ECE532 Group Final Report

Regarding

Misplaced / Suspicious Package Detection and Alert System

Robert Kolaja
Albert Le
Tony Wang

April 8, 2019

TABLE OF CONTENTS

1 Overview

This section provides a high-level overview of the project and covers our motivation, block diagram, and the IP we used and/or created.

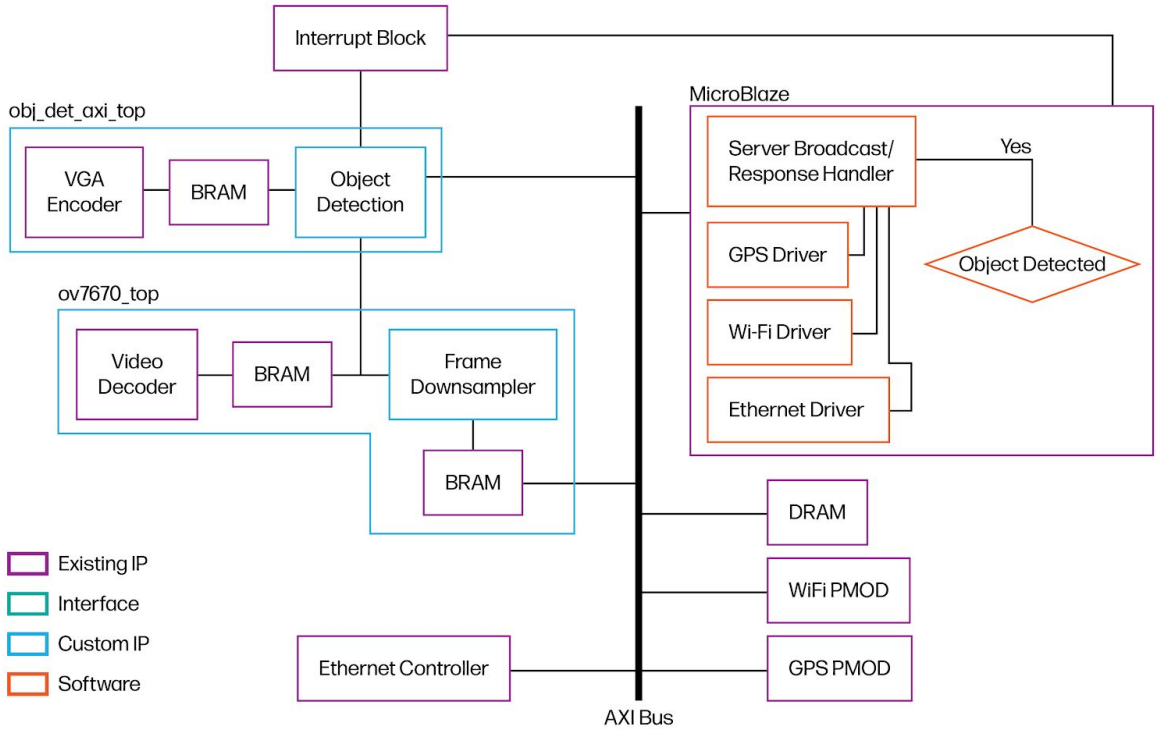
1.1 Motivation

Safety is a major concern for public events, venues and transit. Many surveillance cameras out there today capture video but do not identify suspicious packages nor issue alerts. To identify suspicious packages, public services such as GO Transit and TTC rely on public announcements encouraging individuals to report suspicious packages directly to an employee. We believe that the current method is neither ideal nor efficient. There are projects out there today that implement facial recognition and tracking in the public domain. This methodology provides complete monitoring of public spaces, however, it is also very intrusive. We aim to develop an embedded system with the sole purpose of identifying suspicious packages without incorporating the intrusive and often controversial aspects of tracking.

1.2 Project Goals

The goal of the project is to develop an embedded system to detect suspicious packages in public spaces in real time without human assistance. When a suspicious package is detected, an alert, supporting frame and GPS data will be sent to the remote system.

1.3 Block Diagram



1.4 Brief Description of IP

Table 1-1 describes the IP used in the final implementation of the project, with a brief description, and its source. Original IP blocks are expanded on in Section 4.

Table 1-1: Summary of block-level IP used

IP Name	**Module Name**	**Description**	**Source**
AXI Ethernet Lite	axi_ethernetlite_0	Provides Ethernet interface for MB via the AXI4-Lite.	Xilinx
AXI Timer	axi_timer_0	Timer to count and interrupt MB.	Xilinx
AXI Uartlite	axi_uartlite_0	Provides the controller interface for asynchronous serial data transfer.	Xilinx
Clocking Wizard	clk_wiz_1	Generates 100 MHz, 200 MHz, 50 MHz, and 25 MHz single-ended clocks used by the system	Xilinx
Microblaze Debug Module	mdm_1	Enable JTAG-based MB debugging.	Xilinx

