the timer found on the M16C development board. In order to achieve a range of notes and differing pitch the 8 bit mode was selected. This allows both the cycle time and pulse width of the output wave to be altered.

The Timer A2 register is assigned the hex value 0x27. This sets up the timer in 8 bit PWM mode with the resulting pulse being output on the designated pin. Assigning *ta2* (timer countdown variable) different values alters the cycle time and pulse width of the output square wave. The timer variable is a 16 bit long address in memory and consists of a high and a low order address. The value of the high order address is denoted by *n* and it controls the pulse width of the signal i.e. how long the square wave is high for. This controls the loudness of the tone. The value of the low order address is denoted by *m* and this controls the cycle time of the square wave. The values for both the cycle time and pulse width can be calculated from the following equations:

$$Cycle\ time = \frac{(2^8 - 1)(m + 1)}{f_i}$$

$$Pulse\ width = \frac{n \times (m + 1)}{f_i}$$

where fi denotes the clock speed

Using the method mentioned above a table of notes was defined and a winning tune was implemented in the game code using these notes.

4.2 Game Code

The game code controls many aspects of the game play including: the movement and interactions of the ball and paddles, their position on the screen, the scoring system as well a small state machine used at the start to allow the user to select the type of game and input they would like to use.

Once the system is turned on the user encounters the first state which is a welcome menu, where the choice of game can be made: player versus player or player versus computer. The next state allows the player to select the type of input they would like to use: potentiometer or two way switch. The final state is game in play and the program remains in this state until the user resets the program.

4.2.1 Game Physics

The game starts with the ball moving off from the middle in any one of four directions defined as follows:

- Direction1 – ball moves up and left
- Direction2 – ball moves down and left
- Direction3 – ball moves up and right
- Direction4 – ball moves down and right

The initial direction of movement is random and is created using a random number generator. The ball's movement is checked every frame and its deflection off a player's paddle is determined by where the ball makes contact with the paddle and taking into account its original direction of travel. As described in the introduction if the ball makes contact with the outside edges of the paddles its speed and angle of reflection change. The angle and speed at which the ball moves off at is determined through a range of conditional statements. If the ball makes contact with either the top or bottom of border its moves off in the opposite vertical direction but with the same horizontal speed.

The ball's position in the field of play is constantly monitored and if it travels past either paddle then the player to have touched the ball last is awarded a point. The score for the current game is permanently displayed at the top of the playing field throughout the game. This is achieved through use of a character/figure lookup table which stores the numbers shapes as a position in the display array. Each time the score is changed the values in the display array are changed to mirror the number desired. Each time the ball makes contact with one of the paddles a tone is played by the buzzer. This is achieved by setting the value of the ta2 to a desired frequency value and allowing it to hold this value for a predetermined time. Each time the ball makes contact with a paddle the game speed increases until a maximum speed is reached. If a player scores a point the game speed is reset and the ball once again moves off from the centre of the playing field in one of four random directions. Once a player has scored 9 points the game is over and a victory tune is played.

5 User Instructions

A component layout diagram indicating the name and position of all components can be found in Appendix A. Check all wiring coming out from the box is sound and free from any damage. Ensure power supply is 5 V and connector is right type. Plug in power and VGA connectors and switch on power. The blue power LED will now light up and the VGA monitor will display the first page of the user menu which can be navigated using the player 1 control and is used to select the game type.

It is possible to control the volume of the tones coming from the box using the volume control slide on the front of the box. If at any point the user makes a mistake or wishes to abandon a game or simply restart the options menu, a single press on the reset button will take the user back to the first menu screen shown below in Figure 10.

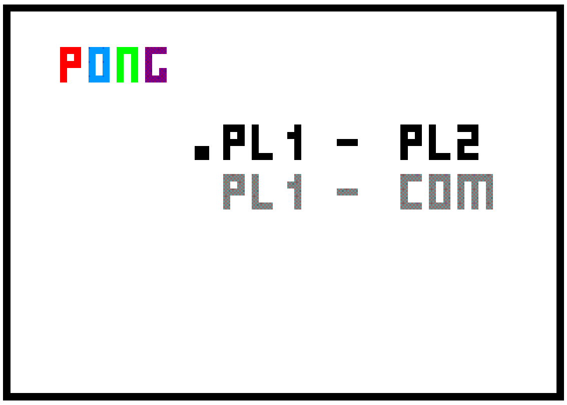


Figure 10: User menu showing game type options

The top of the controllers are easily identifiable by the cables running between the main console box and the control. Any reference made to an upwards direction implies the top of the control. By pressing the switch downwards the user is able to navigate through the choices offered by the menu. To confirm a particular choice the user presses the switch on the player 1 control in an upwards direction. These functions are clearly marked on player1 control. Once the type of game has been selected the second stage of the menu appears on screen and is shown below in Figure 11. It is possible to choose between using either the switches to control the paddle or to use the sliding potentiometer.

