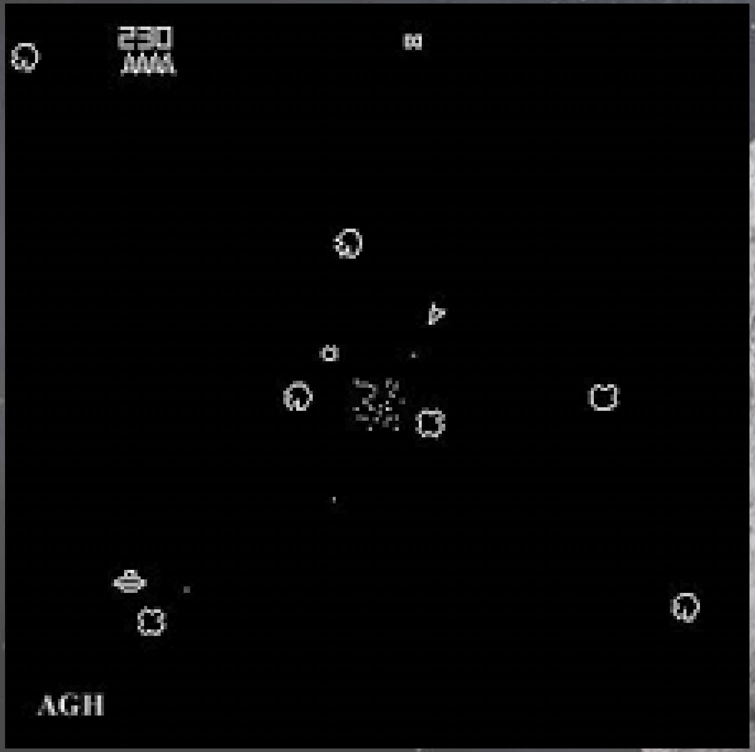
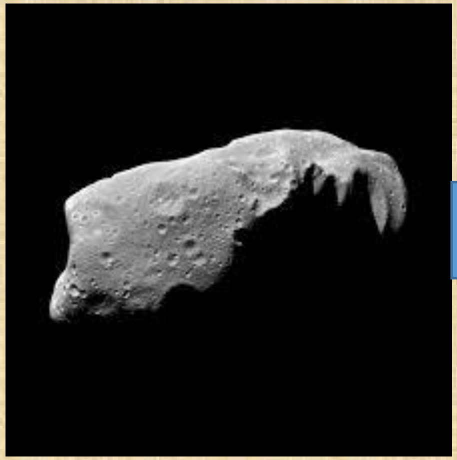
CREATION OF ASTEROIDS GAME USING VERILOG AND XILINX FPGA



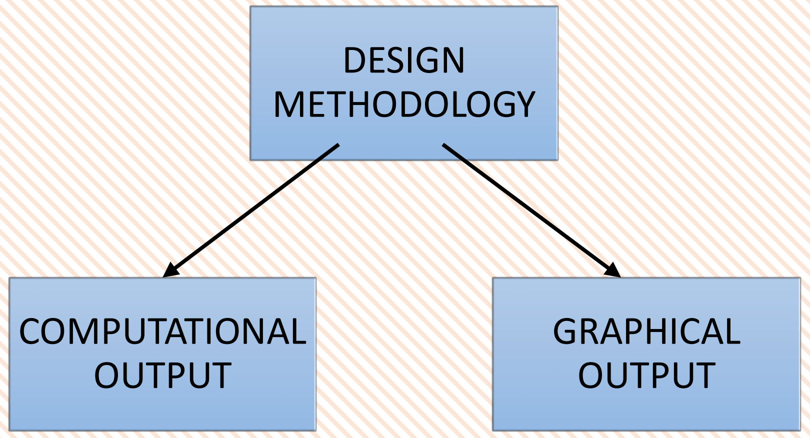
AIM: THE PROJECT IS TO CREATE THE LABKIT A VERSION OF THE POPULAR COMPUTER GAME 'ASTEROIDS'

THE MAIN THEME OF GAME "Asteroids is a game set in 2 dimensions featuring a spaceship in a field of moving asteroids. The asteroids move randomly and spin The player controls the spaceship, moving it around the playing field and shooting at the asteroids to destroy them. When hit by the spaceship's weapon the asteroids break down into smaller asteroids and eventually are removed from the field. The game is completed when all asteroids have been destroyed. The game is lost if the spaceship collides with an asteroid".



