Source Code,
UG Docs,
Full Writeup

OBJECT TRACKER

Demo Video

Educational Object Tracking System using Basys3 FGPA: Create it on your own!

EE2026 Digital Design Final Project by: Joel Ku | Joshua Yeo Wee Tze | Kenneth Wong Cun Wi | Tan Kai Cong

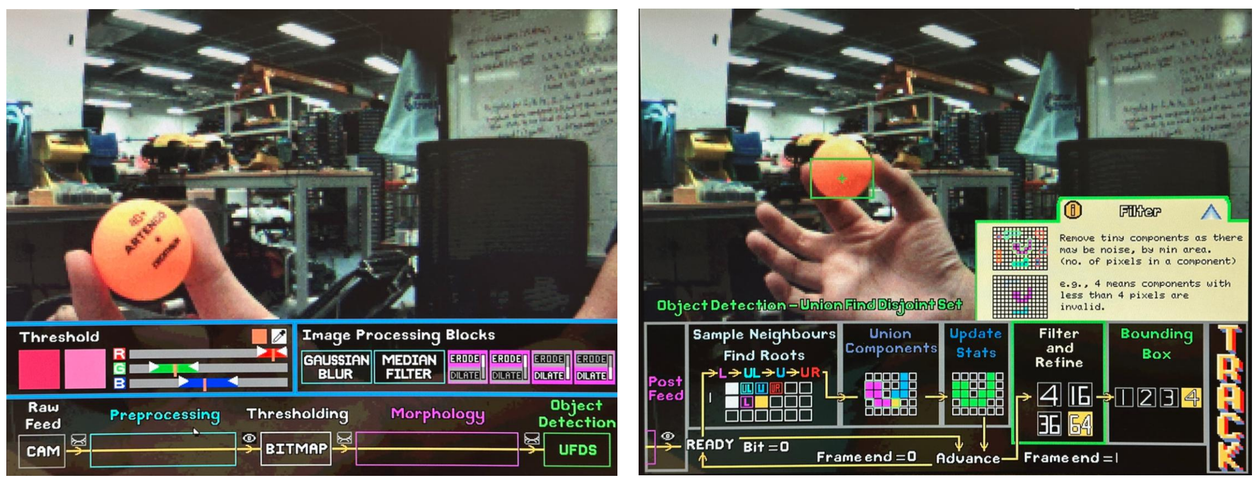

Figure 1: GUI for Image Processing and Blob Detection Settings

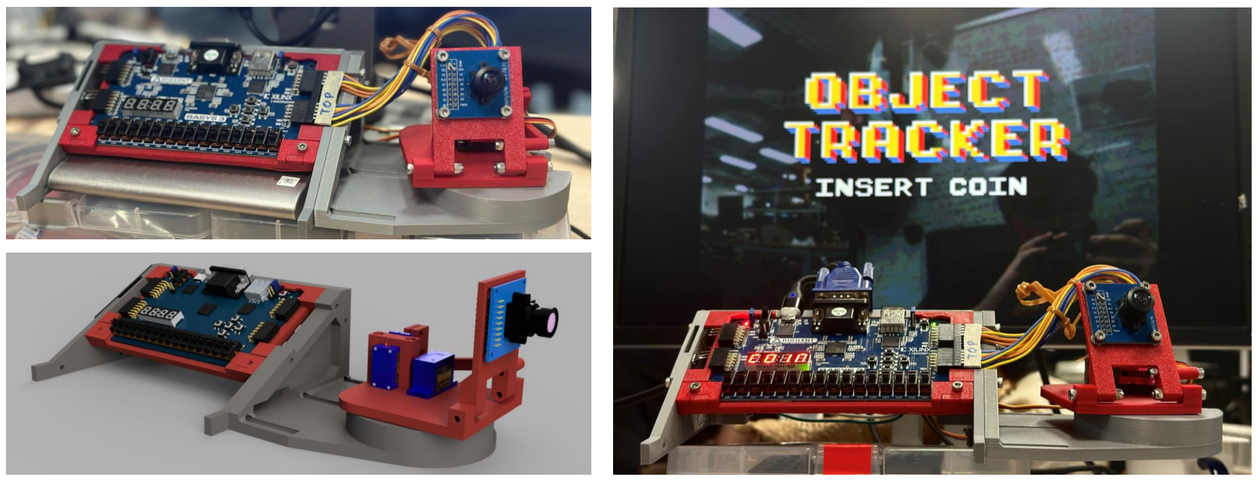

Figure 2: Final Product and Title Splash

Users can interactively customise an **image processing pipeline** via intuitive **drag & drop controls, dynamic sliders** and **scroll-wheel**. They can also gain insights into the **UFDS blob detection** algorithm through **pop-up explanations** and modify its **parameters**.

All modifications are visualised in real-time with **zero frame latency** for users to get immediate feedback on how various changes affect the system and learn the principles of object tracking.

The camera is mounted on a pan-tilt mechanism that can physically track detected blobs through PD controlled servos and tunable K_P and K_D parameters using the FPGA's switches and buttons.

The system integrates the following hardware: 1x Basys3 FPGA, 1x OV7670 Camera, 2x MG90 Servos, 1x USB Mouse, 1x 640x480 VGA display.

Overall Architecture of Computer Vision Pipeline

1. OV7670 Camera Interfacing

SCCB protocol is used to configure the camera's settings by setting its registers. Camera pixel data is streamed to the FPGA in **RGB565** format but is down-sampled into **RGB444** format by our *OV7670_Capture* module to keep within memory restrictions.

Figure 4a: OV7670

2. Preprocessing (GB/MF)

The preprocessing stage aims to perform noise reduction by applying spatial filters with 3x3 kernel size. Two operations are made available to the user:

Gaussian Blur (linear filter using a basic multiply-and-accumulate operation) and Median Filter (non-linear filter implemented with a 3-layer comparator network).

