

Hardware? More like Heart-Ware

Sydney Saenz

Department of Electrical Engineering
and Computer Science
Massachusetts Institute of Technology
Email: saenz728@mit.edu

Jonatan Hernandez-Mijangos

Department of Electrical Engineering
and Computer Science
Massachusetts Institute of Technology
Email: aevum@mit.edu

Abstract—This project implements a real-time, FPGA-based system for estimating human heart rate directly from a video stream. The approach relies on photoplethysmography-by-camera, where subtle color fluctuations in skin caused by periodic blood-volume changes are amplified and isolated through signal-processing techniques. A real-time signal processing pipeline continuously estimates the dominant frequency within a band of 50-200 beats per minute (BPM), providing an online heart-rate estimate with low hardware overhead. The complete design demonstrates how classical signal processing, lightweight machine learning, and FPGA computation can be combined to recover vital-sign information from video in real time.

I. INTRODUCTION

Non-contact physiological sensing is a rapidly growing area of research, driven by the expanding availability of high-frame-rate cameras and low-power compute platforms. Subtle variations in skin color provide enough information to estimate a user's heart rate without requiring a dedicated sensor. Building on techniques introduced by Poh *et al.* [1] and Pursche *et al.* [2], we implement this system in FPGA hardware with a Pulse Oximeter for comparison and verification. The system ingests a video stream, processes each frame through a pipelined architecture, and outputs a continuous estimate of the user's heart rate. A fixed region of interest is displayed on-screen; for each frame, we filter the color signals and frequency analysis is performed using a Fast Fourier Transform (FFT). Another pipeline from a pulse-oximeter heart rate sensor will result in a baseline measurement on display. Overall, this system demonstrates that meaningful information for non-contact heart-rate measurement can be recovered using lightweight computation and modest-resolution video.

II. SYSTEM OVERVIEW

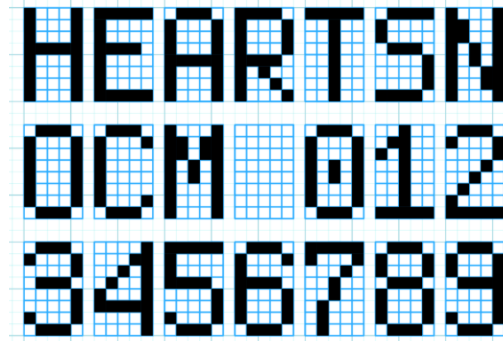
A. Memory Pipelining and Video Feed



The first section of the video feed is the camera feed display.

The camera data is from an OV5640 camera. The camera feed displayed is only a section of the total frame data. For signal processing purposes, only a portion is needed; enough to show and capture the human's face. The camera data is stored in DDR3 Memory, and accessed through AXI Streaming.

The other section is the results display. This section shows the results from the heart rate sensor and the signal processing from the camera data. The letters and numbers use a simple 8x5 bits encoding, where each bit takes up 4x4 pixels for the letters and 8x8 pixels for the numbers.



The modules responsible for the different components of the video feed are implemented. Currently, the numbers displayed are test numbers since the signal processing and sensor are not yet complete. These example heart rates tell us that we can update the heart rate and maintain a clear camera feed.

