Final Design Report

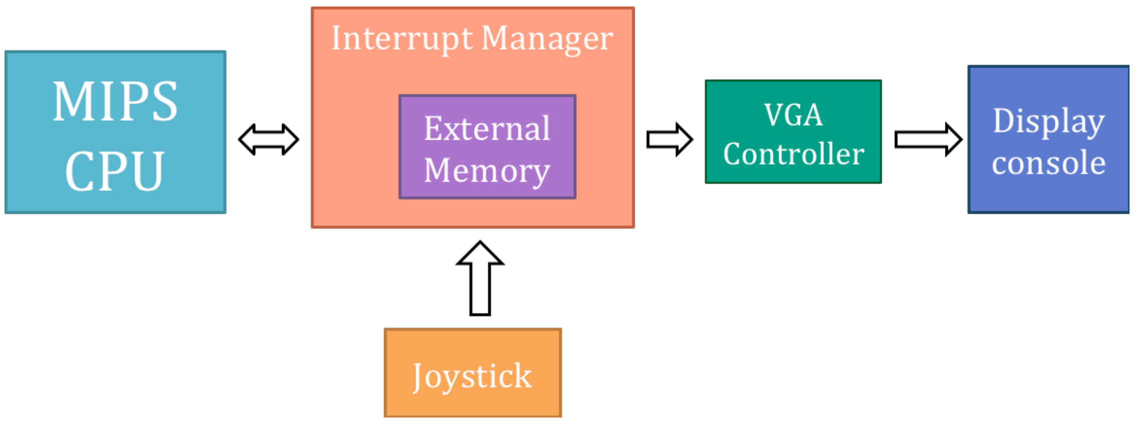
CSCE 230 Digital Design I

Dr Mohamed Shalan

Prepared by: Alia Hassan, 900131729 and Ingy Nazif, 900131781

May 21, 2015

INITIAL DIAGRAM AND DESIGN



Objective

We would like to emulate an arcade game machine, using our own standards, with the above designed structure. This is an initial design and will be updated/perfected after further discussion and research.

Material

- Nexys 3 FPGA board
- Xilinx ISE Suite CAD Software
- VGA connection cable
- CRT screen with 60-120Hz refresh rate
- PMOD JSTK extension (if time permits)

Project Outline

We intend to implement the above structure, using the various components in order to be able to achieve the final goal, which is to play a game on a console using a joystick. We will be using a ready made 32-bit MIPS CPU and building on that as the base of the structure.

CPU: The CPU shall be outsourced and is assumed to be a standard MIPS CPU 32-bit, it contains a built-in data memory, and reads/writes to/from this memory.

Interrupt manager: The interrupt manager is the interface between the joystick and the memory and is what translates the physical movements of the joystick into the data that is then read by CPU to edit the display data segment.

External memory: It will contain the screen array which will be used to communicate with the VGA controller and will be fetched from the data memory contained in the CPU.

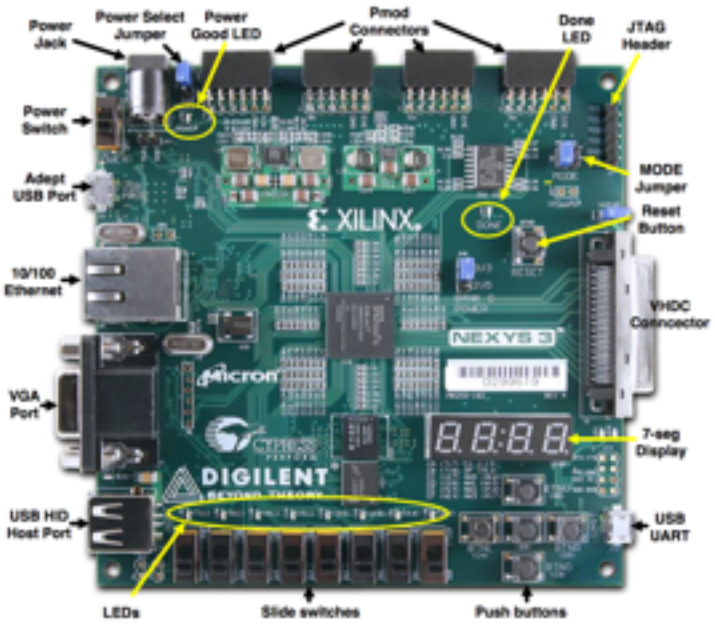
Joystick: The joystick shall interact with the system via the interrupt manager. This manipulation will generate what is called an interrupt and will prompt the CPU to execute a certain section of the code, which will update the screen array (80*60) containing the code of the tile associated to it. This will then be printed onto the console using the VGA control at intervals.