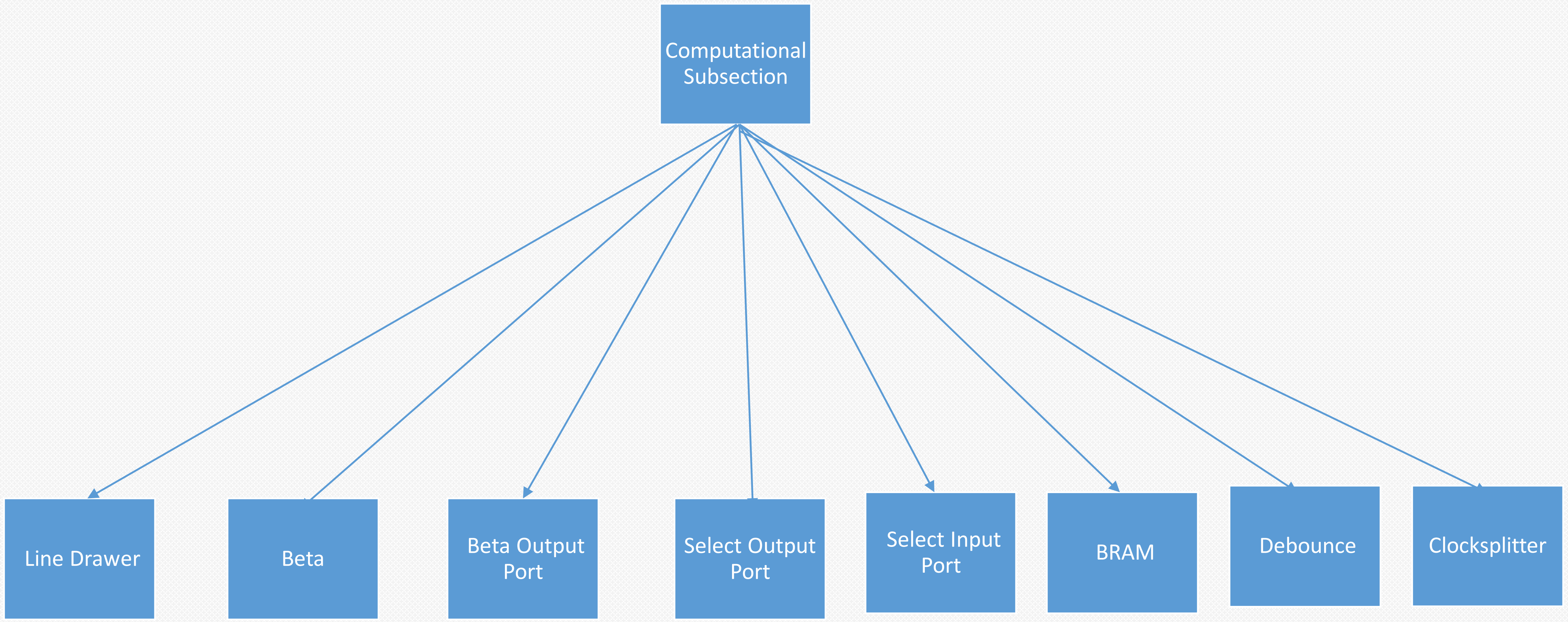
ASTEROID IMAGE

The original was created in 1978 by Atari and was the company's response to the popular and famous 'Space Invaders' game.



The computational section provides a **<u>stream of x and y co-ordinates where a pixel</u>** should be placed in the next frame. The pixels can appear randomly and anywhere on the screen at any time

The graphical section **<u>takes these pixels and draws them onto the screen</u>** appropriately


Computational Subsection
Line Drawer
Beta
Beta Output Port
Select Output Port
Select Input Port
BRAM
Debounce
Clocksplitter