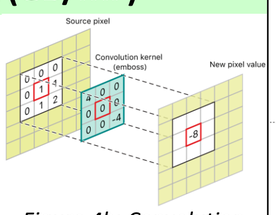

Figure 4b: Convolution

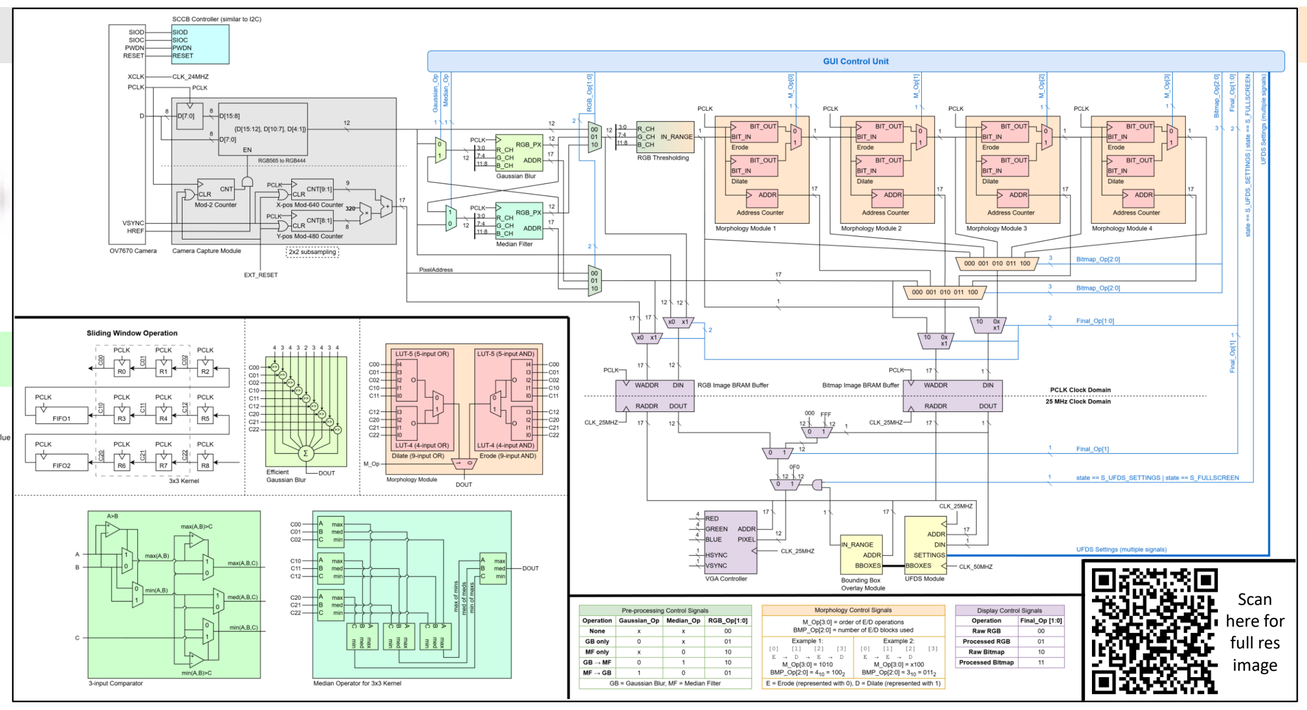

Figure 3: Consolidated Architecture Design

3. Thresholding and Morphology

Thresholding is done in RGB colour space. If a pixel's R, G, B channels fall within the user-defined ranges, it is assigned '1' in the converted bitmap, else '0'. The range is set with the GUI's eyedropper and sliding bars.

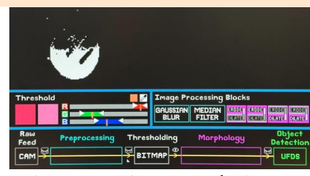

Figure 4c: Converted Bitmap

Functionally, **Erode** reduces the size of blobs, i.e., removing small noisy regions. In contrast, **Dilate** increases blob sizes, which restores the size of eroded blobs or covers holes in detected blobs. They are implemented as bitwise AND and OR of pixels in the kernel respectively. Users can apply up to four Erode/Dilate blocks in any order, using the scroll-wheel to switch between modes.

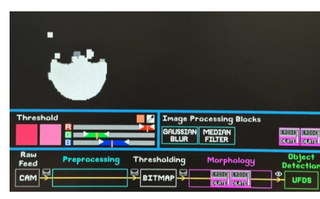

Figure 4d: Eroded Image

4. Union-Find Disjoint Set (UFDS) for Blob Detection

Architecture Overview

After processing the image, this module performs object detection using a *weighted Union-Find Disjoint Set* (UFDS) while streaming a binary image in raster order from VGA/BRAM path. It extracts object bounding boxes and centroids as the frame is processed without storing the full image. This algorithm and implementation is pivotal in achieving optimal time and space complexity given the FPGA constraints.

Clock Domain Crossing - FIFO and Decoupling Bridge

Bitmap pixels arrive at a rate of 25 MHz, while the UFDS Core runs on a faster 50 MHz clock, and the latter takes a variable number of clock cycles per incoming pixel depending on the algorithm's decision path.

Hence, an asynchronous FIFO with a bridging module was implemented. The bridging module formats the incoming data and enqueues it in the FIFO when the pixel is signalled as valid. Therefore, the UFDS Core can wait to read the next pixel only when its internal Finite State Machine (FSM) is ready without risking dropped pixels. These modules enables a safe CDC without metastability propagating false data.