VGA Controller: The VGA controller has two main functions: it reads part of the screen (8*8 pixels chunks) and interprets to select the right tile and sends the data to the VGA port. The VGA controller will contain a small memory containing the tileset for the implemented game.

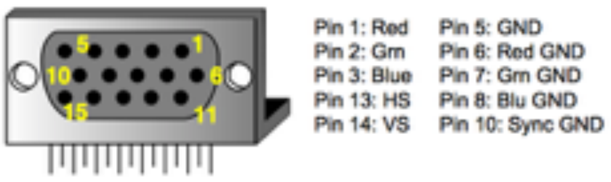
Display console: The console receives the images from the VGA controller, the image is written to the console line by line.

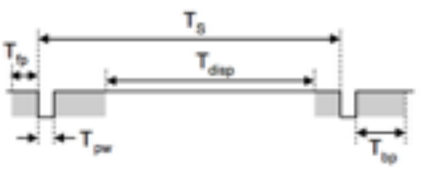
Research

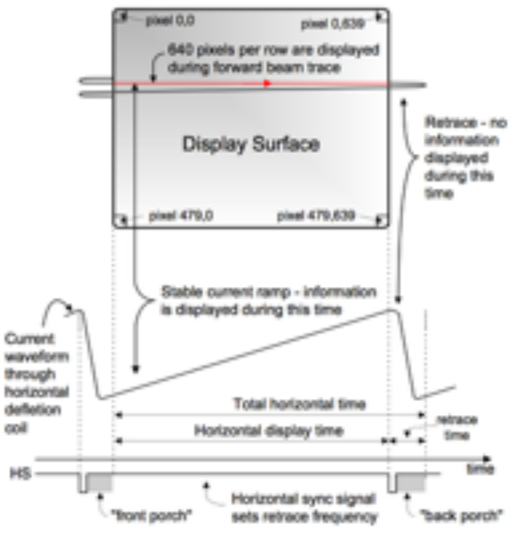
The Nexys3 Board: We chose this board because it supports the 8-bit VGA protocol, has PMOD connectors and has 576Kbits of fast block RAM which is important since the memory is a serious limitation to consider.

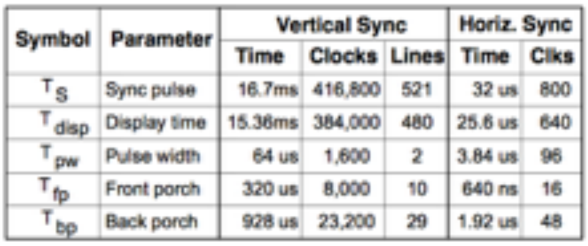


Video Graphics Array (VGA): it is a protocol used to display graphics with the RGB color coding on a CRT and can be used for most LCD displays as well. The VGA port available on the Nexys 3 board is an 8-bit standard VGA port, in which 3 bits are associated with Green, 3 other bits are associated with Red and 2 bits are associated with Blue. There are also two 1bit signals which are hsync (horizontal synchronization) and vsync (vertical synchronization).






pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
pixel 479,0
pixel 479,639
Retrace - no information displayed during this time
Stable current ramp - information is displayed during this time
Current waveform through horizontal deflection coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

These diagrams were taken from the Nexys3 board user manual and explain simply the idea behind the VGA protocol. The table sums up the requirements for a good VGA controller module on a display having a refresh rate between 60 and 120 Hz.