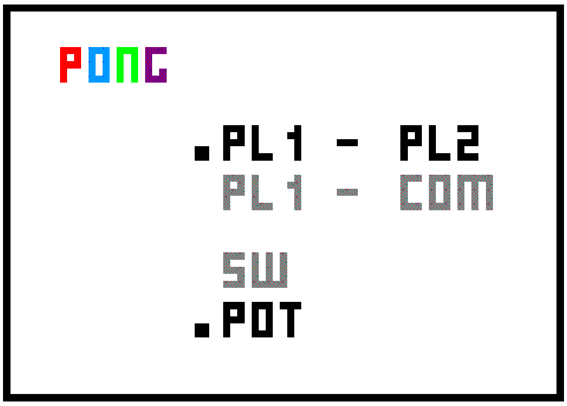


Figure 11: User input selection screen

5.1 Player1 versus Player2

To start a match between two human opponents, highlight the “PL1 - PL2” option and enter selection by pressing switch upwards. The user is then presented with a choice of controller type. The two options are: Switches (SW) or Potentiometers (POT). Highlight selection and confirm using player1 switch. The game will start shortly afterwards and continues until one player has scored 9 points. The end of the game is marked by the winning tune – which is the main theme from the Simpsons. To start a new game or to abandon the current game, simply press the reset button found on the front of the console box next to the power LED.

5.2 Player versus Computer

An AI opponent has been coded into the game and provides a challenge for a single player to beat. To start a match against the computer simply highlight and select the “PL1 – COM” option and then choose control method. Game begins with the ball moving off in a random direction from the centre of the playing field. The speed of the ball increases with each hit and reaches a maximum speed. The first to 9 wins, after which a win tune is played. To start a new game, simply press the reset button and the first menu will appear again.

6 Discussion

During the development of the project some problems were encountered and this section aims to inform the reader of how why they occurred so that they may be avoided if the project is extended. Due to the precise timing required by the VGA synchronisation signals, it is necessary to be aware that changes to the game code or interrupt routine could have adverse effects on the display. For this reason some useful information is provided here regarding altering the current source code.

6.1 Gotchas

One of the most important things to remember when altering the code is that the interrupt routine must never exceed 28 µs. This means that if more pixels are added it must be guaranteed that the interrupt routine remains constant. This is because it controls the horizontal sync timing which in turn controls the vertical sync timing. Changing the horizontal can also mean changing the vertical.

It is recommended to check after any modification that the synchronisation sync signals are still occurring at the correct timings. This can be done using a digital oscilloscope. Alternatively a visual check can be made by simply attaching a monitor and seeing whether or not the screen syncs on and the image is displayed correctly.

If the number of pixels can be increased without adversely affecting the display, then size of the display array must not be made too large. This is because the RAM on the microcontroller board is limited to 31 kB.

Some problems were encountered when using the ADCs. They would work at the high and low ends of the range but would jump past the middle values. After some trouble shooting it was found that the port direction (Port10) was set to output rather than input in one of the initialisation files provided by the development environment (hwsetup.c). This should be changed if the ADCs wish to be used.

7 Future Work

7.1 Improvements

The current game can be played as it is but there are a number of improvements which have been identified and can be made to enhance the display, tidy up the wiring inside the console box and make the system more portable.

When the game is being played some artefacts can be seen on the edges of the paddles and scores. They make the picture less sharp and detract slightly from the game play. The reason this occurs is because as the game cycles through the code it experiences different paths, some longer than others. This extra processing time affects the lines being output and causes the jagged edges to appear. One method to combat this would be to ensure all paths are of equal length i.e. all processes take the same number of processing power and improve the image quality.

The current height of the visible screen does not occupy the full 480 lines available, this is because they are grouped together. This grouping could be increased and along with it the visible height. This would be simple to implement but the appearance of the screen would need to be judged so that it did not appear stretched in the vertical axis.

