Design Document for EAN-13 Barcode Decoder with FPGA and VGA Display

Ananya Haritsa, Helen Bovington, Kamil Marian Zajkowski, Matthew Ethan Modi, Rahul Pulidindi

Table of Contents:

Overview

Our system will read EAN-13 barcodes using a camera and display the 13-digit GTIN (Global Trade Identification Number) on a VGA monitor. EAN-13 is a barcode symbology defined by GS1 to encode unique numbers to identify products. This project implements a hybrid hardware-software system for decoding EAN-13 barcodes, using the OV7670 camera and the DE1-SoC development board. The system leverages both the programmable logic (FPGA) and the integrated Hard Processor System (HPS) on the Cyclone V SoC.

Background

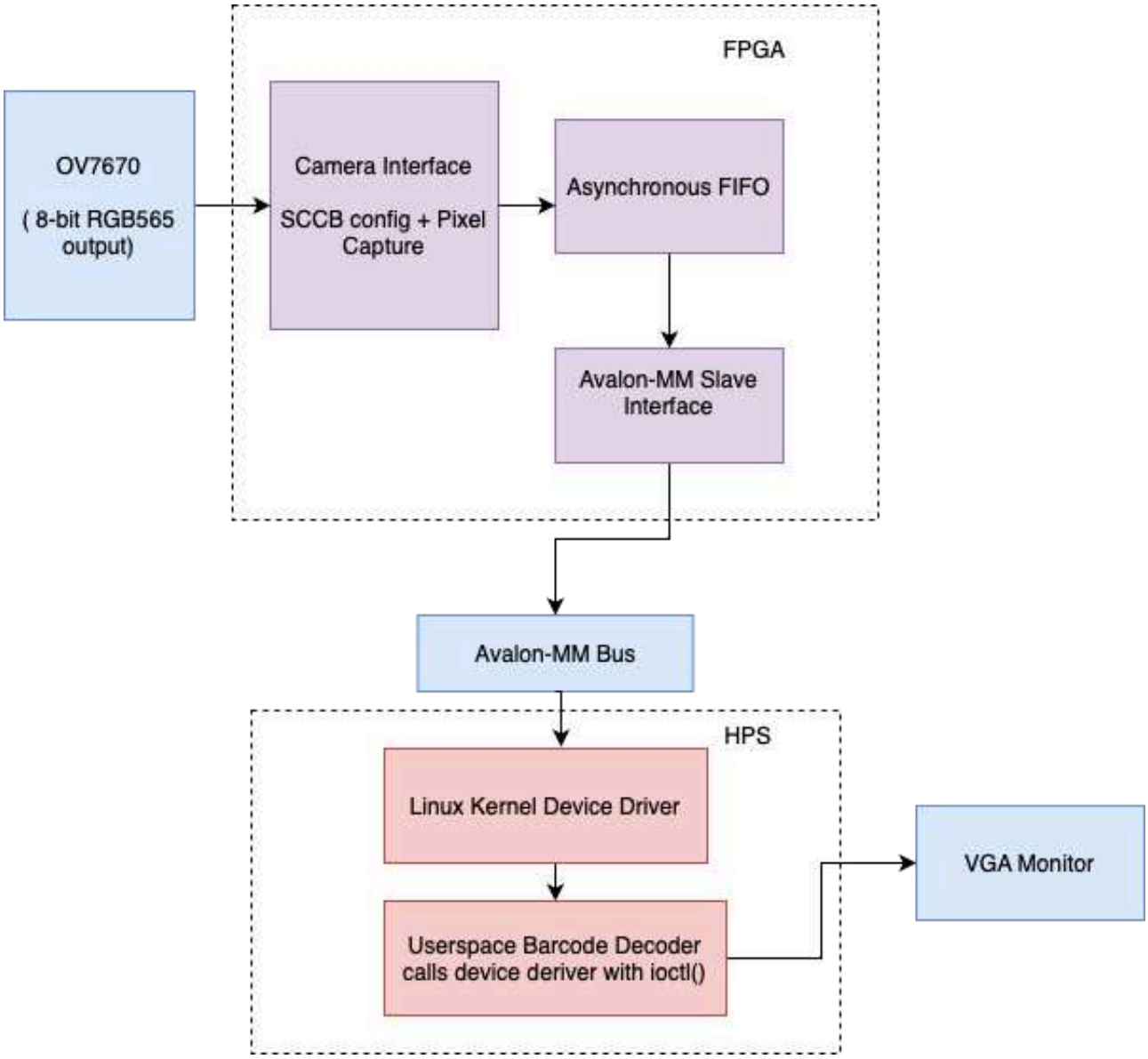
In the United States, EAN-13 barcodes are commonly used on retail goods and books. Retail goods are marked as described by the UPC-A standard with a leading “0” followed by a 12 digit identifier. Books use the ISBN standard which is a subset of the GTIN standard, where the country code is marked as 978 or 979 and commonly referred to as “Bookland”.