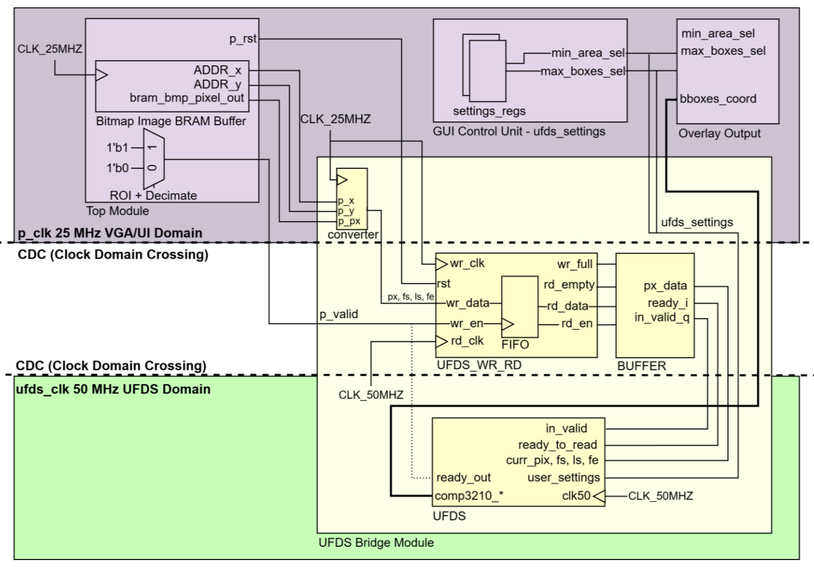

Figure 5: UFDS Subsystem Architecture Design

UFDS Core - Data Structure and Algorithm

UFDS is used to build up connected components through raster scan for one entire frame. This is achieved a pixel at a time, by *finding* neighbouring components that are separated (or *disjoint*), before *unifying* them. Register arrays with a size equal to the number of components are used to represent this *disjoint set* data structure, such that every component has its own statistics (its first pixel (or *root - RT*), area (pixel count), and coordinate sums).

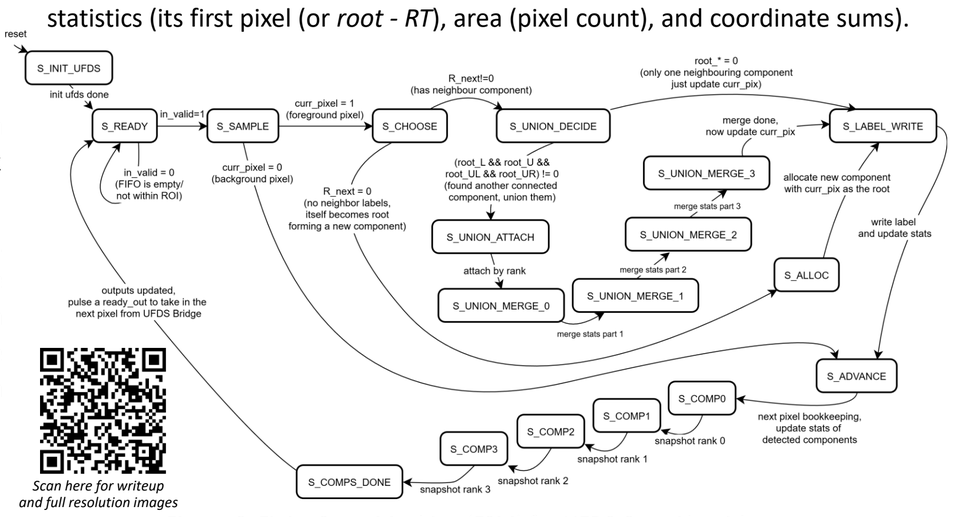

Figure 6: Finite State Machine (FSM) for UFDS Core Algorithm

Find Algorithm: For an incoming bitmap pixel (e.g. in yellow below), there would have been a block of pixels processed previously (e.g. represented in white) and some possibly disjoint components (e.g. in pink & blue).