MicroBlaze	microblaze_0	Soft processor that runs firmware to control and read from Object Detection Unit and send GPS and alert over WiFi to client on a PC.	Xilinx
Memory Interface Generator (MIG 7 Series)	mig_7series_0	Provides interface to the DRAM.	Xilinx
Ethernet PHY MII to Reduced MII	mii_to_rmii_0	Interfaces Ethernet PHY MII to RMII	Xilinx
Object Detection Unit with AXI	obj_det_axi_top_0	Original IP to take in 16-bit pixels from OV7670 Top correlates them with previous frames to detect suspicious objects in the frame. Contains an AXI slave interface to program the IP.	Team
OV7670 Top	ov7670_top_0	Interfaces with the PMOD OV7670 Camera. Contains original down sampler custom IP to lower frame rate and trigger Object Detection Unit	Obtained from Piazza, modified by Team.
Pmod GPS IP	PmodGPS_0	Interfaces with the PMOD GPS	Xilinx
Pmod WiFi IP	Pmod_WIFI_0	Interfaces with the PMOD WiFi	Xilinx
Processor System Reset	rst_clk_wiz_1_100M	Generate active high and active low reset for MicroBlaze, AXI interconnect, and peripherals. Triggered by CPU reset.	Xilinx
	rst_mig_7series_0_81M	Generates reset for MIG 7 Series interface. Triggered by CPU reset.	Xilinx

2 Outcome

2.1 Results

The core of the project functioned as we hoped: the system is able to detect objects which remain in view for a reasonably-long period of time, set a alert when a suspicious object is detected, and broadcast frame data and GPS data to client.

In our final design, the frame at the time the object was detected would be sent to the client rather than sending the frames before and after the alert is raised. Unfortunately, the sent frame was a solid RGB colour and this bug could not be resolved before the conclusion of the project. Due to unexpected networking issues, the remote user must connect to the system to ask for the alert rather than the alert being broadcast to the remote user the moment it is raised.

Table 2-1 below lists the completion status of all the features outlined in the Project Proposal. The feature list in the table was compiled using both the *Functional Requirements* and *Acceptance Criteria* listed in the proposal.

Table 2-1: Summary of Feature Completion Status

Feature	**Completion Status**
Detect a Suspicious Package with Dimensions 56 x 36 x 23 cm (Approximate Size of Carry-on Luggage)	**Complete**; The system flags stationary objects if they remain in view for at least 20 seconds. The final system shall now detect any stationary object in its field of view for 20 seconds. The detected object must occupy at least 1/16th of the camera's field of view after being passed through the black and white threshold in order to be registered as an object by the algorithm.
Save the Frame in which the Image First Appears Along with a Corresponding Timestamp	**Complete**; The frame at the moment of detection is saved for transmission to client. Timestamp requirement was dropped as the frame would be transmitted immediately.
Send an Alert to a Remote Server when a Suspicious Object is Detected	**Complete**; The system is capable of sending alert information to a remote user, however, the user must poll the system to obtain this information.

Send Timestamp, GPS Coordinates, and Video Data from both Before and After the Object was Placed in the Scene	**Partially Complete**; GPS coordinates can be broadcast, but no timestamps were generated. Frames can be sent to remote client via TCP, however sent frames were a solid RGB colour. Major difficulties were encountered with networking. See section 2.3.3 for more.

2.2 Improvements

The following is a list of improvements that should be made to the project:

- Resolve the following issues
    - Run client code (rather than server code) on the MicroBlaze and send alerts the moment the object detection core raises its flag
    - The issue where the transferred frame to client is a single RGB colour

2.3 If We Could Start Over

If we had the opportunity to start this project again from scratch, we would do the following things differently:

2.3.1 Proper Version Control

The project was built using a rather unconventional development approach. This meant moving large archives back and forth between team members and manually tracking and versioning source files and packaged IP. *If we could start over*, we would use a proper version control system (i.e. Git) to keep track of project specific source files. Git submodules could be used to manage packaged IP given their inherent nature of being decoupled from actual projects.

2.3.2 Better Interface and Signal Planning

While we did have a complete macro block diagram, we did not fully develop specific interfaces until some of the Verilog was already written. This led to a large amount of back-and-forth discussion between the code authors as we tried to figure out a suitable interface structure. *If we could start over*, the interfaces on *all* of the custom IP would be determined by ensuring the functionality is understood. Without knowing exactly what each block is supposed to do, it is difficult to fully define its interface since there may be unforeseen external dependencies (e.g. input data, flags, ready bits).

2.3.3 Better Resource Management

A lot of time and effort was put in to leverage the DMA to transfer frames from the frame buffer to DRAM to alleviate the load on the MB. Although we were able to get the DMA working to transfer frame data, the DMA was not included in our final design due to unforeseen issues. A FIFO was inserted between the frame buffer and DMA due to interfacing issues when the frame buffer and DMA when connected directly. Our design included writing an entire frame to FIFO for transfer. However, we did not realize that the FIFO was too small to fit an entire frame until

late into the project. Not being able to fit the entire frame into FIFO resulted in synchronization issues as the frame could be overwritten in the frame buffer. A solution would have been to double buffer. However, this solution was not feasible as our final BRAM utilization was at 97%. *If we could start over*, we would better plan the resources at the beginning of the project. In addition, we would have used the video board to give ourselves more resources to work with.

