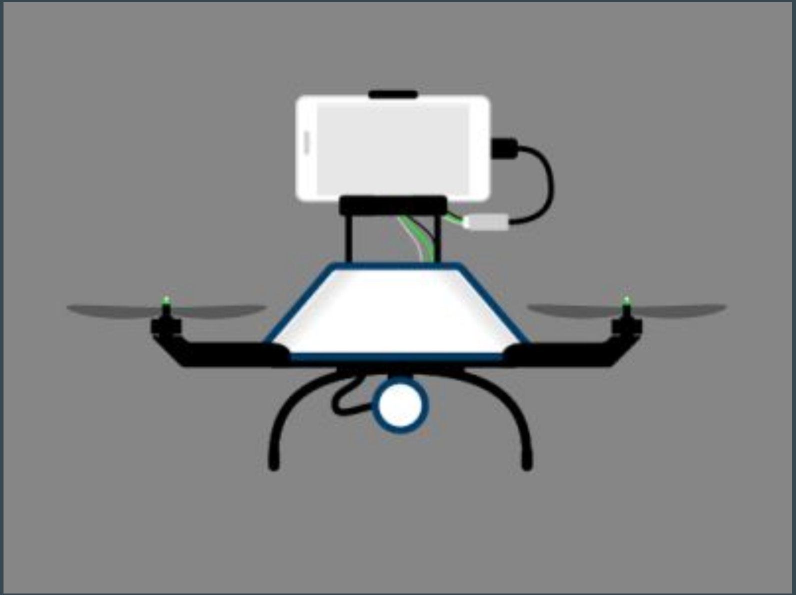
Self-Driving Delivery Robot

Final Demo Presentation

Group 5
Sheran Cardoza
Fyyaz Khan
Shahrooz Ghayouri

Project Details

Motivation

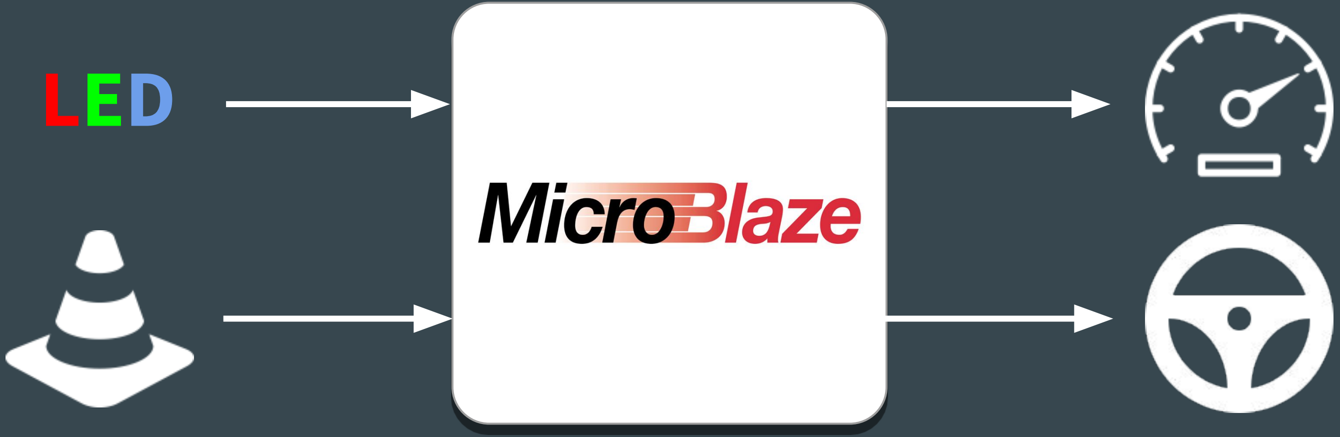


- Inspired by autonomous drones
- Emulate a drone using a self-driving robot
    - Receive commands remotely
    - Find destination
    - Navigate obstacles

Goals

	Status	Explanation
➔ Receive commands remotely ○ From a host PC over WiFi ○ Receives "LED color" of destination	🔴	No time to integrate into main project, spent most time on point light detector
➔ Find destination autonomously ○ Destinations are marked with an LED	🟡	Can spot and track LED, but presence of noise throws it off
➔ Navigate obstacles ○ Using Ultrasonic range detectors	🟢	Can avoid colliding with obstacles