At the moment only a single PCB was designed to hold the VGA connector and reduce the wiring complexity inside the box. Ideally a full PCB which would plug directly into the dual header strips on the microcontroller board, and which would remove the need for so many jumper wires would make the box less likely of malfunctioning through wire contact and make it tidier and easier to test.

The player controllers are hardwired inside the console this makes the console a bit clunky to transport. To improve on this a connector could be mounted on the side of the box and allow the user to plug the controllers in and out. This would also allow for different styles of controllers to be used. The slide potentiometer on the player1 controller is defective. The sliding resistive track does not always have good contact causing the player1 paddle to jitter because a clear value is not present for the ADC. This was only diagnosed once the final stage of the project was entered into and needs to be replaced.

7.2 Additional Features

The original Pong game has been accurately recreated in this project and some improvements to the original were thought of during the design phase but due to the time limit they were not implemented. Some of the extra ideas included a four player version, making different controllers, reducing the paddle size over the game time and creating an improved AI with varying levels of difficulty.

The four player Pong would have a player paddle on all four sides of the screen each controlled by a separate player. This would add a whole new dimension of game play. The different user input methods considered were infra red (IR) sensors or accelerometers instead of the sliding potentiometers. The original game play could be made more challenging by reducing the size of the player paddles over time. This would make it more difficult for the player to hit the ball. Improving the AI would allow a user to select a level of difficulty to play against. It would also allow a tournament to be implemented which itself would lead to the possibility of a high score table. The high score table could show best performances against the AI in the tournament. A pause button would be useful because it allows a player have a rest between/during points. If an improved AI and a high score table were implemented a more intricate menu system would be needed to access all the new options.

8 Works Cited

[1] Europe, R. (n.d.). *Renesas M16C/62P Starter Kit*. Retrieved from Renesas Technology: http://eu.renesas.com/fmwk.jsp?cnt=rsk_62p.jsp&fp=/products/tools/introductory_evaluation_tools/renesas_starter_kits/rsk_62p/