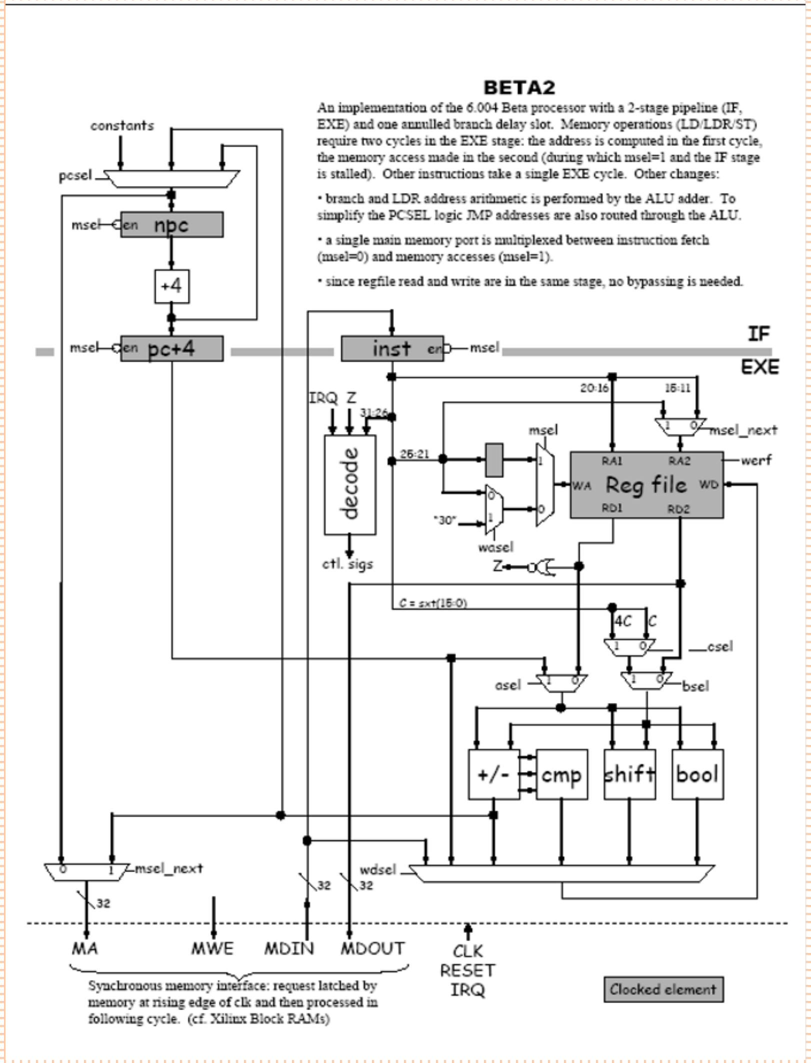
Line Drawer

In this module, the line drawing is performed by knowing: one co-ordinate, which is always the leftmost; a gradient function; the length of the line.

The final feature of the Line Drawer is to output the pixel (1023,1023) whilst not drawing so as to not inadvertently place a pixel on the screen.

Beta

A Beta is a type of microprocessor developed in the 6.004 class. It interprets instructions stored in code.