System Overview



Proposed Block Diagram

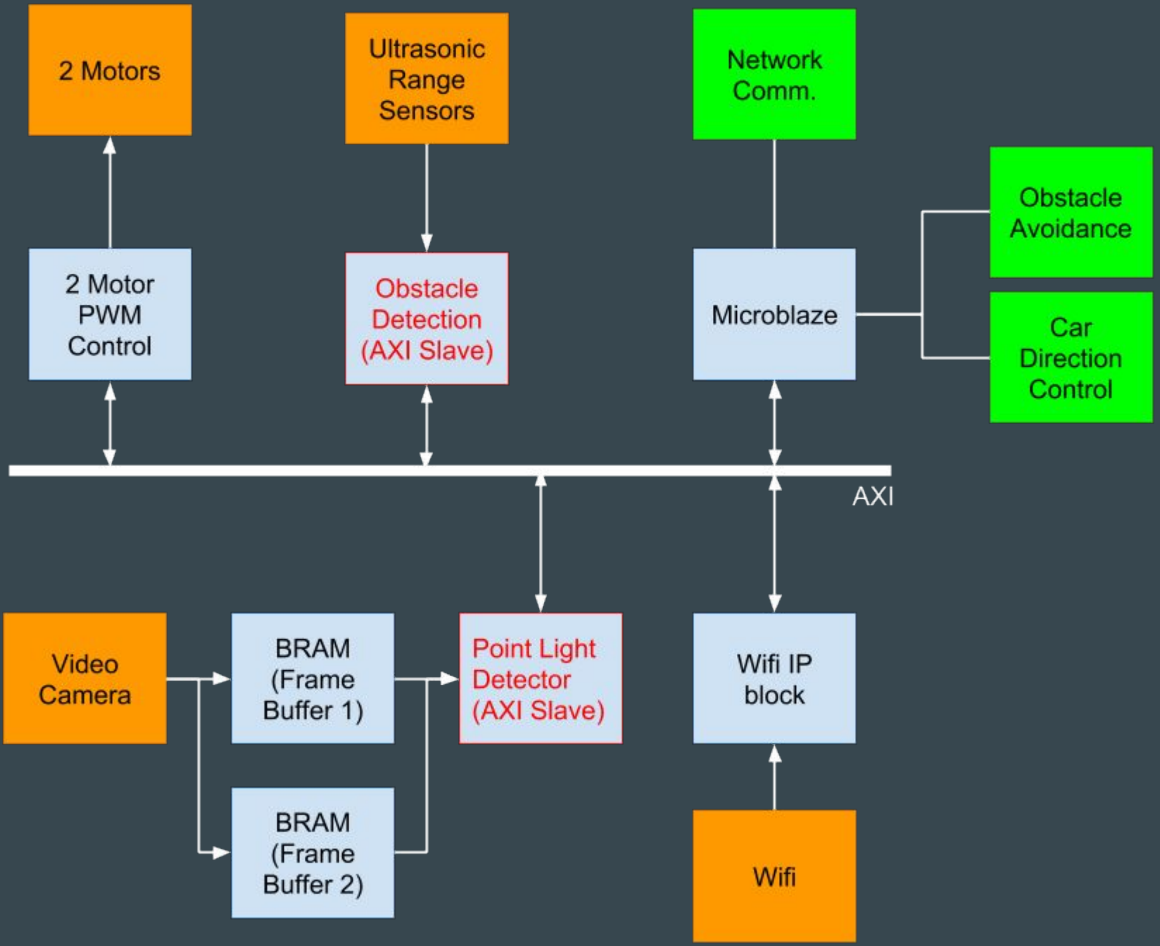
LEGEND

🟧 External Hardware (PMODs)