9 Appendix A: Circuit Diagram

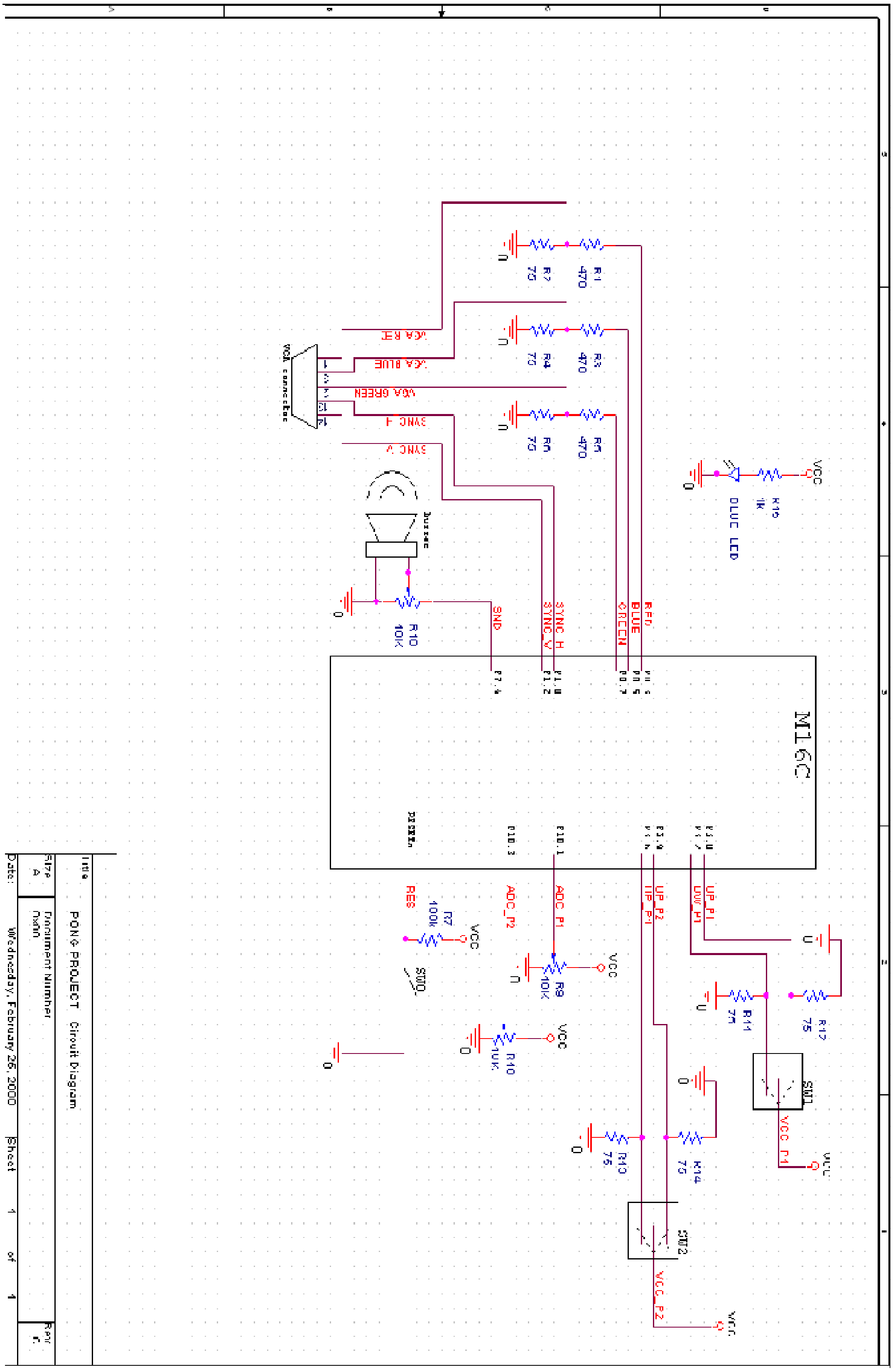


Figure 12: Circuit Diagram

10 Development board pin out diagram