B. Reading Data from a Pulse Oximeter

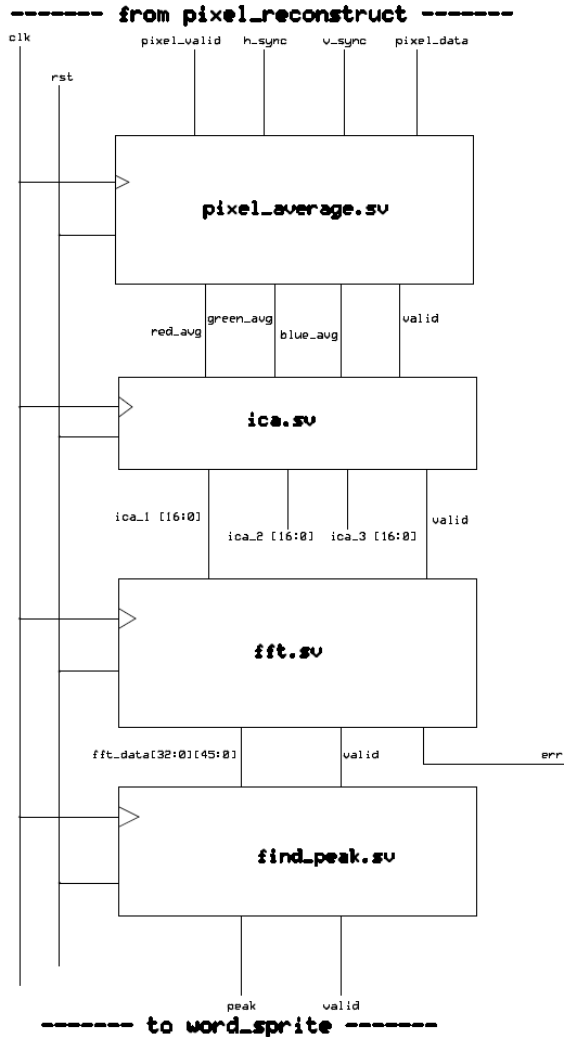


The sensor used in the project is the MAX30101 from SparkFun. This version comes with a MAX32664 biometric hub; a small and fast Cortex M4 processor responsible for handling calculations. The sensor communicates over I2C, and has specific addresses for configuration and reading.

the first two, it will move to the READING state after reading in the data. For the latter, it will read in the high byte of the heart rate, then move on to READ_HR_LO.

- **READ_HR_LO:** Sets appropriate commands for the i2c master to read in the data bytes. Reads the low byte of the heart rate, then moves on to read the rest of the bytes. Sets appropriate commands for the i2c master to read in the data bytes.
- **READ_REST:** Sets appropriate commands for the i2c master to read in the data bytes. Reads the remainder 4 bytes or checks for the i2c master to indicate the last byte, then goes back to READING and restarts the reading states variable.
- **DATA_DELAY:** Delay system by 5s. This is a large delay between enabling and reading to allow the FIFO to populate with samples.

III. SIGNAL PROCESSING OVERVIEW



The signal processing pipeline is designed to extract physiological information from a real-time video stream operating at 25 fps. Although the FPGA camera ultimately reads out data at 25 fps, our simulation setup over-samples at 60 fps; this does not meaningfully affect the downstream processing in Python,

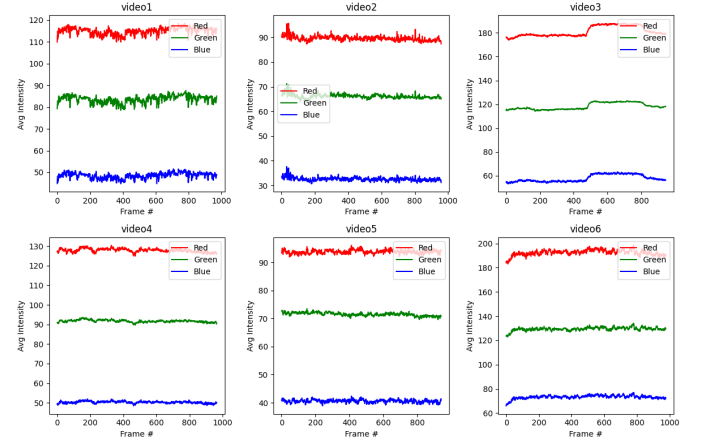
nor the final result. The entire method, initially prototyped in Python using six 15-second videos recorded directly from the OV5640 camera, was migrated into SystemVerilog with a combination of hand-written modules and Xilinx IP (Intellectual Property). Key HDL modules of the signal processing pipeline implement color channel averaging over frames, Independent Component Analysis, Fast Fourier Transform (FFT), and peak detection. All modules were expressed first in Python (via Jupyter Notebook) before being implemented in HDL.

The signal chain consists of three major stages: per-frame color processing, a linear Independent Component Analysis (ICA) transform using a learned unmixing matrix, and a frequency-domain analysis stage driven by a real-time FFT. The overall latency is dominated not by computation but rather by the need to buffer a sufficient number of samples for the FFT.

