# ECE540 Project 2:

# Rojobot Line-Follower Using Picoblazes and VGA Controller

# Theory of Operation

## Members: Colten Nye and Hoa Quach

1. **Hierarchy:**

Nexys4fpga

Kcpsm6

nexus4\_bot\_if

VGA subsystem

Bot.v

Colorizer

Icon wrapper

Clock Gen

Icon ROM

dtg

Proj2.v

(psm)

7-segment display

1. **VGA controller**

The module vga\_subsystem acts as a top module for each of the vga components, including the 25 MHz vga clock generator, the vga timing generator, the icon generator, and the colorizer. The vga\_subsystem takes in data from the bot module and feeds it to the icon module, as well as gives data to the bot that it needs for generating the map feed. The vga module then receives this map feed and gives it to the colorizer. Ultimately, the output if this module comes from the colorizer and goes out to the vga port for display.

To have the map be displayed at 640 x 480 picel as reuired, we shifted the horizontal and vertical sync signal from the VGA right by 2 ( >>2) before giving it to the bot to read the worldmap’s pixel. This resulted in the worldmap pixel only changing once for every 4 changes of the sync signals. Thus the world map is stretched 4 times.

1. **Colorizer**

The colorizer combines the icon feed from the icon module and the world map feed from the bot module. The icon feed takes priority. If there is a pixel from icon that is not transparent, that will be displayed. Otherwise, the map will be displayed. The colorizer contains constants defining the colors associated with the various icon and map codes.

1. **Clock Generate**

This is a piece of IP supplied by Xilinx that was configured to divide the 100 MHz clock down to the required 25 MHz for the VGA system.

1. **Icon Generation**
2. *Icon ROM*

The icon chosen was a picture in the Space Invader videogame. Since the image was fairly simple, we decided to fill in the Coefficient (COE) file manually. We also decided that it’s easier to have 8 icons, one for each orientation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The icon ROM was generate using Xilinx Distributed Memory Generator with:

Depth = (16 px high) x (16 px wide) x (2 bits/color) x (8 icons) = 4096 bit

Width = 2 bits/color

1. *Icon wrapper*

The icon wrapper file takes in the X and Y position of the Rojobot and the VGA’s horizontal and vertical sync to calculate address into the icon ROM. The icon Rom then looks up the pixel value stored at that address and returns to the VGA subsystem.

Since we know that the icon is exactly 16x16 pixels, and that the position of the bot occupies a 4x4 pixel grid location, the icon must be drawn at an offset of (16-4)/2 = 6 pixels in the negative x and y directions in order to make the icon center over the bot’s location. Also because we have 8 different images in the ROM, the address into the ROM for each image is simply offset by a multiple of 256 (16x16).

For the icon size, we thought that 16x16 is a good size, therefor the icon was not scaled up. However because the worldmap was scaled up 4 time, the Rojobot’s coordinates and where the icon should be displayed was no longer matched. This was ‘fixed’ by shifting the bot’s coordinates left twice (LocX/Y <<2). This was equivalent to multiply the coordinates of the Rojobot by 4 and thus allowed the bot coordinates’ scale to match with the worlmap’s.

1. **Tracking algorithm**

The Picoblaze psm assembly code was a modification of proj2demo that was provided. Since most of the input and output to ports of the Picoblaze had been provided, they were all reserved as-is in the implementation of the tracking algorithm. In the demo, the Picoblaze wait to read and accept the interrupt from the Rojobot before reading the buttons to generate the next command to the Rojobot. This was done in the ‘next\_step’ function of the code. In the actual tracking, the main change is that the Picoblaze has to issue new command base on the information receive from the Rojobot instead of the button. Therefor the ‘next\_step’ function is where we put our new algorithm.

The algorithm to allow the robot to make any direction turn is that the robot will first make a 3 x 45 degree left turn (135 degree) to face the most left direction (relative to its current heading) then start turn right at 45 degree increment until it find the line. This will potentially allow the robot to u-turn if it’s in a dead-end and couldn’t find a turn.

The Rojobot has 7 states:

Init: the bot is on a potentially new section of track and will record the current position

Probe Forward: The bot will try to move forward 1 step

Forward: The bot found a line and continue to go forward until it’s off the line

Probe Reverse: the bot didn’t find a line after probing 1 step so it will reverse and try to turn right 45 degrees

Reverse: the bot takes 1 step back to get back on the line after it went off

Turn right: the bot turns right 45 degrees

Turn left 135 degree: the bot turn left 3 time (45 degree each time)

if proxi is block

issue LSFS

else

if state = init

store current position

issue LFRF

state = probe\_forward

if state = probe\_forward

if on the line

issue LFRF

if current position = old position

state = probe\_forward

else

state = forward

else

state = probe\_reverse

if state = forward

if on the line

issue LFRF

else

isuue LSRS

state = reverse

if state = reverse

if on the line

issue LSRS

state = turnL

read&store curr orient

store #turns = 0

else

issue LRRR

if state = turnL

read curr orient

if curr orient = old orient

issue LSRF

else

update old orient

#turns +1

if #turns = 3

issue LSRS

state = probe\_forward

else

continue turn left

if state = probe\_reverse

if on the line

state = turnR

issue LFRS

read&store current orient

else

issue LRRR

if state = turnR

read curr orientation

if curr orient = old orient

issue LFRS

else

issue LSRS

state = probe\_forward