Each EAN-13 barcode consists of:

- A 3-digit GS1 prefix (country or organization code)
- A manufacturer code
- A product code
- A checksum digit, which is calculated using a modulo-10 algorithm for error detection.

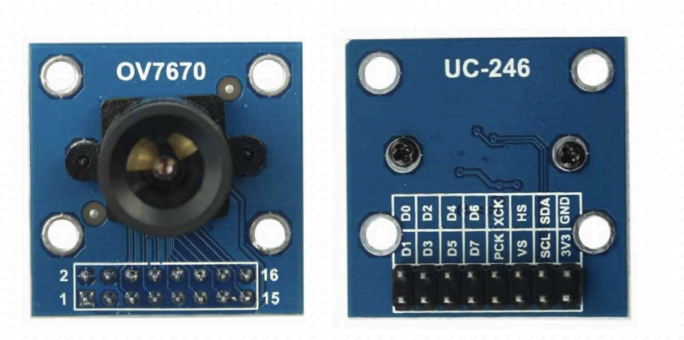
EAN-13 is designed for optical scanning, and the encoded information is not stored as characters but as a sequence of bar widths and spacings, with strict rules for start, middle, and end guards, and left/right digit parity patterns. This makes it ideal for real-time decoding from images or video frames, such as in our system. Our project leverages this predictable encoding structure to decode bar widths from a single scanline of a barcode image, allowing us to extract the 13-digit GTIN from visual data captured by an OV7670 camera. This approach mimics the working principle of physical barcode scanners and offers a hands-on demonstration of digital image processing, signal sampling, and hardware/software co-design.

Block Diagram



System Components

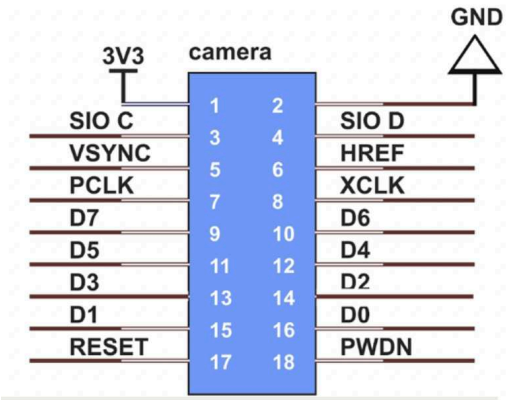
Camera → FPGA Fabric Interface



Interfacing OV7670 camera

To capture barcode image data, our project uses the OV7670 a compact, low-power CMOS image sensor that outputs 8-bit VGA video data at up to 30 frames per second. It communicates with a host device through a combination of a parallel pixel bus and a Serial Camera Control Bus (SCCB) for register configuration. The camera outputs formatted image data in RGB565 format, which requires assembling two 8-bit data values per pixel using the PCLK signal. The timing and data synchronization rely on HREF (line valid) and VSYNC (frame sync) signals, which the FPGA will monitor to capture one full horizontal row of pixel data per barcode scan.