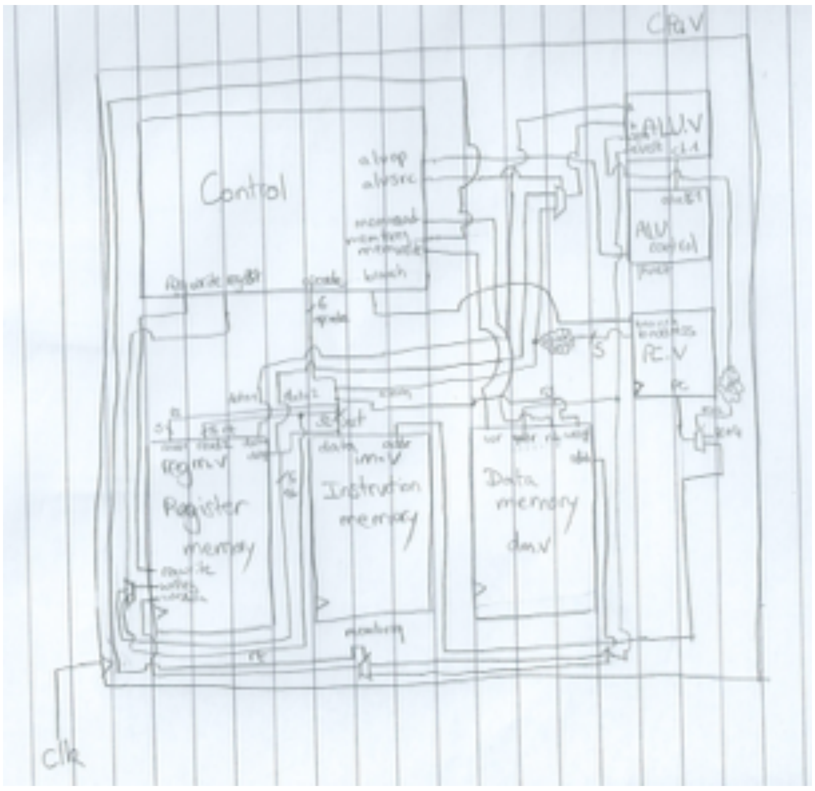
Graphics storage and display standards: two common techniques seemed the most interesting as to our application and needs: bitmapped graphics and character/glyph graphics. Bitmapped graphics basically represent a 2D array with each pixel defined in it, while Glyph graphics represent the screen by blocks of pixels (tiles).

	Pros	Cons
Bitmapped graphics	Flexible Precise	Takes up a huge amount of memory. If each pixel is represented in one byte the total memory just for the screen would be around 300kB
Character/Glyph graphics	Less flexible Repeated motifs/tiles	Takes a lot less space. If we have 12 different tiles of 8*8 pixels each pixel is a byte, then we need 768B compared to 300kB. But we need a separate way to describe the array.

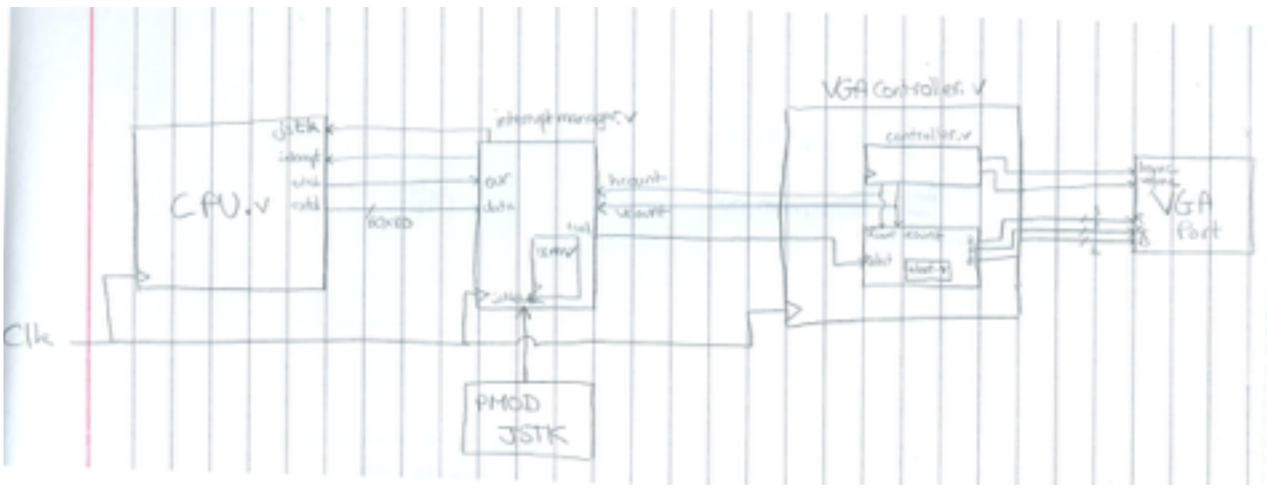
Hence we will be using the glyph graphics technique rather than the bitmapped graphics technique.