A. Channel Selection and Preprocessing

Each incoming video frame arrives as an RGB565 pixel array provided by the Video Signal Generator. A fixed region of interest (ROI) is defined in advance and used for all frames. For a certain region inside the ROI, the system extracts the red, green, and blue components and accumulates the frame-wise sum for each channel. Because the sample pools are selected to be a power of two, the final averaging step is implemented as a simple right shift, eliminating the need for a divider module. Sparse-sampling or only sampling of the mid-face has provided more accurate results. This operation is essentially a streaming version of the `center_of_mass` computation seen in Labs 05 and 06.

The outputs of this stage are three components for each frame, represented as $\{avg_R, avg_G, avg_B\}$. These signals form the raw temporal traces from which the later ICA and FFT stages can extract the heart rate.



B. Independent Component Analysis

The averaged RGB signals are a linear mixture of multiple sources of temporal variation: lighting fluctuations, camera gain behavior, spatial motion, skin color changes associated with blood-volume modulation, et cetera. These sources are linearly mixed in the RGB space, so the next stage applies a linear unmixing transformation obtained from ICA.

Before FPGA implementation, the ICA unmixing matrix was trained offline using the collected set of six training videos, of two different subjects each in 3 different lighting. In Python, each video is reduced to its frame-wise RGB averages and all videos are concatenated into a single dataset. The FastICA algorithm is then applied to this combined dataset to estimate a 3×3 unmixing matrix W . This matrix maximizes the statistical independence of the output components and empirically isolates one component dominated by periodic cardiac activity. Because ICA is performed only during the training phase, the FPGA system merely stores the resulting matrix W and applies the linear transformation

$$s(t) = Wx(t)$$

in real time, where $x(t)$ is the current mean-centered RGB vector. This operation requires only nine multiplications and three additions, making it trivial to implement in hardware. Each resulting independent component $s_i(t)$ is represented as a signed 16-bit value, and the first component is typically selected because it contains the most reliable physiological information.

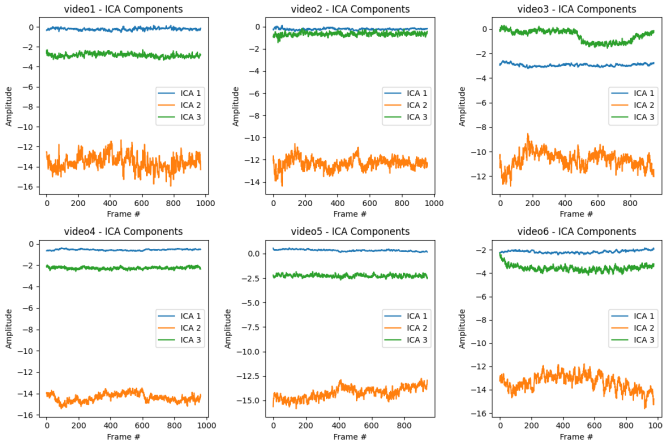
The originally trained unmixing matrix

$$W = \begin{bmatrix} -0.12029778 & 0.14201219 & 0.03559064 \\ 0.54497371 & -0.99840553 & 0.14995908 \\ 0.02248132 & 0.09794917 & -0.28072700 \end{bmatrix}$$

was uniformly scaled and rounded to integer values to simplify computation in HDL, without affecting the recovered signal. The resulting fixed-point matrix

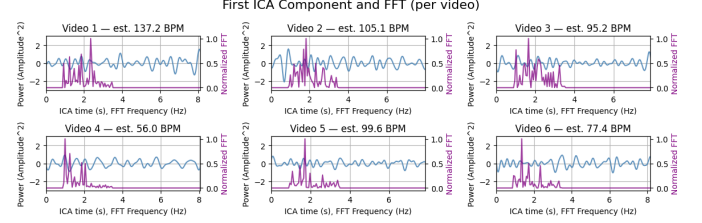
$$W_{\text{int}} = \begin{bmatrix} -120 & 142 & 35 \\ 544 & -998 & 149 \\ 22 & 97 & -280 \end{bmatrix}$$

preserves the structure and relative weighting of the original matrix, ensuring that the subsequent FFT analysis remains unchanged.