We reviewed the OV7670 datasheet: ([OV7670 Camera Module Datasheet (Rev. C, PDF)]), which outlines key capabilities such as exposure control, gamma correction, white balance, color saturation, and hue control, all configurable through SCCB. These settings allow us to fine-tune the image quality if needed, especially for robust performance under varied lighting conditions — which is critical for accurate barcode decoding.



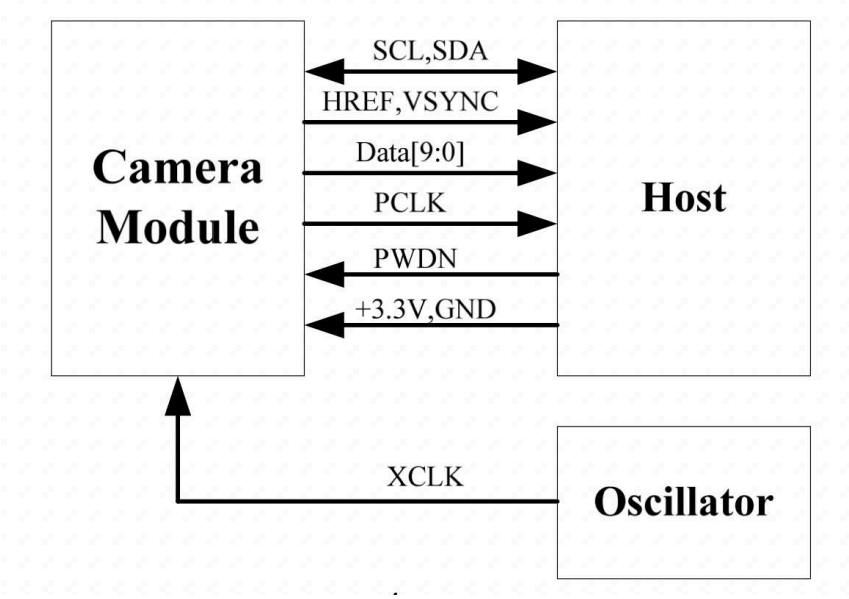
CMOS OV7670 Camera Module Pinout

To implement this interface, we plan to adapt a verified Verilog-based implementation from GitHub – camera_interface.v (AngeloJacobo/FPGA_OV7670_Camera_Interface). This codebase provides two key modules:

1. An SCCB controller FSM that sends a boot sequence of register writes to configure the OV7670 into RGB565 streaming mode.
2. A pixel acquisition FSM that captures data on PCLK rising edges, assembles 16-bit pixel values, and writes them into an asynchronous FIFO for downstream access.

Although the original design targets a Xilinx platform, we plan to port and adapt it to our DE1-SoC (Cyclone V) board by replacing vendor-specific clocking modules and interfacing logic. The camera is externally clocked using a 24 MHz oscillator signal (XCLK), which our FPGA will generate. The FIFO will decouple the 24 MHz pixel domain from the 50 MHz system logic, enabling safe clock domain crossing.

Ultimately, our design will trigger a capture on button press and save a single horizontal scanline to memory, which the HPS will later process. Additional features such as dynamic brightness and contrast adjustment via on-board keys (also part of the GitHub code) may be retained to assist with debugging and improving image quality during development.



FPGA Fabric Processing

The FPGA will perform the following functionality:

1) Capture button of camera being pressed.
   When an outside user presses the button to the camera, a flag is set signaling to the FPGA to prepare to capture the data collected from the photo.
2) Read the middle row of pixels.
   For the duration of the next frame coming into the FPGA fabric, wait for the middle row of pixels.
3) Save the middle row of pixels.
   Upon determining the end of the middle row, set the flag back to its original status, indicating that the FPGA should not save the remainder of the incoming data.
4) Interface with the HPS.
   A mutex or handshake signal may be used to pause the HPS from reading shared memory during capture, ensuring data integrity. Extract one row of pixels when a button is pressed. The HPS can then safely access the captured row for further software-side decoding of the barcode.