3 Project Schedule

This section discusses our project schedule and compares our originally-proposed milestones with what was actually accomplished on a weekly basis.

Table 3-1: Comparison of Original and Actual Milestone Accomplishments

Milestone	Original Goals (From Proposal)	Actual Accomplishments
1	Be able to display GPS data in the SDK terminal.	**Incomplete**; Deferred to M3
	Be able to send messages from the FPGA to the server via WiFi.	**Incomplete**; Deferred to M2
	Be able to receive and print on the terminal messages from the FPGA via WiFi.	**Incomplete**; Deferred to M2
		Completed Having had trouble completing the Warm-up Demo, M1 was changed to be solely the completion of the warm-up.
		Completed A Python implementation of the object detection algorithm was developed to test a variety of filters on images of different resolutions.
2	Be able to send GPS data to the server via WiFi.	**Partially Complete** The GPS PMOD block was added to a simple block design but difficulties were encountered while trying to read coordinate data.
	Deferred from Milestone #1 Be able to send messages from the FPGA to the server via WiFi.	**Completed** WiFi PMOD was integrated into the block design and scripts were developed to send messages from

		the server running on the MB to remote client.
	Deferred from Milestone #1 Be able to receive and print on the terminal messages from the FPGA via WiFi.	**Completed** Messages from the remote client were sent to MB and printed in SDK terminal. An image echo system was created and tested. The sample image (720 x 810 px) was sent and returned over TCP to a Python script running on a laptop.
	Be able to write camera image data into buffer.	**Incomplete**; Deferred to M4
	Be able to send interrupt MicroBlaze after buffer is written.	**Incomplete**; Deferred to M6
3	Be able to detect the existence of a completely visible arbitrary sized rectangular shaped box in an empty room.	**Partially Complete / In Progress** The Python implementation of the detection algorithm was completed. Design work on the hardware implementation of the algorithm commenced.
	Be able to send alert message to server after detecting box.	**Partially Complete / In Progress** The Python implementation was able to produce an alert, but this functionality was *not* yet implemented in hardware.
		Completed PMOD Camera Interface and VGA Display blocks were added to a Vivado Block Design and tested. Valid camera data was displayed on a connected display.
	Deferred from Milestone #1 Be able to display GPS data in the SDK terminal.	**Completed** A block design to integrate the GPS PMOD and relay information out via the board's USB port via UART was built. Accurate GPS readings were obtained and visible in the SDK terminal.
4	Be able to detect the existence of a completely visible 56 x 36 x 23 cm duffle	**Requirement Repealed** The size requirement was removed.

	bag in an occupied room with people moving.	**New Requirement:** The system shall now detect any static object in its field of view. The detected object must occupy at least 1/16th of the camera's field of view after being passed through the black and white threshold in order to be registered as an object by the algorithm.
	Continued from Milestone #2	**In Progress** Work began on sending meaningful data (i.e. GPS coordinates) over WiFi.
		Partially Complete The hardware implementation of the object detection algorithm begins to take shape. Grayscale conversion is functional and the system accepts threshold values set via the board's dip switches.
	Deferred from Milestone #2 Be able to write camera image data into buffer.	**Partially Complete / In Progress** Began integration of the DMA into system to transfer data. Able to transfer data to and from AXI stream FIFO and DRAM. AXI stream interface implemented into camera IP to transfer frames from IP to DRAM.
5	Be able to selectively buffer frames before and after the package was detected.	**Requirement Repealed** The requirement to send frames before and after detection was removed. **New Requirement** Send the frame at the moment when object is detected.
	Be able to send buffered frames to the server.	**Incomplete**; Deferred to M7

	Be able to playback/display the buffer frames on the server.	**Incomplete**; Deferred to M7
	Continued from Milestone #4	**Complete** Able to broadcast GPS data to the remote client.
		Redesigned and simplified hardware implementation of suspicious object detection algorithm (custom IP) to reduce BRAM utilization from 89% to 30%. Began testing with real camera data.
		Redesigned the camera IP AXI stream interface to send data to FIFO when complete frame is written to BRAM. Changed data transfer path: camera IP ⇒ FIFO ⇒ DMA ⇒ DRAM
6	Fine tuning and testing complete system	**Partially Complete** Completed hardware implementation of object detection algo and tested object detection on live camera from PMOD. Can detect foreign objects placed in front of the camera. Detecting suspicious objects is not working currently, possibly due to high frame rate or too much noise.
	Deferred from Milestone #2 Be able to send interrupt MicroBlaze after buffer is written.	**Partially Complete** Tested AXI interrupt handler by connecting dummy GPIO module to interrupt concat block. Was able to use the GPIO and INTC drivers to accept interrupts from the GPIO block. Implemented basic test handler routine to update the value of the board LEDs whenever the switch value changes.
		In Progress Completed hardware implementation of down sampler module to feed the object detection IP with frame at a