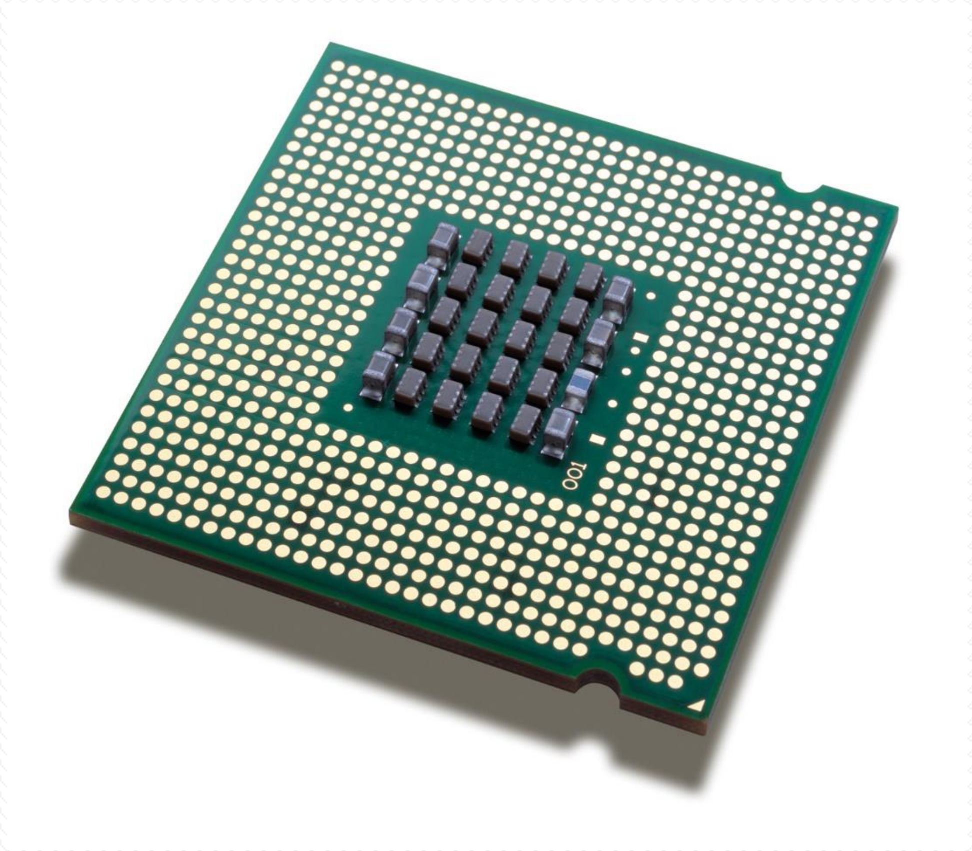

BETA2

An implementation of the 6.004 Beta processor with a 2-stage pipeline (IF, EXE) and one annulled branch delay slot. Memory operations (LD/LDR/ST) require two cycles in the EXE stage: the address is computed in the first cycle, the memory access made in the second (during which msel=1 and the IF stage is stalled). Other instructions take a single EXE cycle. Other changes:

- branch and LDR address arithmetic is performed by the ALU adder. To simplify the PCSEL logic JMP addresses are also routed through the ALU.
- a single main memory port is multiplexed between instruction fetch (msel=0) and memory accesses (msel=1).
- since regfile read and write are in the same stage, no bypassing is needed.

Synchronous memory interface: request latched by memory at rising edge of clk and then processed in following cycle. (cf. Xilinx Block RAMs)

A **microprocessor** is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.



Connection of the Beta to the LineDrawer

The LineDrawer Module and the Beta were then connected through the Output Port modules

Programming the Beta

Library Functions

The library functions created in §2.3.2.1 were testing sequentially being with the WritePort functions

Programs

The line drawing program was tested by programming it in to the Beta and monitoring the outputs of the x and y of the line drawing module on the logic analsyer

Beta Output Port

Beta output port is a set of registers for each of the 32 bits on each output port. They effectively, in combination with the **‘Select Output Port’** module create memory mapped ports for the beta such that it can write to hardware items.

Select Output Port

This module is actually simply some combinational logic substantiated between the Beta and Beta Output/Input Ports. Each Sel_Port line is taken high when the memory address of the output of the Beta is equal to the desired address of the port.

Select Input Port

This was essentially substantiated as a **MUX** and is in the main labkit code

BRAM

The BRAM is used by the Beta as both **program memory and data memory**.

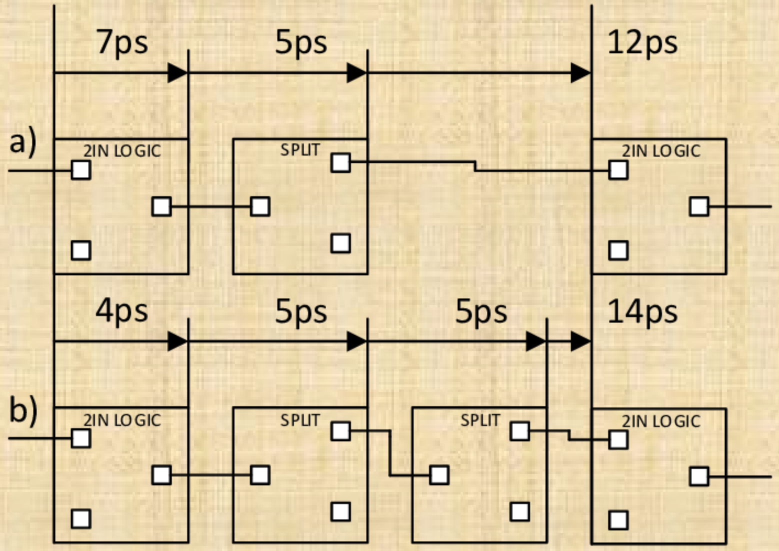
Block RAMs (or **BRAM**) stands for Block Random Access Memory. Block RAMs are used for storing large amounts of data inside of your FPGA.