FPGA Fabric → Device Driver Interface

We will implement a custom device driver that enables the transfer of RGB color data from the FPGA to the HPS. Specifically, the driver will expose the RGB values of each pixel in the middle row of the captured image frame. We aim to offload as much image processing as possible to software running on the HPS for the following reasons:

1) Performing most of the data processing on software allows for faster iteration. Developing and testing processing algorithms in software allows for significantly faster iteration. Compiling software changes typically takes under a minute, while recompiling FPGA fabric can take 15 minutes or more.
2) Software also allows for greater flexibility. Software provides more adaptability to changing conditions. For example, if ambient lighting changes or we need to fine-tune denoising, thresholding, or filtering parameters, we can update the software without requiring time-consuming hardware recompilation.

This hybrid approach enables us to leverage the FPGA for efficient data acquisition while retaining the flexibility and rapid development cycle of software-based processing.

Device Driver Details

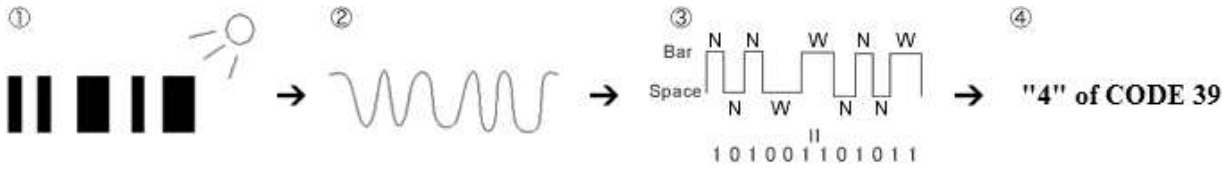
Given the complexity of developing device drivers, and the fact that this project uses the same board as the FPGA board in the class labs, we plan to base much of the infrastructure for our device driver off of the driver we completed in lab 3. Overall, it will have the same core functionality of hardware interface, memory mapping, and kernel development.

Device Driver → Software Interface

As stated above, this part of the project will be fairly simple as it will be based on the lab3 driver to software interface. We will choose the correct ioctl functions and parameters to successfully interact with the driver and implement our desired functionality.

Software Image Processing

The rgb pixels read from the camera will be converted to a number in software since the process will involve multiple processing steps and conversions. The diagram below shows the barcode processing at various stages of the decoding process.



(Barcode processing stages[1])

The steps in the process are as follows:

1. First, the physical barcode layout with light and dark stripes.
2. An image sensor reads the black and white stripes as brightness values and produces an analog signal. This signal does not have clean thresholds for timing and must be classified by pulse length.
3. An amplifier is then used to convert it to digital pulses which have consistent widths (either “narrow” or “wide”)
4. The EAN-13 specification then describes how numbers are encoded into narrow/wide stripe patterns.
    - https://internationalbarcodes.com/ean-13-specifications/
    - https://images-na.ssl-images-amazon.com/images/I/A1DW6hmAlaL.pdf

Components:

1. Determine pixel color based on thresholds.
    a. First convert RGB color space to HSV to determine absolute brightness (value)
    b. Set a minimum and maximum value for white and black respectively.
    c. Middle pixels are defined as background and ignored.
2. Identify the barcode based on guard patterns.
    a. There are standardized “guards” on the left, right, and in the middle
    b. Left guard: 101
    c. Center guard: 01010
    d. Right guard: 101
3. Scale the pattern when the camera is yawed. This requires measuring the left and right guard scale and interpolating between them.

[1] https://www.denso-wave.com/en/adcd/fundamental/barcode/scan/index.html

4. Determine narrow and wide bar widths according to scale linear interpolation.
5. Assign narrow and wide bars across the image.
6. Convert narrow and wide bar pattern to number.

NOTE: More justification for these processing steps is provided in the “Optical Effects” section.