		lower frame rate. Began integrating the down sampler, camera and the AXI Stream + FIFO IP with object detection IP.
7	*Deferred from Milestone #5* Be able to send buffered frames to the server. Be able to playback/display the buffer frames on the server.	**Partially Complete** DMA + FIFO implementation was dropped as there were synchronization issues. AXI interface was added to down sampler to use MB to transfer frame to DRAM as the alternative. Able to transfer target frame over ethernet and save to disk of remote client. Image could be displayed on remote client. However, target frame was a solid RGB colour.
	Final system testing and refinement	**Complete** Completed hardware implementation of object detection algo and tested object detection on live camera from PMOD. Can detect foreign objects placed in front of the camera and identify when a foreign object is classified as suspicious. Completed testing of sending alert over WiFi when suspicious object detector flags a suspicious object along with the GPS data. We were unable to obtain a GPS signal on campus, so the number of satellites found were sent instead.

4 Description of Blocks

This section will describe the IP blocks in more detail, with a focus on the original IP designed by the group.

4.1 OV7670 Top

The OV7670 Top module (ov7670_top) interfaces directly with the PMOD camera's Serial Camera Control Bus (SCCB). It deserializes the 8-bit pixel stream from the OV7670 camera into

a 16-bit serial pixel stream. It is a modified version of the **ov7670_top** module in the PMOD camera ZIP folder uploaded to Piazza. Changes include:

1) Adding a Frame Downsampler module and Downsampled Frame Buffer

   The downsampler counts VSYNC pulses from the camera and the **capture_addr** bus of the decoder to decimate the frame rate by a set amount. To degrade the frame rate by a factor *N*, the module will read the entire BRAM Frame Buffer every *N* VSYNC pulses and pass this data to the object detection unit and the Downsampled Frame Buffer. The **obj_det_wren** output is used as a read enable on the BRAM frame buffer and a write enable on the Downsampled Frame Buffer. This enable is outputted to the Object Detection AXI Unit to signal that the data on **obj_det_pixel** and **obj_det_addr** are valid.

2) Adding an AXI Slave interface with slave registers holding a read address for and the read data from the Downsampled Frame Buffer.

   It is used by the MicroBlaze to read the downsampled frames, when it receives a suspicion alert from the Object Detection AXI Unit.

Figure 4-1 shows the block diagram of this module, with the downsampler integrated. While the connections between the camera decoder and frame buffer were mostly the same, the **we** (write enable) output of the camera decoder previously unused. To ensure that all frame buffers in the design were synchronized and only storing valid data, the **we** output was outputted to the **capture_wren** signal which gates the write port of the frame buffer. All other frame buffers in the design, including the Object Detection AXI Unit have their write enables controlled by the **obj_det_wren** outputted by the frame downsampler.