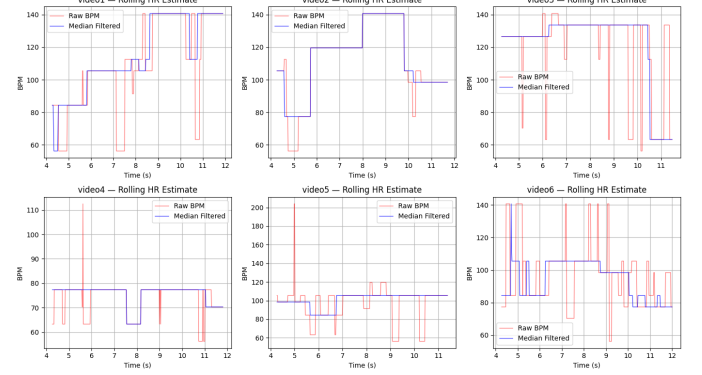


1) *Frequency-Domain Analysis*: After ICA, the signal must be converted into the frequency domain to estimate the user's heart rate. A Python implementation performs

two complementary analyses. One process uses the full 15-second recording to compute a single high-resolution FFT:



The other process mimics the FPGA's behavior by applying an FFT to a rolling buffer of fixed length:



The latter approach is more realistic, as the FPGA maintains a rolling buffer for continuous output.

On the FPGA, ICA outputs are written into a rolling buffer length $N = 512$. Once the buffer is full, the samples are fed directly into the Xilinx XFFT IP core operating in a pipelined streaming mode. The Xilinx FFT IP core was configured using the Vivado Design Suite GUI and tested under Vicoco. Then, we created a wrapper-module to process the raw output of the IP core, which releases data in 32-bit $\{IM_x, RE_x\}$. Using 512 points at 25 fps gives a frequency resolution of approximately ≈ 0.23 Hz, corresponding to roughly a 7 bpm resolution. After the XFFT produces its complex output, we compute the power spectrum (magnitude-squared), and the module only exports the 0.8-3 Hz (roughly 50-200 BPM) band to another module for locating the dominant peak, which reduces space while performing the bandpass function. This frequency corresponds to the heart rate, which can then be displayed on screen in beats per minute, using a previously-computed lookup table.

IV. CURRENT RESULTS

A. Signal Processing in Simulation

The Python simulation on the same OV5640 camera data that will be used in hardware validates the feasibility of the architecture and serves as a direct reference for the hand-coded filters and Xilinx IP integrations used in the final FPGA design. Reasonably consistent results can be seen in the signal processing output (see: *Frequency-Domain Analysis*) and the trained ICA matrix has proven robust against different subjects, skin colors, and lighting. [3]

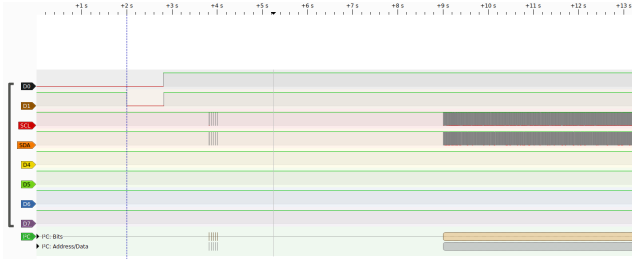
B. Signal Processing on FPGA

The Xilinx FFT IP Core has proven to be an effective and fast processor for the input data; it is easily able to compute and release a 512-point Fast Fourier Transform well within the rolling-buffer time constraints (the buffer updates once per frame, or at a rate of 25Hz). This lends itself to the possibility of performing a 1024-point FFT every frame, which would in theory provide much more precise heart-rate estimation. However, the downside of a fast update is that the on-screen view is difficult to read as a result of many continuous updates in rapid succession.