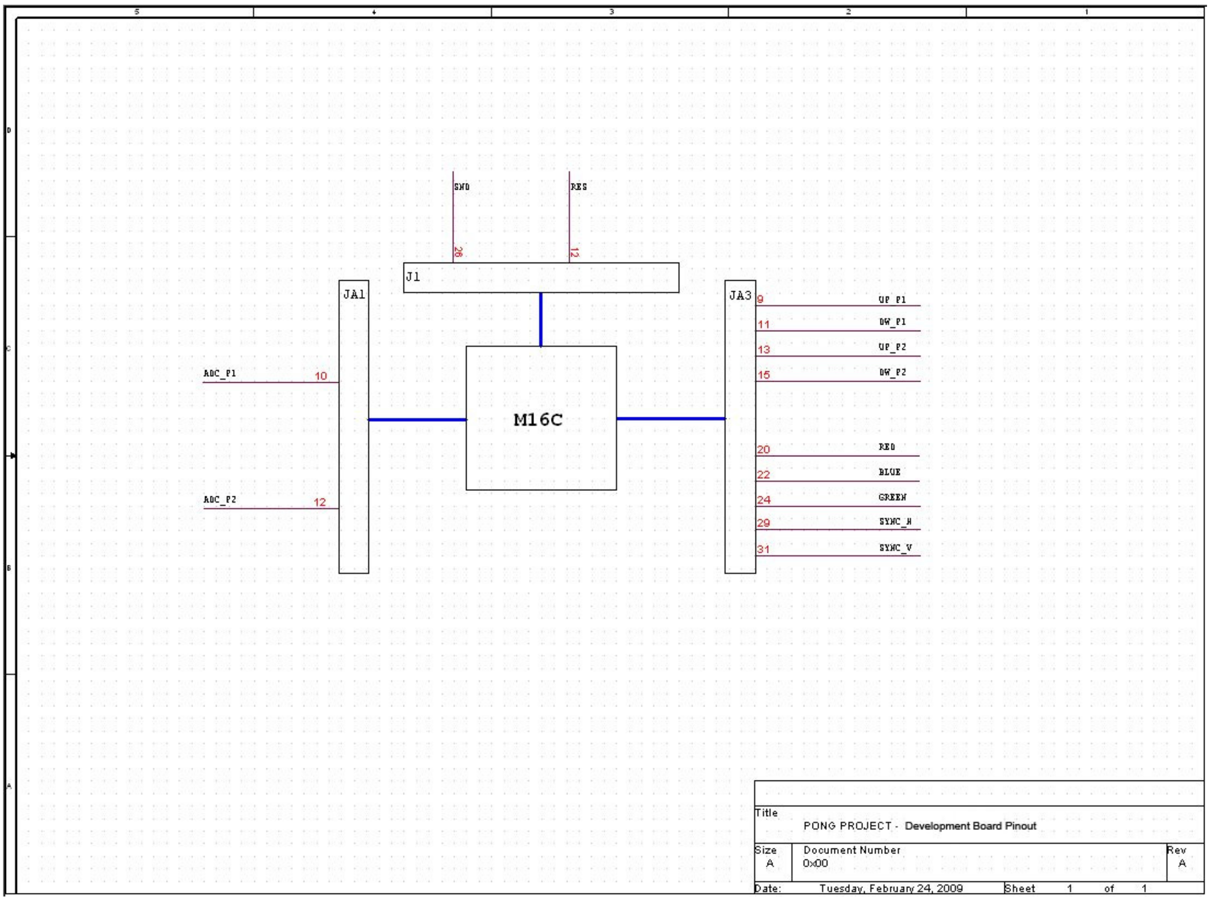


Figure 13: Diagram shows which pins on the development board connect to physical hardware