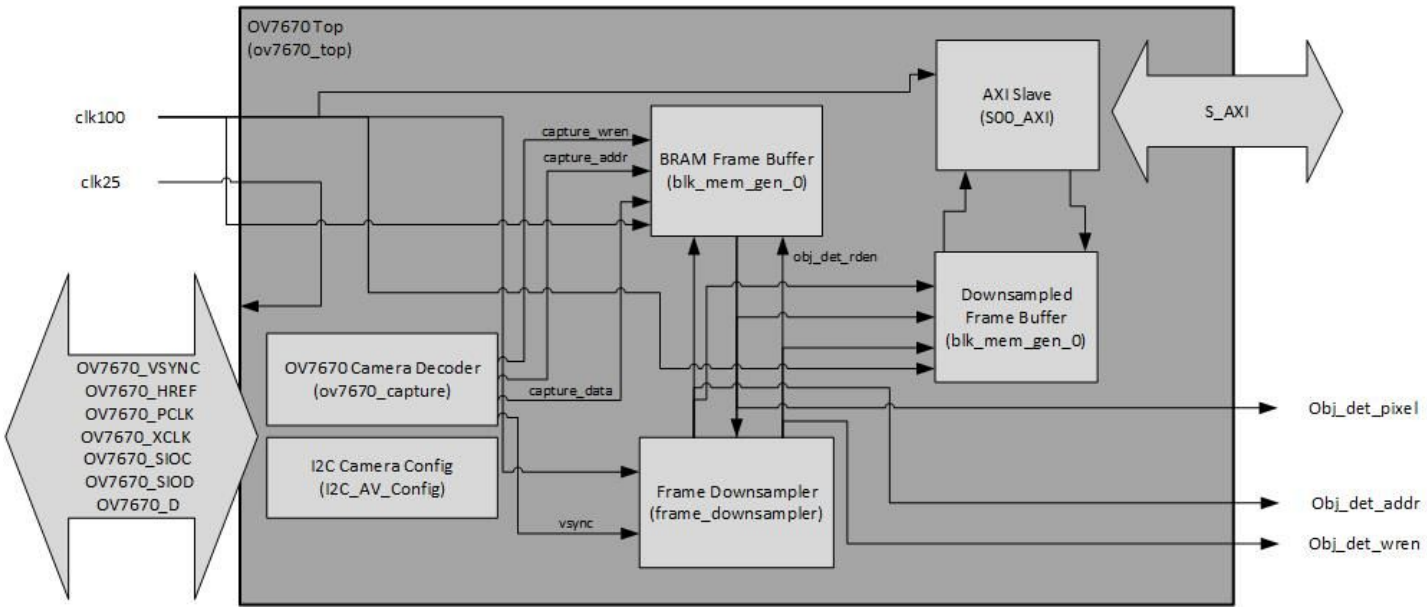


Figure 4-1. Block diagram of OV7670 Top Module

4.2 Object Detection AXI Unit

The Object Detection AXI Unit (obj_det_axi_top) is the core image processing unit of the system.

Figure 4-2 shows a block diagram of the unit. It consists of the suspicious object detection logic (obj_det_unit_top), VGA encoder to display intermediate results on a VGA monitor, and an AXI Slave interface to hold the IP's status and control registers. With the exception of the AXI Slave interface, BRAMs, and VGA encoder (vga444), all other modules and code are original.

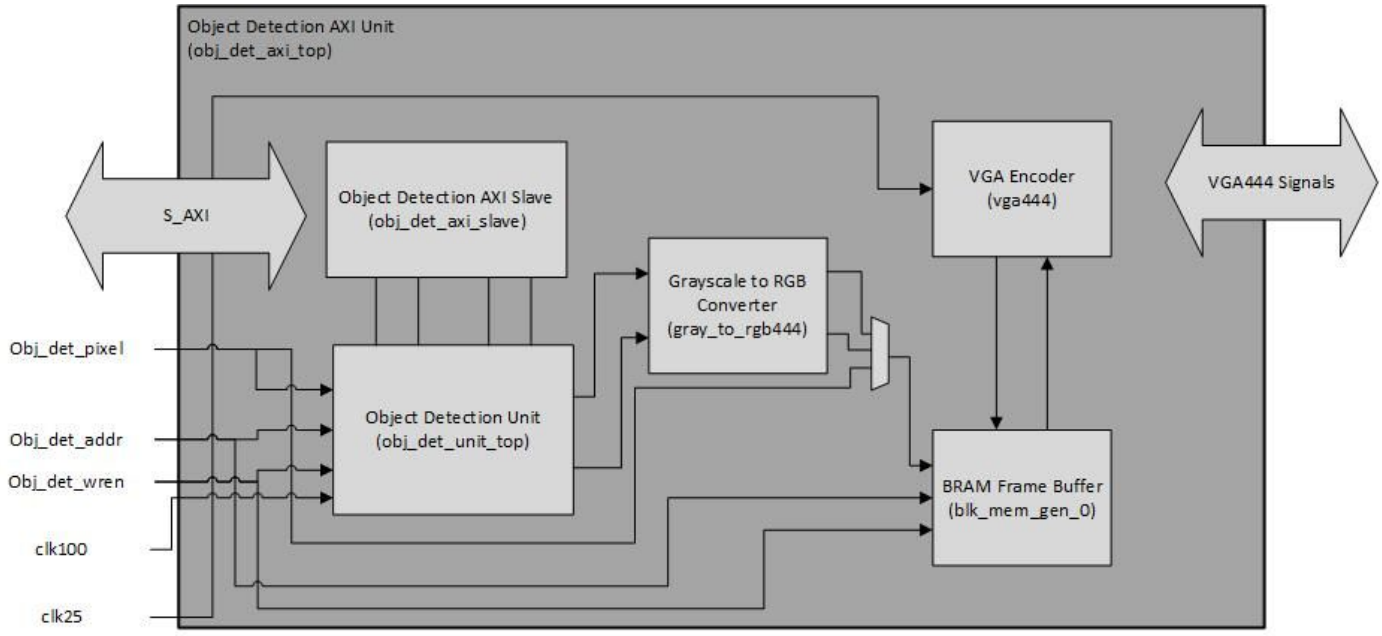


Figure 4-2. Block diagram of the object detection AXI unit

The module takes in 3 primary inputs from the OV7670 top module: the 16-bit RGB444 pixel value (**obj_det_pixel),** the 17-bit address of the pixel (**obj_det_addr**), and a write enable (**obj_det_wren**) which indicates that the pixel value is valid. These inputs are connected straight through to the Object Detection Unit (**obj_det_unit_top**) which runs the detection algorithm. There are two clock inputs: clk100 (at 100 MHz) and clk25 (at 25 MHz). The 100 MHz clock runs the AXI slave, the object detection unit, and the write port of the BRAM frame buffer. The 25 MHz clock is required to run the VGA encoder and the read port of the BRAM frame buffer. To avoid read/write conflicts, the write port of the frame buffer uses a write enable signal (**obj_det_wren**).