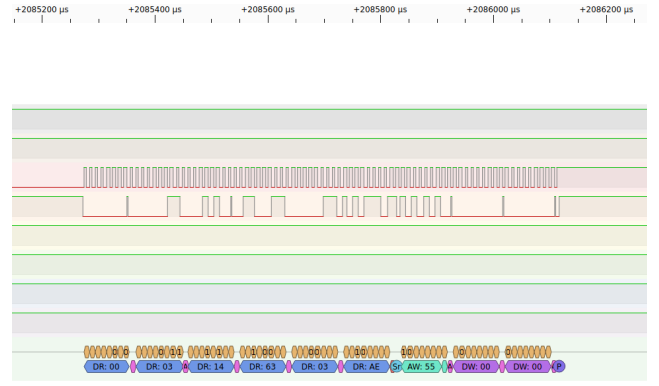
C. Number Display

In order to display the heart rates in decimal format, the heart rates must be split into hundreds, tens and ones. An approximation of $1/100$ and $1/10$ was done to avoid dividing by 100 and 10. To divide by 100, we first multiply by 656, then shift right by 16. This makes sense because $\frac{656}{2^{16}} \approx 0.010 \approx \frac{1}{100}$. To divide by 10, we use a similar logic. First, multiply by 205, then shift right by 11. $\frac{205}{2^{11}} \approx 0.10 \approx \frac{1}{10}$. Once hundreds, tens, and ones are separated, we can display them using the encoding mentioned earlier.

D. Heart Rate Sensor



I probed the pins between the host and the MAX32664 to capture the data being transmitted. During the first two seconds, you can see the initial rstn and mfio sequence. Then, right after, you can see the ENABLE state communication over the i2c lines. After 5 seconds, the i2c lines begin to communicate the heart data readings. Some initial issues we encountered were the timings of the delays. At first, we had the minimum constraints for the delays, but the device wouldn't go into application mode or wouldn't have data ready to read. So, we increased those delay times until the system was stable.



Here is the sample data read from FIFO. The first byte is the status byte, then the next two is the heart rate x10: 0x0314 which is 788. So the measured heart rate is 78.8.

After getting some heart readings displayed on the screen, we compared the heart rate sensor results to that of a Garmin watch. Surprisingly, the heart rate sensor was 1-2 bpm away from Garmin's measured heart rate. Even more surprising, the heart rate sensor was updating before the Garmin watch would. We tested this by raising the patient's heart rate through short intense exercise. The display would update before the Garmin watch would; about a second faster. This meets our requirements and stretch goals of being within 3% accuracy and faster than the Garmin watch.

V. CONCLUSION

The complete processing chain for this project forms a compact, relatively real-time physiological monitoring system built directly into hardware. Previous work [2] has only implemented these strategies into software, often requiring long video samples or high computational overhead. The offline ICA training allows the hardware to apply a fixed 3×3 separation matrix without iterative optimization, while the FFT stage enables continuous extraction of heart-rate information from the recovered independent component. A pulse-oximeter serves as a comparison and benchmark for signal estimation, and a Garmin watch (which contains an official accuracy statement) can be used for further comparison and quality assessment.

VI. CODE REPOSITORY

<https://github.mit.edu/6205F25/fa25-6205-team41>

VII. INDIVIDUAL CONTRIBUTIONS

A. Code

Jonatan worked on every systemverilog file related to the heart rate sensor and video display feed as well as the pixel averaging module and its testbench.

Sydney worked on the digital signal processing pipeline, including instantiating the Xilinx IP, training the ICA matrix, and implementing filtering and peak detection. Also implemented a full Python simulation to demonstrate feasibility and display results of the process.

B. Report

Jonatan worked on the heart rate sensor subsections in sections II and IV. Jonatan also worked on the video feed subsections in sections II and IV. Sydney wrote the Signal Processing Overview, as well as parts of Current Results, Conclusion, and Introduction.

REFERENCES

- [1] M.-Z. Poh, D. J. McDuff, and R. W. Picard, "Non-contact, automated cardiac pulse measurements using video imaging and blind source separation," *Opt. Express*, vol. 18, no. 10, pp. 10 762–10 774, May 2010. [Online]. Available: <https://opg.optica.org/oe/abstract.cfm?URI=oe-18-10-10762>
- [2] Y.-P. Yu, P. Raveendran, and C.-L. Lim, "Dynamic heart rate measurements from video sequences," *Biomedical Optics Express*, vol. 6, pp. 2466–2480, 06 2015.
- [3] S. Saenz and J. Hernandez-Mijangos, "6.2050 Final Project," Nov. 2025. [Online]. Available: <https://github.com/sydsaenz/2050FP>