11 Appendix B: Component Layout

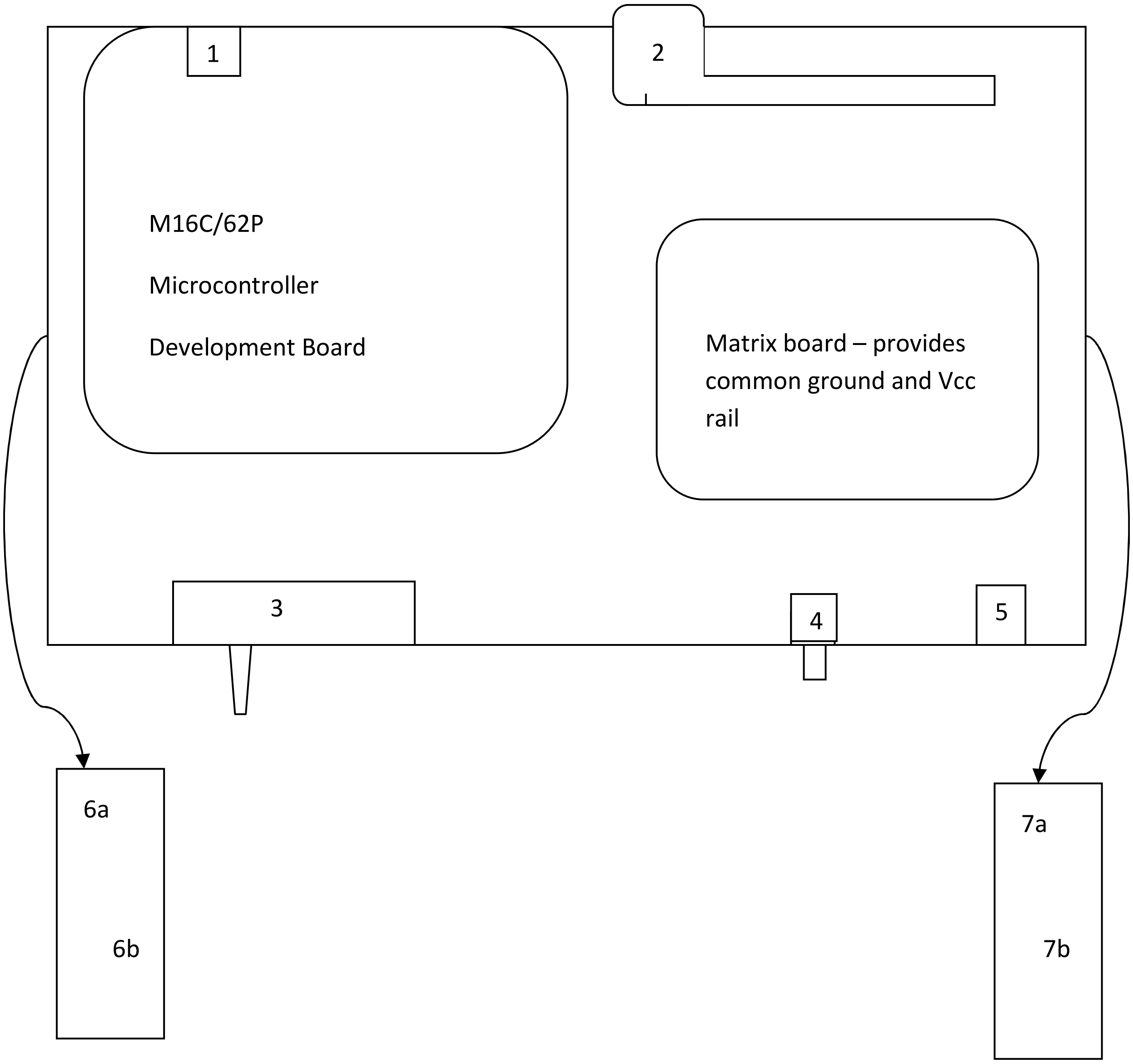


Figure 14: Component layout

1. 5V Power jack
2. VGA Connector PCB
3. Volume control slide potentiometer
4. Reset button
5. Blue power on LED
6. Player1 control. a – two way input switch; b – slide potentiometer
7. Player2 control. a – two way input switch; b – slide potentiometer

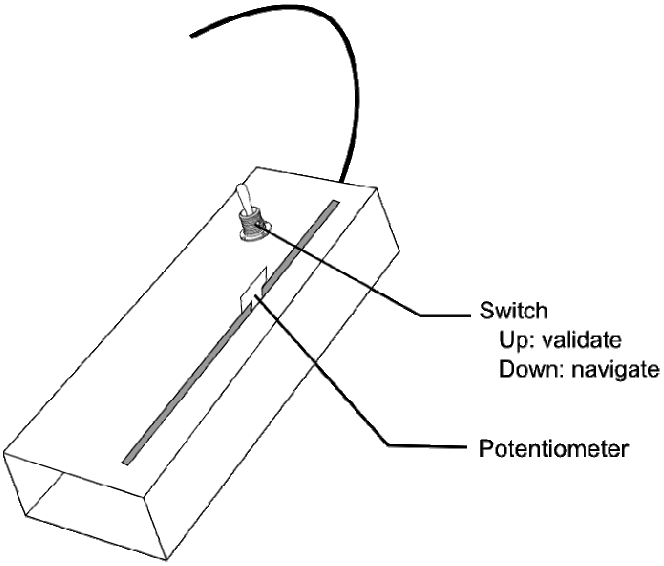


Figure 15: Close up of user control showing two way switch and slide potentiometer placement

12 Appendix C: Additional VGA Timing Information and PCB

Vertical synchronisation in terms of horizontal lines:

2 lines front porch
2 lines vertical sync
25 lines back porch
8 lines top border
480 lines video
8 lines bottom border