Software → VGA Monitor

To display UPC numbers on the screen, we plan to adapt the memory map used in Lab 2 to perform a similar functionality and display information about the item scanned. We will need to download lab2-img.tar.gz from the class website, unpack it to create the (sparse) 16 gb lab2-16G.img file, and then flash it using dd (slow) or bmaptool (much faster) onto the fpga. When booted, this will configure the fpga (soc_system.rbf), start the kernel (zImage), and load the Device Tree (soc_system.dtb). The barcode number, after being computed in the software, will be converted to characters and drawn into a register or the memory of the FPGA before it is called upon to determine what item to display on the VGA monitor.

```
/* Look for and handle keypresses */
for (;;) {
  libusb_interrupt_transfer(keyboard, endpoint_address,
                            (unsigned char *) &packet, sizeof(packet),
                            &transferred, 0);
  if (transferred == sizeof(packet)) {
    sprintf(keystate, "%02x %02x %02x", packet.modifiers, packet.keycode[0],
            packet.keycode[1]);
    printf("%s\n", keystate);
    fbputs(keystate, 6, 0);
    if (packet.keycode[0] == 0x29) { /* ESC pressed? */
        break;
    }
  }
}
```

Similarly to how this code looks for a keypress before updating its screen, our program will look for the memory allocated to storing the barcode data to be updated before it updates its screen.

Optical Effects

To properly instruct users of the barcode scanning system, we plan to experiment with and improve the limitations of the device as we iterate the image processing parameters. This way, although there may remain significant limitations regarding the flexibility of the device usability, we will create the most usable product possible. We have determined the key limitations that will guide our design process and affect the usability of the scanner.

Distance

How far can the user be from the barcode? This limitation is derived from the resolution of the camera and its ability to distinguish narrow and wide bars. We will decide an acceptable number of pixels for a narrow bar based on testing (likely no less than 2-3 since the pixel borders will never be perfectly aligned with the barcode stripes). The maximum distance will assume that the camera is aligned horizontally to the barcode since that is the worst case (smallest bar width). Once, upon our testing, the camera consistently misidentifies the barcode, we will know we are too far.

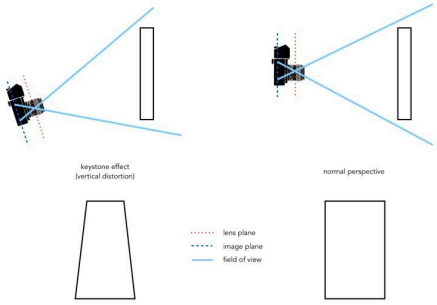
Similarly, we need to determine how close the camera can be to the barcode? This limitation is based on the camera's minimum focus distance and field of view.

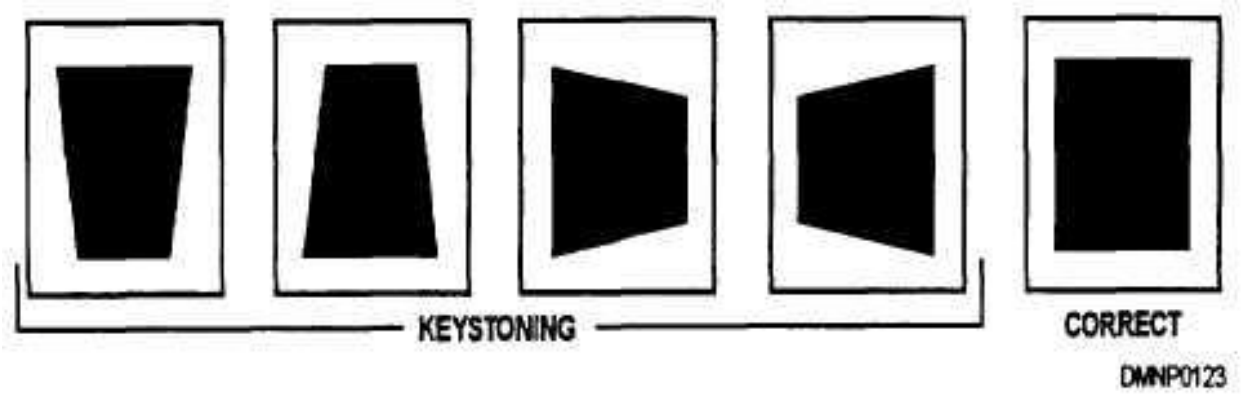
Alignment (Camera Orientation)

There are three angles which define the alignment of the camera to the barcode: pitch, yaw, and roll. For our purposes, pitch and yaw together can be classified as "angle of incidence", representing how aligned the camera sensor plane and barcode plane are. Roll

Angle of Incidence

Pitch and yaw represent the alignment of the camera sensor plane to the plane of the barcode. If the camera is normal to the surface, the angle of incidence is 0°. The more oblique (misaligned) the camera is from the surface, the smaller the projected height and width of the barcode. However, while the visual size of the barcode shrinks as it is projected at a high angle of incidence, the two axes have different effects.