The Object Detection Unit (which will be elaborated on in section 4.2.1), has two internal BRAM's both of bit depth 1, and capacity 76800. One, the reference buffer, stores the reference frame that all subsequent frames will be compared to, and the other, the difference buffer, stores the difference of the previous frame to the reference. The read data ports of these two buffers are outputted and put into a mux along with the raw RGB pixel data, which controls frame is written to the BRAM frame buffer. The VGA encoder reads this frame buffer and outputs frames to a VGA monitor, allowing for the internal memory of the algorithm to be monitored in real-time during debugging and configuration.

The Object Detection AXI Slave is a customized AXI slave which holds 4 read/write registers, and 1 read-only register, used to configure the Object Detection Unit and read its status bits, such as suspicious object detected. Among the customizable parameters include: black and white threshold, object detection pixel threshold, suspicion frame threshold, and the starting of the algorithm. See Appendix A for the full register map of the AXI Slave.

4.2.1 Object Detection Unit

The object detection unit contains modules to perform the grayscale and black/white preprocessing on incoming pixel data which is then passed to the Suspicious Object Detection FSM. The connections are shown in Figure 4-2-1.

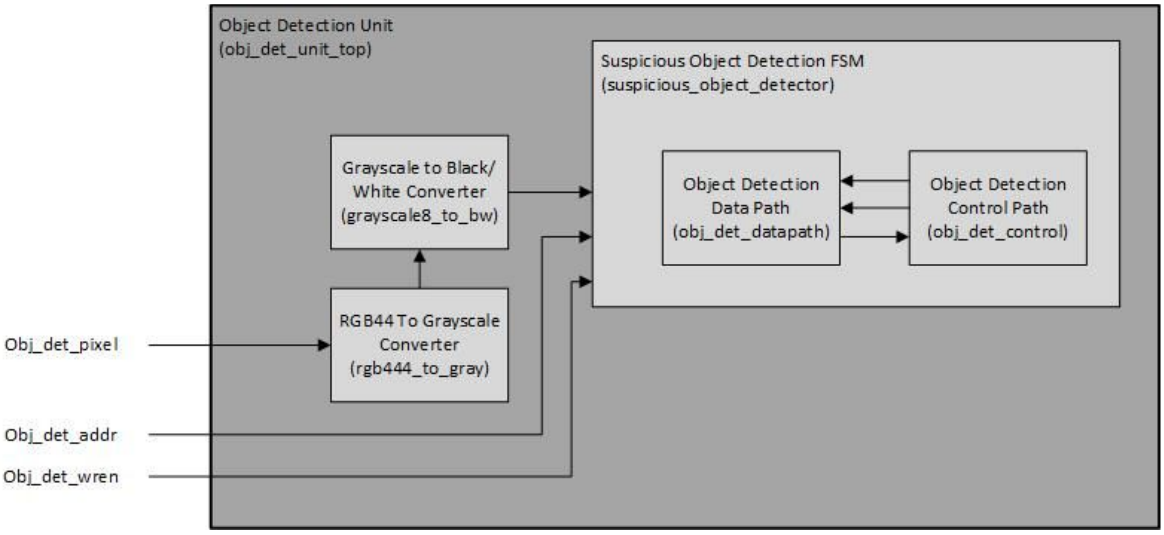


Figure 4-2-1. Block diagram of the object detection unit

In the software implementation of the algorithm, an 8-bit grayscale value was passed through the binary threshold for black and white conversion for the preprocessing step. For every incoming 16-bit RGB444 pixel, the following algorithm was applied, through the **rgb444_to_gray** and **grayscale8_to_gray** modules. This algorithm is intended to emulate the floating-point RGB to grayscale equation: gray = 0.3125 * R + 0.5625 * G + 0.125 * B.

1. Zero out upper 4 bits of 16-bit RGB444 pixel to produce a 12-bit RGB444 value.
2. Separate single 12-bit signal into 3 seperate 4-bit signals: R(ed), G(reen), and B(lue).
3. Left-shift each signal by 4 to make them into 8-bit signals
4. Apply the following transformations to perform division:
    a. (R >> 2) + (R >> 4) = R * 0.3125
    b. (G >> 1) + (G >> 4) = G * 0.5625
    c. (B >> 3) = B * 0.125
5. Sum resulting signals into **grayscale** which is the 8-bit grayscale equivalent of the 16-bit RGB444 input pixel

    Note: The hardware implementation of the algorithm in the **rgb444_to_gray** module simplifies into just **gray = R * 5 + G * 9 + B * 2.**

The resulting 8-bit grayscale pixel is then passed through a threshold, where all pixels above the 8-bit threshold are converted to black (8'd0) and all pixels below are converted to white (8'd255). The LSB of this 8-bit signal is taken as the 1-bit black and white pixel that is passed into the Suspicious Object Detection FSM.

