Project Goals

- Design a Sprites Graphics engine inspired from the TI TMS9918
  - Extend texture resolution from 8 pixels to **32 pixels**, sprite resolution >= **64 pixels**
  - Update colors from Light/Dark pixels to **9 bit colorspace** (512 color alternatives/pixel)
- Enable screen scrolling in all directions
- Runtime image programming interface for background patterns
  - No Graphics MIFs!
  - Allows for simplified creation of new game tracks and menus
  - Mitigates limited RAM space on the Cyclone V
- Update VGA resolution to XGA (1024 x 768 60 Hz)
- Implement real-time computation of sprite rotation
- Enable game sounds
- Model car physics and have realistic race dynamics
Sprite Graphics Implementation - Pattern Tables

- 9 Independent Dual Port RAMs to represent 512 colors/pixel
- Large “register address space”
- 8192 32 bit words / (32 bits/pattern)
  - 256 patterns

Pattern Tables:
- Red Color Pattern Table “Bit 2” 0x0000
- Red Color Pattern Table “Bit 1” 0x0000
- Blue Color Pattern Table “Bit 0” 0x0000
- Green Color Pattern Tables
  - Pattern0 Line 0 (U32)
  - Pattern0 Line 1 (U32)
  - Pattern0 Line 31 (U32)
  - Pattern1 Line 0 (U32)
  - Pattern1 Line 31 (U32)
Sprite Graphics Implementation - Pattern Lookup

Name Table

0xFFFF

0x0000

Name0 (U8)
Name1 (U8)
Name2 (U8)
Name3 (U8)
Name4 (U8)

Pattern Tables LUTs

2D Background Pattern Space

0 1 2 3 4 ...

64 x 1024 Patterns

hCount

vCount

0xFFFF
Sprite Graphics Implementation - Movement

- Coarse and fine grain movement
  - 32 pixels “nameOffsetX/Y”
  - 1 pixel “pixelOffsetX/Y”
- Updated synced to VSYNC of VGA
- Unsigned offsets were a non-ideal design choice
  - Made movement more complicated than necessary
- Reasonably smooth movement, still isolating a few bugs
Programmatic Map Generation

```
treeRowsTop = range(0, 1024, 16)
treeRowsBottom = range(1, 1025, 16)
houseRowsZero = range(0, 1024, 24)
houseRowsOne = range(1, 1025, 24)
houseRowsTwo = range(2, 1026, 24)
houseRowsThree = range(3, 1027, 24)

for row in range(0, 1024):
    for col in range(0, 64):
        if (col == 29):
            print "1"
        elif (col == 30 or col == 31):
            print "2"
        elif (col == 31):
            print "3"
        elif (col == 32):
            print "4"
        elif (col == 33):
            print "5"
        else:
            print "6"

for row in range(0, 1024):
    for col in range(0, 64):
        if (col == 29):
            print "1"
        elif (col == 30 or col == 31):
            print "2"
        elif (col == 31):
            print "3"
        elif (col == 32):
            print "4"
        elif (col == 33):
            print "5"
        else:
            print "6"
```

```
for i in range(0, 256):
    if i == 1:
        print "straightGrassLeft.png"
    elif (i == 2):
        print "roadTileWithoutLine.png"
    elif (i == 3):
        print "roadTileWithLine.png"
    elif (i == 4):
        print "straightGrassRight.png"
    elif (i == 5):
        print "tree-0-0.png"
    elif (i == 6):
        print "tree-0-1.png"
    elif (i == 7):
        print "tree-1-0.png"
    elif (i == 8):
        print "tree-1-1.png"
    elif (i == 9):
        print "TopRedHouse-0-0.png"
    elif (i == 10):
        print "TopRedHouse-1-0.png"
    elif (i == 11):
        print "TopRedHouse-2-0.png"
    elif (i == 12):
        print "TopRedHouse-3-0.png"
    elif (i == 13):
        print "TopRedHouse-0-1.png"
    elif (i == 14):
        print "TopRedHouse-1-1.png"
    elif (i == 15):
        print "TopRedHouse-2-1.png"
    elif (i == 16):
        print "TopRedHouse-3-1.png"
```

```
"trackXNames.txt"
```

```
grassYellow.png
straightGrassLeft.png
roadTileWithoutLine.png
roadTileWithLine.png
straightGrassRight.png
tree-0-0.png
tree-0-1.png
tree-1-0.png
tree-1-1.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
```

```
"trackXPatterns.txt"
```
Sprite Rotation - Rotation Matrix Approach

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

Wikipedia: Rotation Equation
Sprite Rotation - Development Approach

1) High Level Software Algorithm POC

```scala
val theta = -0.6
val rotationOriginX = -64 * math.cos(theta) - 64 * math.sin(theta)
val rotationOriginY = -64 * math.sin(theta) + 64 * math.cos(theta)
val translatedXOrigin = rotationOriginX + 64
val translatedYOrigin = -(rotationOriginY - 64)

println("TRANSLATED ORIGIN Y" ++ translatedYOrigin.toString)
println("TRANSLATED ORIGIN X" ++ translatedXOrigin.toString)

val vecXNormalized = 2 * math.cos(theta)
val vecYNormalized = 2 * math.sin(theta)
```

2) System Verilog implementation + Modelsim Validation

3) Signal-Tap II Debugging of hardware realized solution
Lessons Learned

- Teamwork in an academic setting is difficult
  - Different experience levels, time commitments, interest etc. etc.

- Quartus II software has many quirks
  - $X \leq Y$ can yield unexpected results, sometimes it’s better to manually index the bits you care about
  - Parameter constants can be different in the RTL viewer from what you would expect based on your System Verilog code
  - Warnings are almost too forgiving, some may be better to fail the compilation (net inference)

- Module based encapsulation is critical to help debug RTL code and allow for reasonable viewing of the system interconnections

- Signal Tap II is a crucial debugging tool, without it our project would have missed several desired deliverables.

- Open source drivers can be unpredictable to work with and be non-trivial to build for an embedded target