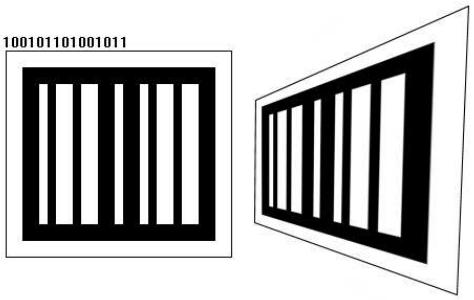




Since barcodes are horizontally symmetric, the pitch of the camera does not alter the processing, since selecting a single pixel row will mitigate keystoning on the horizontal axis. The only limitation imposed by camera pitch is that the center row of pixels remains within the

bounds of the barcode. This equates to the user moving the scanner up and down to align the center row of pixels within the height of the barcode.

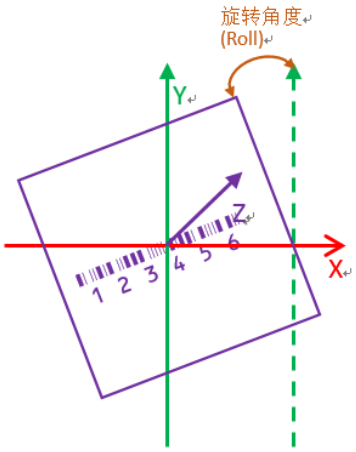
On the other hand, since barcodes encode data along their horizontal axis, the yaw of the camera causes meaningful keystoning of the bars. Keystoning means that the closer side of the barcode will appear larger than the farther side. This equates to a non-uniform stretching of the bar widths. This effect must be addressed in the software algorithm. Our plan is to use the left and right barcode identifiers to calibrate the most and least stretched sides of the barcode, then linearly interpolate between the two scales as we parse narrow and wide bars in the middle.



For angle of incidence tolerance, ±72° has been achieved in commercial scanners[2] in both tilt and skew angle.

Roll

The camera roll is similar to the pitch since it does not require a software algorithm to account for. Since we will be selecting a single row of horizontal pixels from the camera, the user will be responsible for orienting the camera to the correct roll. However, this does not mean that the camera must have 0° of roll. As long as the full width of the barcode is intersected by the X axis of the camera, the data will be read. The amount of roll allowed by the camera is dependent on the height of the barcode. A taller barcode would allow for more roll while still scanning. The one additional consideration with roll is that the horizontal sample of the barcode would stretch as the roll increases. This is already accounted for in the algorithm.



A 360° roll tolerance has been achieved in commercial scanners[3].

Scene Brightness

The camera will have auto exposure, however scenes which are too dim or bright will compress the dynamic range of the image. To solve this we will experiment with using an LED above the camera to enable a constant brightness in more lighting conditions.

[2] https://www.lmppos.com/product/2D-Wireless-Barcode-Scanner.html
[3] https://www.lmppos.com/product/2D-Wireless-Barcode-Scanner.html

Image Noise

The image will have natural noise due to thermal and electrical effects on the image sensor, especially in dim conditions. Raising the brightness with an LED may solve this problem if the noise is strong enough to affect the white/black thresholds for bar colors.

Background

The barcode will not always be on a pure white background. The patterns to the left and right of the barcode may resemble the white/black patterns of the bars. To allow the system to recognize the barcode itself, the EAN-13 specification implements standard “guards” on the left and right sides of the encoded barcode data. These consistent patterns allow the processing system to identify the barcode location.