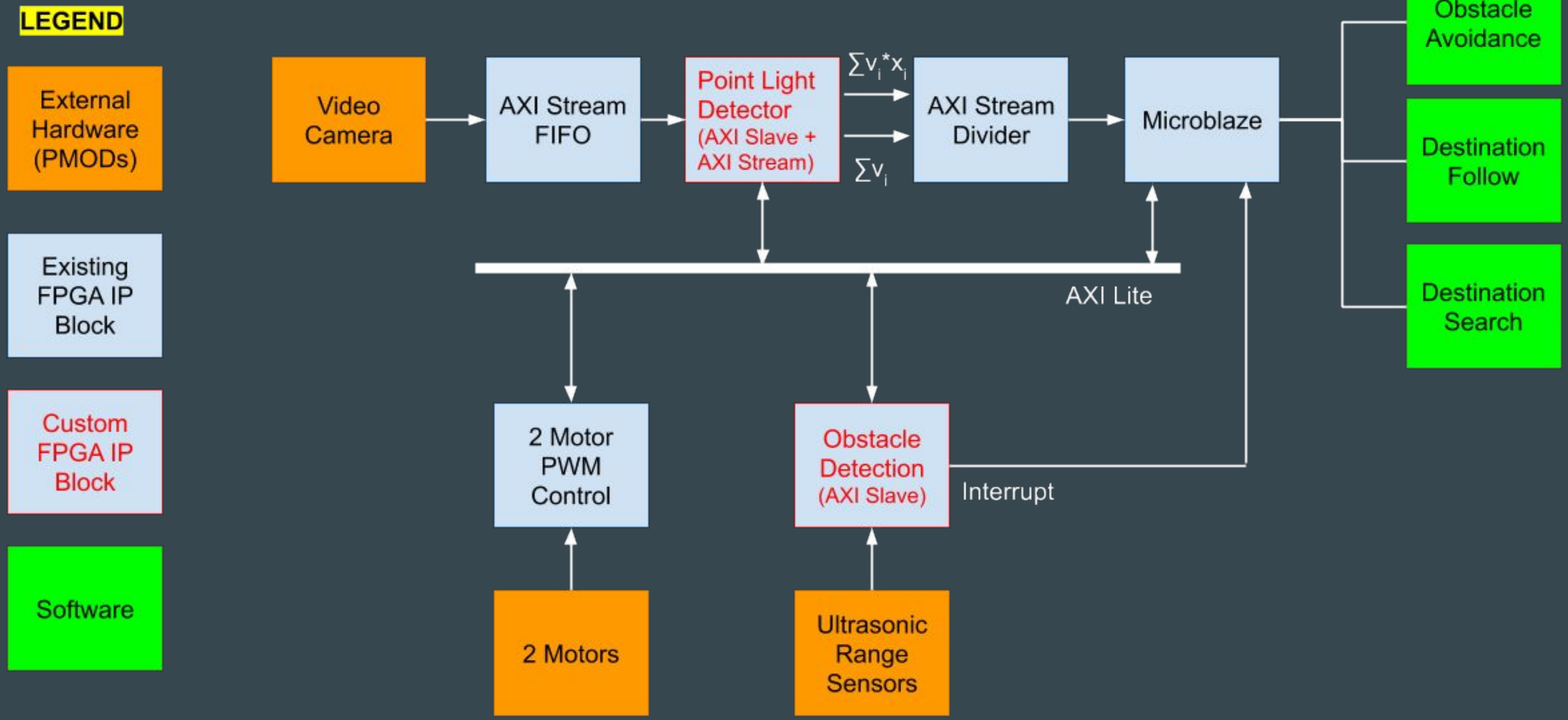
⬜ Existing FPGA IP Block

⬜ Custom FPGA IP Block

🟩 Software



Final Block Diagram



What Changed In Our Design

- Path from camera data to Microblaze
    - Changed from AXI Slave only to AXI Stream to greatly simplify the implementation of the Point Light Detector IP
    - Used AXI Stream FIFO between all AXI Stream peripherals to ensure that no packets are dropped
- Removed WiFi and Network Communication blocks due to time constraints
- Used interrupt for Obstacle Detector IP to notify processor
    - This frees up processor since it doesn’t have to poll the IP

Challenges - Design

- Camera streaming problems
  - Was getting garbage data from the camera
  - Solved by looking at signals on ILA and fixing interface between the camera and the light detector
- Unreliable output from Point Light Detector
  - To perform division, we realized we needed a proper hardware divider IP block (we were initially relying on Verilog ‘/’ operator)
  - Calculate moving average of light positions to smooth out noisy variations
  - Added several parameters which we continuously tweaked until the robot could reliably follow LED point light

Challenges - Tools

- WiFi issues in Vivado 2018.1
  - WiFi Drivers/Libraries don't build in 2018.1
  - Moved our project to 2017.4
- Interrupt issues in Vivado 2018.1
  - Interrupt port not recognized - interrupt vectors not generated in MB BSP
  - 2017.4 was better but not perfect - had to “hack” in the interrupt port
- Ultrasonic PMOD problems
  - Sometimes gets “stuck”
  - Solve by unplugging and plugging back in

Design Process

Our Code

- Obstacle Detector IP block
- Point Light Detector IP block
- “Obstacle avoidance” software block
- “Destination follow” software block
- “Destination search” software block

Not Our Code

	Modified	Source
● Ultrasonic MAXSONAR PMOD IP block	**YES**	Digilent PMOD IP library [1]
● Interrupt handler setup routines	**YES**	Xilinx Interrupt Controller example [2]
● Camera IP blocks	**NO**	Past project titled “Drag and Stamp” [3]
● PWM IP Blocks	**NO**	Digilent PMOD IP library [1]

Ensuring Success

- Used same Vivado version to develop each IP block
- Developed IP blocks in isolation, integrate one by one
- Used slave registers in IP blocks to avoid recompiling hardware design
  - Changing parameters in software allowed fast iteration
- Connected some essential signals to slave registers
  - After programming FPGA, processor can read these signals to ensure everything is still working
- Used ILAs to probe signals

Managing Multiple Designers

- Divided design into distinct components that can be individually tested
- Packaged components into IP blocks that can easily be distributed and included into each designers project
- Ensured we were aware of the interfaces (external signals exposed) of the IP blocks each designer was developing

Conclusion

What We Learned

- Should perform more rigorous testing of components earlier in the design process
  - Ensure tested outputs are valid, robust, and repeatable
- Don't underestimate work involved in integration
  - Start important integration work earlier
- Compilation is slow
  - Use slave registers to avoid recompiling
- Sometimes design can work one day and fail the next
  - Keep important ILAs hanging around
  - Hook some vital signals to slave registers for quick evaluation of failure symptoms

References

[1] https://github.com/Digilent/vivado-library/releases

[2] https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/intc/examples/xintc_example.c

[3] https://github.com/mankaijon/G1_DragandStampSystem