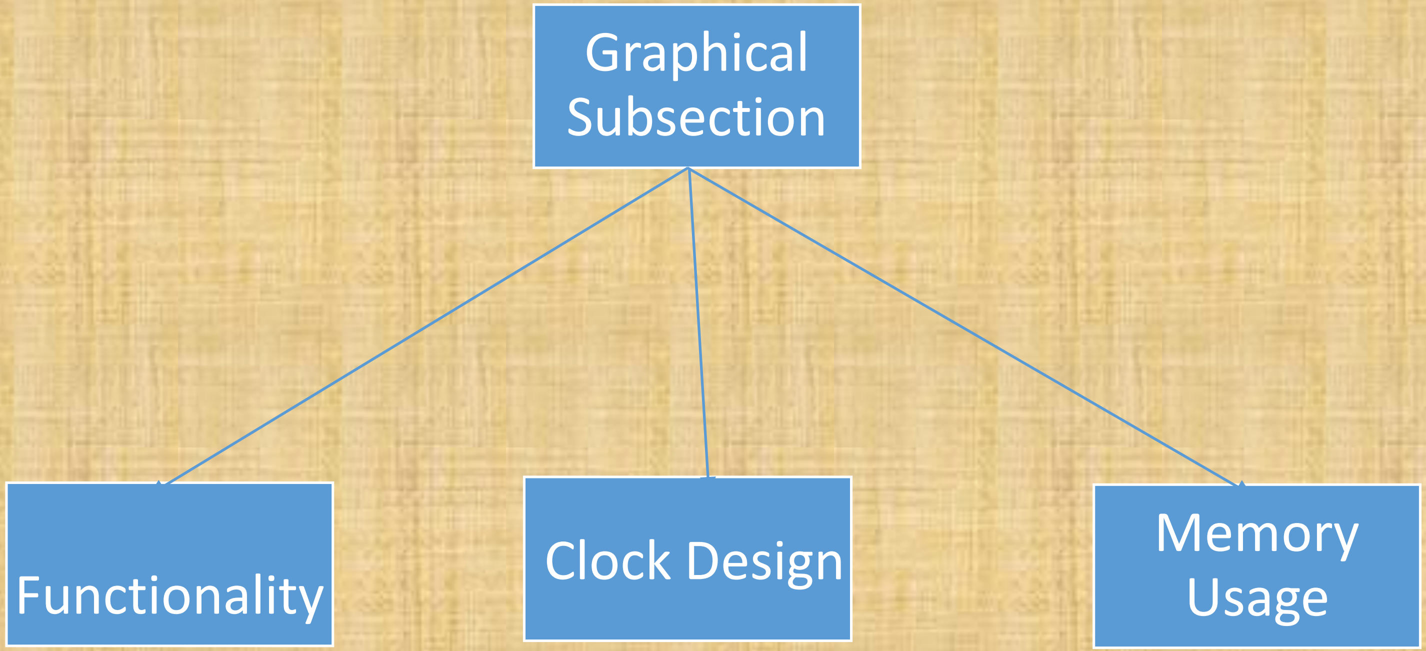
Debounce

This debouncer module is used for all push button inputs The module used also provides debouncing to cope with mechanical bounce of the switches

Clocksplitter

This module simply halves a given input clock frequency at its output.