---

525 lines total per field

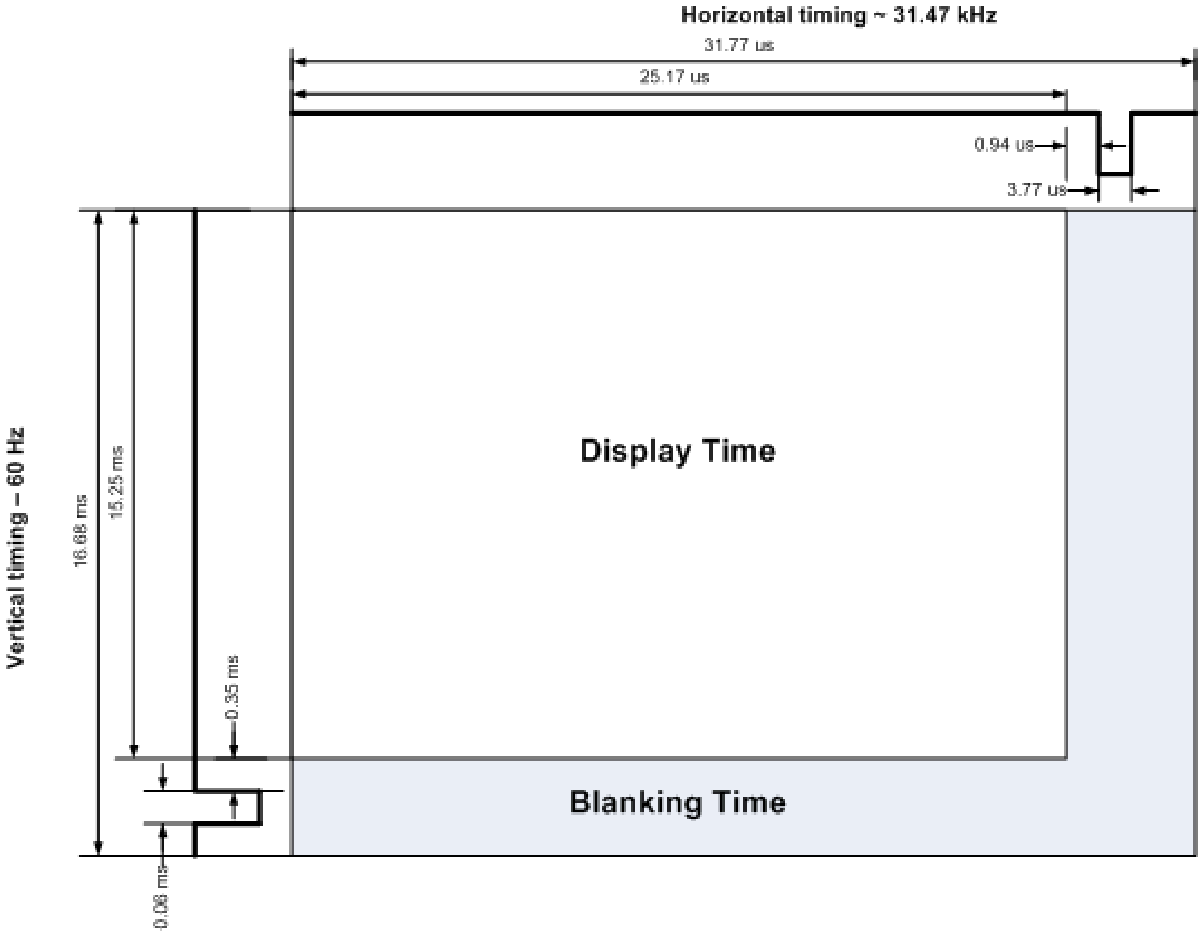


Figure 16: Diagram shows how the horizontal and vertical blanking periods function and shows the industry standard times used to define the various intervals

.

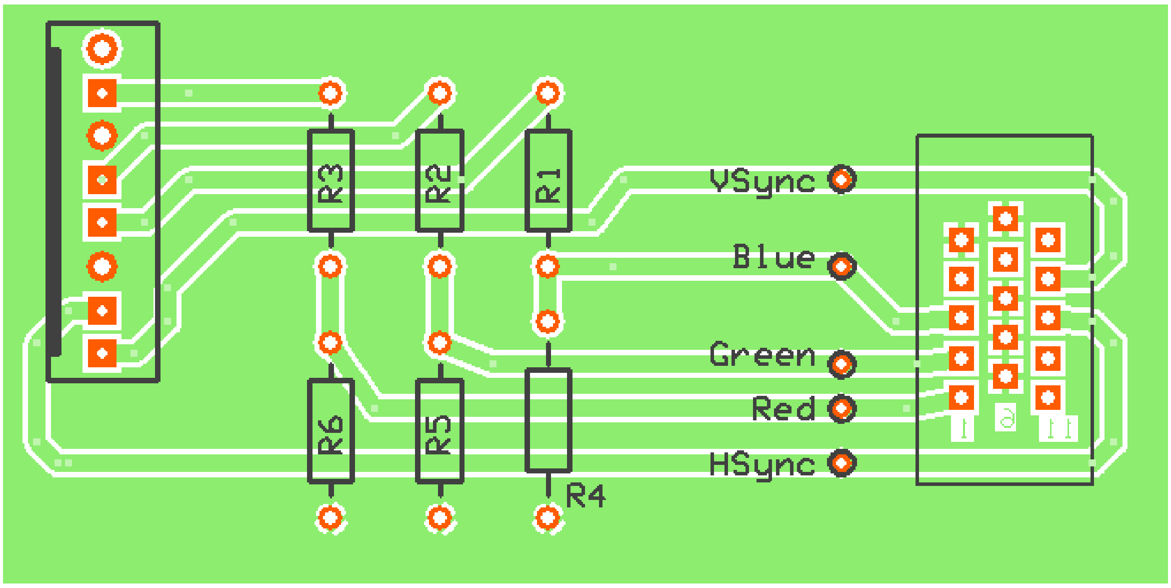


Figure 17: VGA connector PCB layout

R1, R2 and R3 = 470 Ω

R4, R5, and R6 = 75 Ω