4.2.2 Suspicious Object Detector

The suspicious object detector contains the main logic for comparing the current frame with the reference frame to identify foreign objects and compare with the previous foreign objects to determine stationary objects. Figure 4-2-2 shows the block diagram for the module with the datapath expanded to show internal blocks.

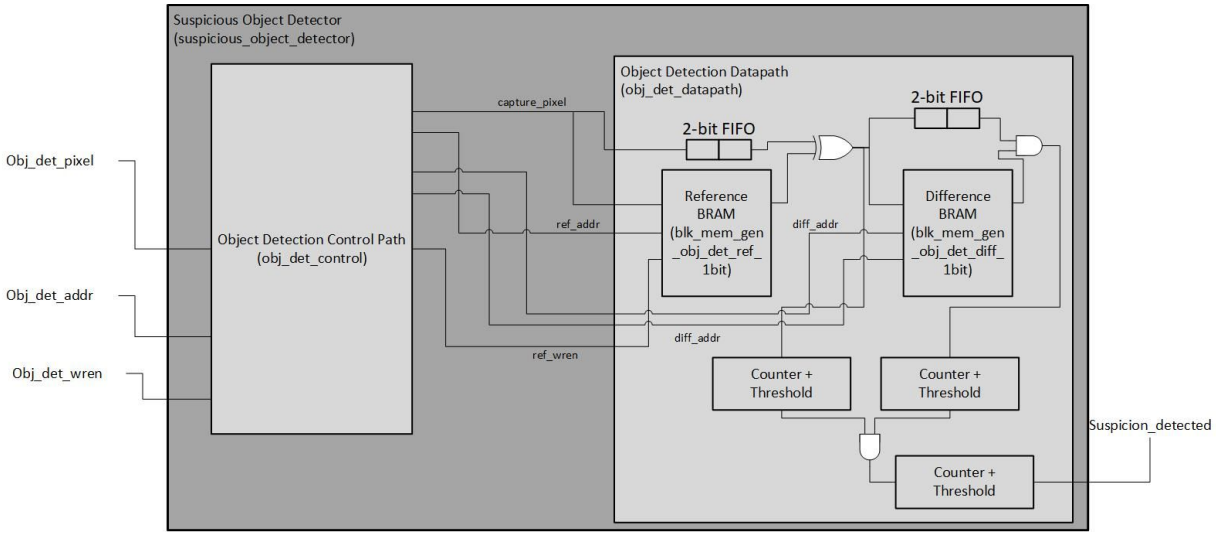


Figure 4-2-2. Block Diagram of Suspicious Object Detector

The algorithm is as follows. Note: All frames are preprocessed to and stored as 1-bit black/white QVGA (320x240) frames.

1. Take an image of the unoccupied space where we would like to monitor for suspicious objects. Store this image as the **reference frame**
2. For every subsequent frame:
    a. Compute the pixel-wise difference (XOR) between the current frame and the **reference frame.** Store this difference as the **difference frame.** This is to identify foreign objects not in the reference frame.
    b. Compute the pixel-wise similarity (AND) between the current **difference frame** to the previous **difference frame** to look for objects that remain stationary in the video over successive frames.
3. A suspicious object is identified if: An object is detected in the current frame AND This object has remained stationary for a number of frames (several seconds)

For the hardware implementation, the datapath has two 76800 capacity, 1-bit internal BRAM, each holding single QVGA black and white frames. The reference BRAM is a single port BRAM with a Write First Operating mode. The difference BRAM is single port BRAM with a Read First Operating mode, allowing the pixel value at the address of the BRAM to be read and compared to the current difference while the result of the AND gate is written to the BRAM simultaneously. Figure 4-2-3 shows examples of these BRAM contents.

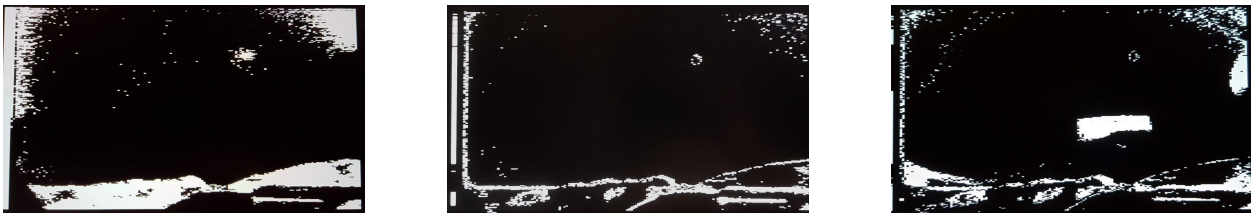


Figure 4-2-3. From left to right: 1) The preprocessed reference frame, stored in the reference BRAM 2) The preprocessed difference frame of a empty space, stored in the difference BRAM. 3) The result of the AND gate showing a foreign object in the space. This image will be written to the difference BRAM and used as a comparison point for subsequent frames.