INITIAL BLOCK DIAGRAMS

Simplified CPU block diagram:

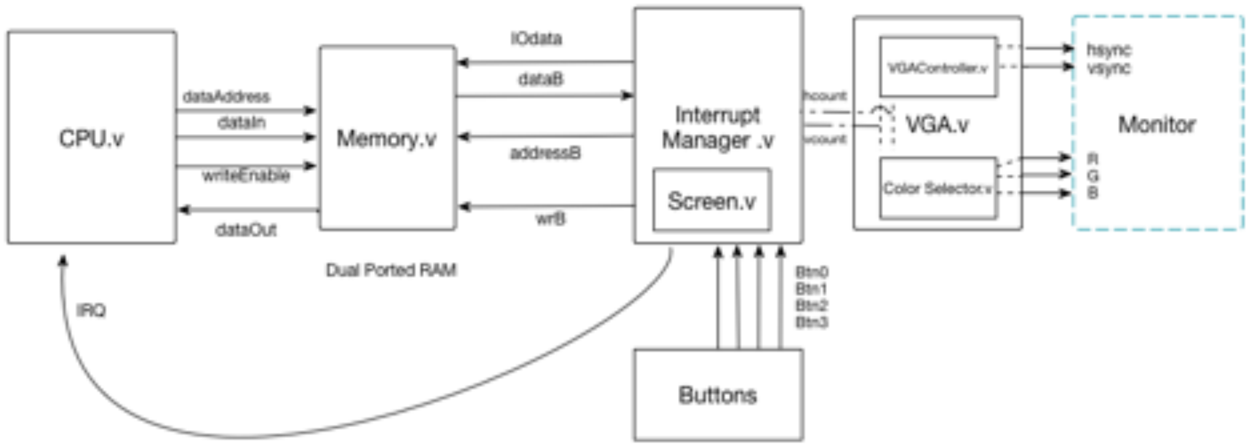


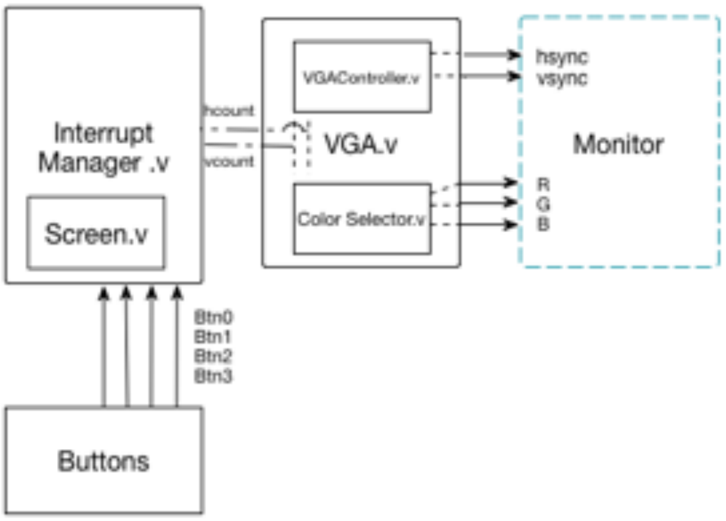
Simplified block diagram of the system (internal blocks are block RAM memory):



FINAL DIAGRAMS

Note: after implementing and testing the VGA controller with tiled memory source we worked in parallel on two different tracks, hard coding the game and integrating a CPU and it's code.





Above: integrating a 6502 CPU

Right: Hard Coded (into the interrupt manager)

After we figured out that a MIPS CPU would not have interrupts natively implemented we decided (for the integration with CPU) to choose a different CPU and we chose a 6502 processor.

After that we wrote the code for the CPU integration part, however, we were unable to assemble it and therefore we could not test it.

Moreover, we decided that the buttons PMOD module for convenience and practicality.