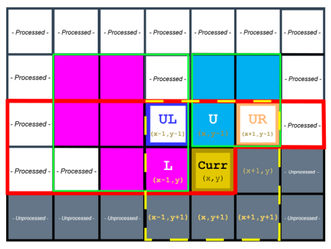

Figure 7a: S_SAMPLE

1. S_SAMPLE: In the bitmap, '0' is *background,* so it skips and advances to the next pixel. For '1', it finds components to union with by scanning its 8 neighbouring pixels (yellow dotted box).

Optimisation 1: For *8-connectivity raster scan*, since components found will only connect with the 4 processed (UL, U, UR, L) pixels, sampling just this reduced set is sufficient.

Optimisation 2: As the *disjoint set* structure is stored in separate registers, we only need to store the previous and current row of pixels. The necessary buffer size is **reduced by 99%** from the full 240 rows to 2 rows (boxed in red).

Find Algorithm (cont.):

Figure 7b: S_CHOOSE

2. S_CHOOSE: If there are no adjacent '1' pixels, the current pixel becomes the root (*RT*) of a new component. Else, for each adjacent component (pink and blue), UFDS will trace the connected pixels along the white arrows to *find* the first pixel (or *walk up the tree to find the* root *RT*), to obtain the ID of each component.

Optimisation 3: We store ID of components (e.g. '2' for pink) in every pixel to avoid *tree walk,* achieving a **~75% reduction** in clock cycles.

Weighted Union Algorithm: A pixel and its neighbouring components are *disjoint* and implies the existence of a single larger component, which will be formed by a *Union* operation on the *disjoint sets*.

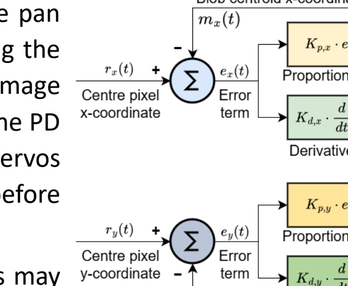

Figure 7c: S_UNION_ATTACH

3. S_UNION_DECIDE & ATTACH – ***Optimisation 4 (Weighted Union)*****:** By *attaching* smaller components under the largest component (i.e. all blue/'3' is now labelled pink/'2'), the *UFDS tree* is *shallow* at $O(\log n)$, resulting in very efficient *find* and *union* operations.

Figure 7d: S_LABEL_WRITE

4. S_LABEL_WRITE: Before advancing to output and the next pixel, the current pixel is labelled. A single larger component and its (green) bounding box is obtained through UFDS.

Optimisation 5: *Union* updates register arrays which is expensive. They are **pipelined** into multiple stages to reduce the **critical path delay** and to ensure **timing constraints are met**.

Figure 8: Output

Filter and Output: Components smaller than user-defined threshold size are most likely noise and filtered. Up to 4 bounding boxes are output to the main module for display and object tracking.

5. BRAM Usage, hardware design

Graphical elements are designed in Excel with 4-bit pixel values to denote colours. Due to the limited BRAM, graphics are generated in 3 different ways to achieve balance between LUT and BRAM usage.

More LUT usage ⟵ ⟶ **More BRAM usage**

Figure 9a: Top left of settings page

Simple fixed elements like borders and lines are generated with if-else statements

Figure 9b: Glyph of the word "Right"

Letter glyphs are stored as a custom font in BRAM and reused to generate text in the educational tab writeups

Figure 9c: Erode/Dilate blocks

Complex graphics are stored as an entire block in BRAM and displayed with reference to the top-left coordinate

6. Pan-Tilt Mechanism

The camera is mounted on a pan-tilt mechanism. The pan and tilt servos run on separate PD control loops, using the pixel distance of the object's centroid from the image centre in the x- and y-axis as the errors respectively. The PD term output provides the target velocities, but as servos are position controlled, the PD term is integrated before being mapped to the servo's PWM range.

K_p and K_d parameters are pre-tuned initially, but users may modify them using the 16 switches, mapped to four 4-bit values, namely pan servo K_p & K_d, and tilt servo K_p & K_d.

Figure 10: Feedback Control Loop