For every frame, the number of times the XOR gate comparing the current pixel to the corresponding pixel in the reference frame results in a 1 is passed through a counter and threshold. If the threshold is passed, a foreign object is detected. The result of the XOR is compared to the previous result of the XOR gate for the previous frame with an AND gate to implement a temporal low-pass filter. The number of pixels that are the same (AND = 1) are counted, and if they pass a threshold, a suspicion counter is incremented. If at any time during the frame stream, the threshold is not met, the suspicion counter is reset. Once the suspicion counter reaches a threshold, the alert is triggered and the corresponding bit in the AXI register is set, so it can be polled by the MicroBlaze.

5 Design Tree

The key files of our project is listed below:

Design: Main Vivado Project
Note: Bolded lines indicate modules generated from the IP Catalog, such as the Block Memory Generator. These folders include the .xci and .xml files for the IP.

- Block Diagram
    - Bd
        - design_1.bd
        - design_1_ooc.xdc
- Design Sources

- design
  - src
    - ip
      - pmod_camera
        - I2C_AV_Config.v
        - I2C_Controller.v
        - I2C_OV7670_RGB444_Config.v
        - ov7670_capture.v
        - vga444.v
    - new
      - obj_det_axi_top
        - **blk_mem_gen_0**
        - **blk_mem_gen_obj_det_diff_1bit**
        - **blk_mem_gen_obj_det_ref_1bit**
        - **obj_det_axi_slave_0**
        - obj_det_axi_top.v
        - rgb444_to_gray.v
        - grayscale8_to_bw.v
        - gray_to_rgb444.v
        - vga_rgb_to_grayscale.v
        - suspicious_object_detector.v
        - obj_det_control.v:
        - pulse_detector.v
        - Obj_det_datapath.v

Software

- GPS_WIFI
  - SDK
    - gps_wifi
      - main.cc
      - gps.c
      - gps.h
  - Client
    - IoT_client_gps.py

- Frame_Broadcast
  - SDK
    - frame_broadcast

            - main.c
            - send.c
        - frame_broadcast_bsp
            - tcp.h
            - tcp.c
    - Client
        - IoT_client_frame.py

6 Tips & Tricks

To future students…

Be sure to plan out all interfaces and agree on your datapath *before* writing any code. This way, you don't waste any time doing work that could have been avoided by simply putting pen to paper (yes, paper) and defining every block at the interface level.

Start integrating early. While it is a good idea to get things working in isolation, by integrating sooner rather than later, you'll avoid having to deal with a potential snowball effect at the end of the project.

Do not underestimate SDK's ability to throw nonsensical errors. For example, sometimes the application would report missing header files when those files actually exist. In the best case, simply deleting and re-adding the `#include` line will solve the problem. Other times, re-generating the board support package will do the trick. In the worst case, re-running synthesis and implementation may be necessary.

Appendix A: Object Detection AXI Slave Register Map

Base Address	Slv_reg#	Register Name	Bits	Field(s)	Default Value (Decimal)	R/W	Description
XPAR_OBJ_DET_AXI_TOP_0_BASEADDR	slv_reg0	CONFIG1	[31:16]	DECIMATION_RATE	16'hFFF0 = 16'd65520	R/W	Number of frames to drop between captured frames. Used to downsample the incoming frame stream from the camera. Downsampled frame stream is used in algorithm and stored in dedicated BRAM to store downsampled frames
			[8:1]	BW_THRES	8'd112	R/W	Grayscale threshold above which pixels will be changed to black. Pixels with a grayscale below this threshold will be changed to white
			[0]	START_CAPTURE	0	R/W	Set to 1 to start algorithm
XPAR_OBJ_DET_AXI_TOP_0_BASEADDR + 1	slv_reg1	STAT1	[17]	OBJECT_SUSP	0	R	Output that goes high when suspicious object is detected
			[16]	OBJECT_DETECTED	0	R	Per frame output that goes high when an object is detected in a frame. Goes low between frames
			[15:0]	SUSPICION	0	R	Number of consecutive frames that contain a potentially suspicious object
XPAR_OBJ_DET_AXI_TOP_0_BASEADDR + 2	slv_reg2	DET_THRES	[16:0]	DETECTION_THRES	76800/16	R/W	The number of active pixels to signal a detected object
XPAR_OBJ_DET_AXI_TOP_0_BASEADDR + 3	slv_reg3	STATIC_THRES	[16:0]	STATIC_THRES	76800/32	R/W	The number of active pixels that must be the same between consecutive frames to be considered static
XPAR_OBJ_DET_AXI_TOP_0_BASEADDR + 4	slv_reg4	SUSP_THRES	[16:0]	SUSPICION_THRES	40	R/W	Number of frames that suspicion must be maintained to trip alert