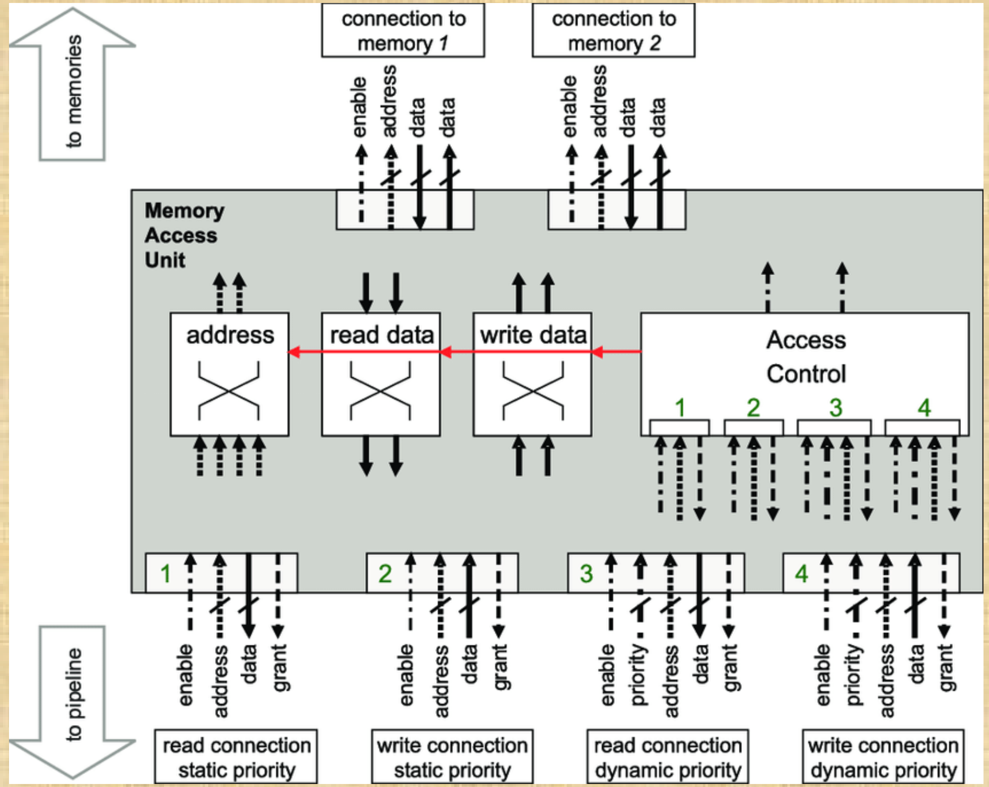


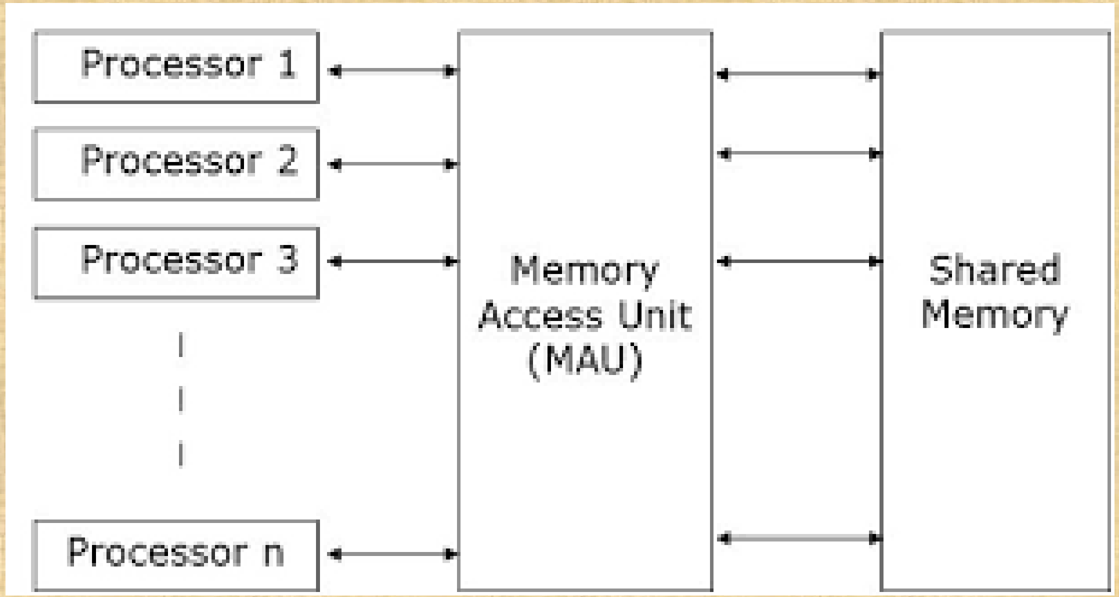

Graphical Subsection
Functionality
Clock Design
Memory Usage

Functionality

The Memory Access Unit (MAU) serves as an interface between the line drawer module and the XVGA module

MAU takes a set of x and y coordinates which corresponds to a new pixel that needs to be drawn on the screen, stores it in the memory