Extract of the Assembly code

(for the interrupt service)

```
 *=$1000 ;.ORG *=$1000
EQUATES:.ORG *+$5
POS_PXS: .SET  $01
POS_PYS: .SET $26
POS_TX: .SET $1C
POS_TY:.SET $01
IODATA: .SET $0700
SCREEN0: .SET $0200
SCREEN1: .SET $02C8
SCREEN2: .SET $0390
ROWS: .SET $1E
COLS: .SET $28
PL_TILE: .SET $00
WALL_TILE: .SET $03
EMPTY_TILE:  .SET $01
POS_PLX: .SET $0702
POS_PLY: .SET $0704
off: .SET $0050

 ;.ORG $FFFC
 ;.SET START
 ;.SET IRQ

 .ORG $0705
 IRQ: ;interrupt code
 SEI
 LDX POS_PLX
 ; getting address
 LDY POS_PLY
 CPY $10
 BCS bigger10
 LDY #$00
 STY $50
 ; calculate address
 STX $51
 JMP getInput

 bigger10: CPY $20
 BCS bigger20
 LDY #$01
 STY $50
 TXA
 CLC
 SBC #$10
 STA $51
 JMP getInput

 bigger20:
 LDY #$02
 STY $50
 TXA
 SBC #$20
 STA $51

 LDA POS_PLX ; if j is odd store 1 in 52
 AND $01
 STA $52
 ; if( A! =0 ) substract 0
 BEQ getInput
 SBC $01

 getInput: ; Y had the right index for offset
 ; j* M => 53
 ; j= 20= 16+4 shift 4 bits then store result then shift
2 bits and add
 LDA POS_PLX
 ASL
 ASL
 ASL
 ASL
 STA $53
 LDA POS_PLX
 ASL
 ASL
 ADC $53
 ADC $51
 ; A has i+j*M
 STA $54

 ; get input/output data
 LDX IODATA
 CPX #$04
 BNE LEFT ; if(right)
 CLC
 ;check if j was even or odd
 LDY $54
LDA $52
 CMP #$00 ; if(odd)
 BEQ even
 ; if(offset== 0)
         LDX $50
         CPX #$00
         BNE offsetnot0
         LDA SCREEN0, Y
         AND $f0
         LSR
         LSR
         LSR
         LSR;A has the tile
                  ; if( tile !=BLOCK)
                  CMP #WALL_TILE
                  BEQ endIRQ
                  ; move is valid
                  LDX SCREEN0, Y
                  TXA
                  AND #$0f
                  STX $59
                  LDX PL_TILE
                  TXA
                  LSR
                  LSR
                  LSR
                  LSR
                  ORA $59
                  STA SCREEN0, Y
                  DEY
                  LDX SCREEN0, Y
                  TXA
                  AND #$0f
                  STX $59
                  LDX EMPTY_TILE
                  TXA
                  LSR
                  LSR
                  LSR
                  LSR
                  ORA $59
                  STA SCREEN0, Y

                  LDX POS_PLX
                  INX
                  STX POS_PLX
                  JMP endIRQ
         offsetnot0:
         LDX $50
         CPX #$01
         LDA SCREEN1, Y
         AND $f0
         LSR
         LSR
         LSR
         LSR
         ;A has the tile
                  ; if( tile ==BLOCK)
                  CMP #WALL_TILE
                  BEQ endIRQ
                  ; move is valid
                  LDX SCREEN1, Y
                  TXA
                  AND #$0f
                  STX $59
                  LDX PL_TILE
                  TXA
                  LSR
                  LSR
                  LSR
                  LSR
                  ORA $59
                  STA SCREEN1, Y

                  DEY
                  LDX SCREEN1, Y
                  TXA
                  AND #$0f
                  STX $59
                  LDX EMPTY_TILE
                  TXA
                  LSR
                  LSR
                  LSR
                  LSR
                  ORA $59
                  STA SCREEN1, Y
                  LDX POS_PLX
                  INX
                  STX POS_PLX
                  JMP endIRQ
         LDX POS_PLX
         INX
         STX POS_PLX
         JMP endIRQ
offsetnot1:
LDA SCREEN2, Y
AND $f0
LSR
LSR
LSR
LSR
;A has the tile
         ; if( tile ==BLOCK)
         CMP #WALL_TILE
         BEQ endIRQ
         ; move is valid
         LDX SCREEN2, Y
         TXA
         AND #$0f
         STX $59
         LDX PL_TILE
         TXA
         LSR
         LSR
         LSR
         LSR
         ORA $59
         STA SCREEN2, Y

         DEY
         LDX SCREEN2, Y
         TXA
         AND #$0f
         STX $59
         LDX EMPTY_TILE
         TXA
         LSR
         LSR
         LSR
         LSR
         ORA $59
         STA SCREEN2, Y
         LDX POS_PLX
         INX
         STX POS_PLX
         JMP endIRQ



         DEY
         LDX SCREEN2, Y
         TXA
         AND #$f0
         STX $59
         LDX EMPTY_TILE
         TXA
         ORA $59
         STA SCREEN2, Y
         LDX POS_PLX
         INX
         STX POS_PLX
         JMP endIRQ
```

Sources

Nexys4 User Manual: http://www.digilentinc.com/Data/Products/NEXYS4/Nexys4_rm_V2.pdf

VGA Fromat and Timing: http://www.javiervalcarce.eu/html/vga-signal-format-timming-specs-en.html

Documentation about the 6502 processor and it’s assembly language :http://www.obelisk.demon.co.uk/6502/architecture.html

6502 CPU in Verilog :https://github.com/Arlet/verilog-6502