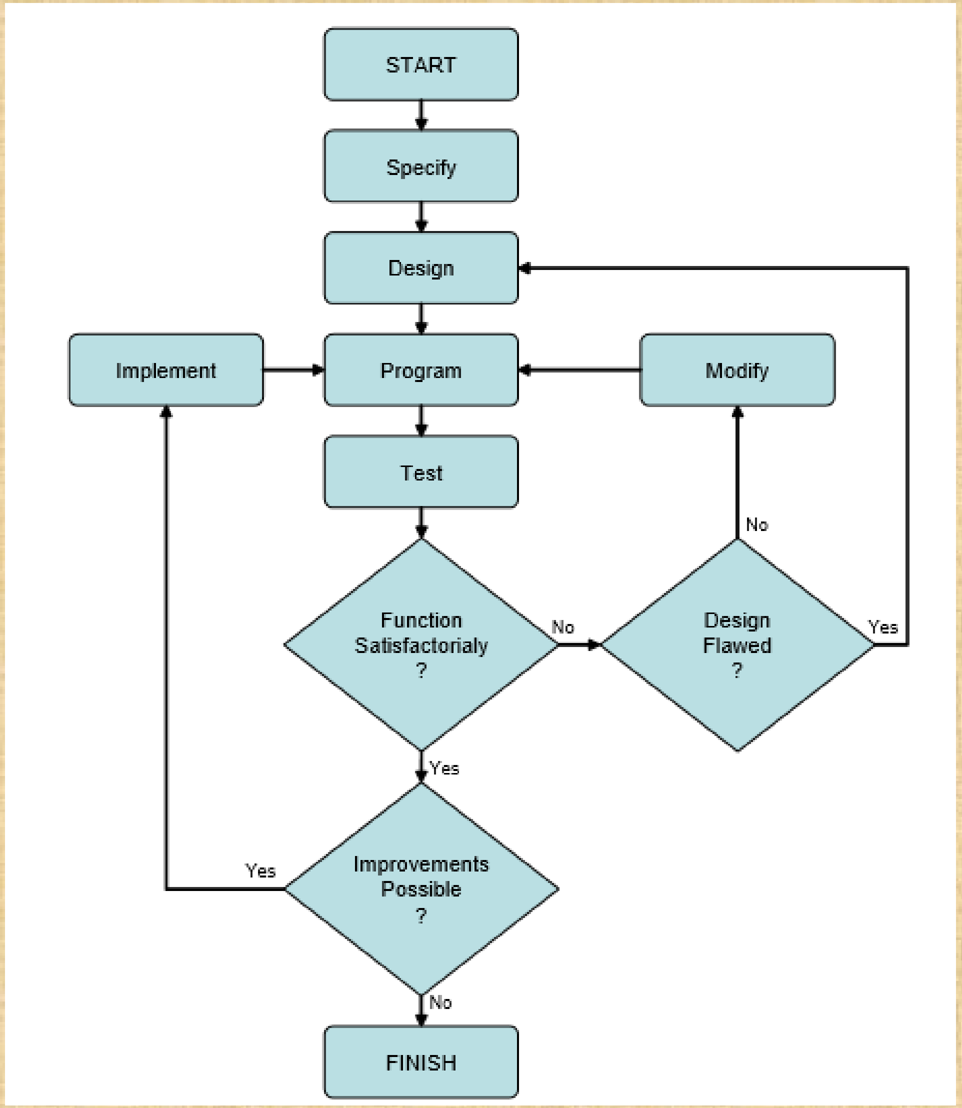
Clock Design

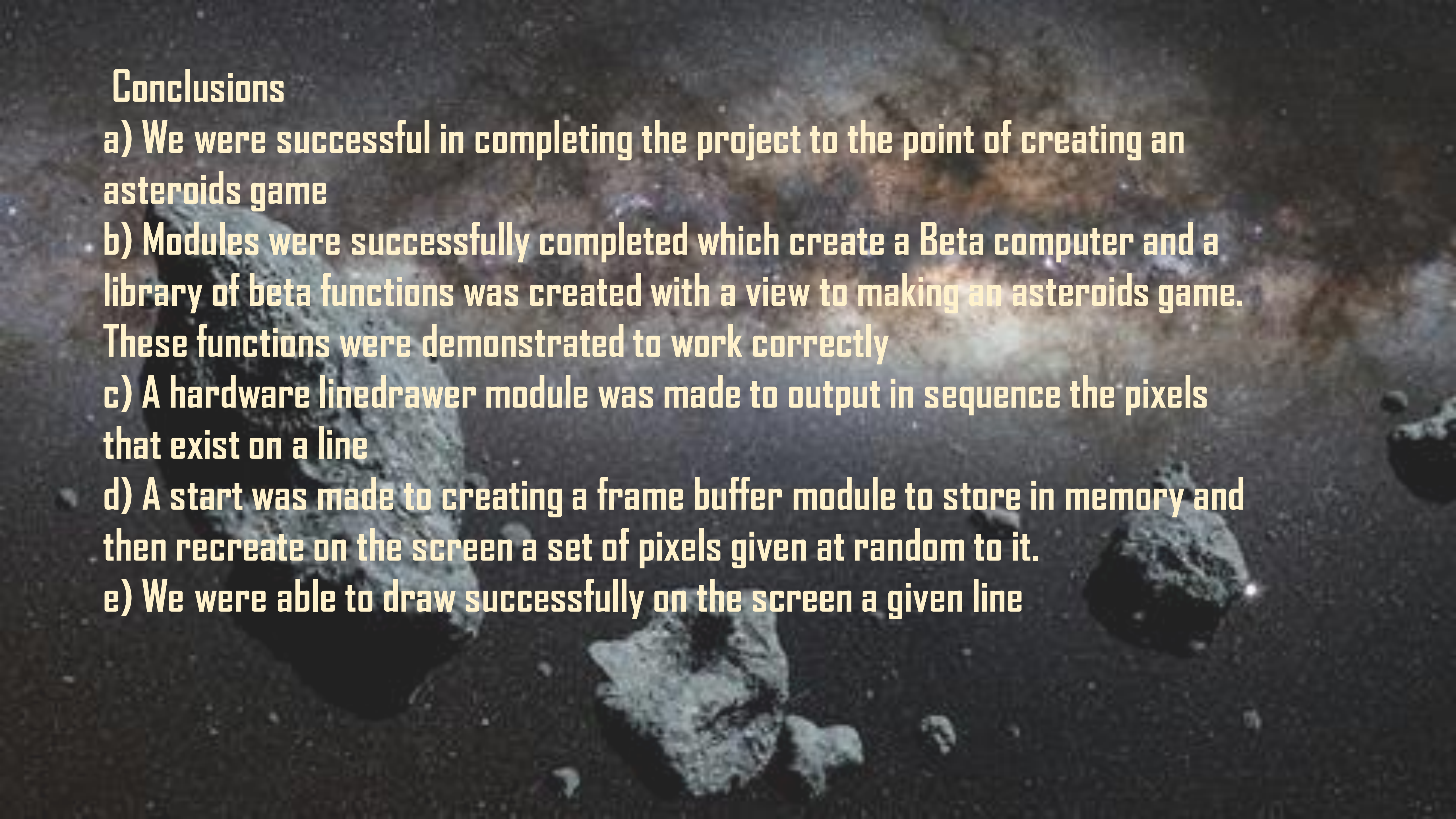
It is to order for the interface between the MAU and the XVGA module to work

Memory Usage

The screen resolution is 1024 * 768 pixels. Therefore, total memory needed is: 2 * 1024 * 768 bits = 1572864 bits = 192Kbytes This is more than 50% of the Block RAM (BRAM) available from the FPGA but less than 100%.

Testing and Debugging





Conclusions

a) We were successful in completing the project to the point of creating an asteroids game
b) Modules were successfully completed which create a Beta computer and a library of beta functions was created with a view to making an asteroids game. These functions were demonstrated to work correctly
c) A hardware linedrawer module was made to output in sequence the pixels that exist on a line
d) A start was made to creating a frame buffer module to store in memory and then recreate on the screen a set of pixels given at random to it.
e) We were able to draw successfully on the screen a given line

For Debounce

```
module debounce (reset, clock_65mhz, noisy, clean);
input reset, clock_65mhz, noisy;    output clean;

   reg [19:0] count;    reg new, clean;

   always @(posedge clock_65mhz)
    if (reset) begin new <= noisy; clean <= noisy; count <= 0;
 end
     else if (noisy != new) begin new <= noisy; count <= 0;
end
     else if (count == 650000) clean <= new;      else count <=
count+1;

endmodule
```

Betaport module

```
module beta_port(clk, reset,select,data_in,data_out);
input clk;
  input reset;
     input select;
    input [31:0] data_in;
    output [31:0] data_out;
  reg [31:0] data_out;

    always @ (posedge clk)
   begin
 if(reset)
    begin
       data_out <= 0;
   end
    else if(select)
   begin
 data_out <= data_in;
  end
  end endmodule
```