**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

Logo, company name

Description automatically generated

**BÁO CÁO ĐỒ ÁN 1**

**HIỆN THỰC SoC CAMERA**

|  |  |
| --- | --- |
| **GIẢNG VIÊN HƯỚNG DẪN:** | **TRƯƠNG VĂN CƯƠNG** |
| **SINH VIÊN THỰC HIỆN:** | **NGUYỄN TRẦN TRÍ THỨC – 19521007**  **CAO CHÍ NHÂN - 19520794** |

**Ho Chi Minh City, /2022**

MỤC LỤC

[Chapter 1. GIỚI THIỆU TỔNG QUAN 5](#_Toc107782817)

[Chapter 2. TÌM HIỂU LÝ THUYẾT 5](#_Toc107782818)

[2.1. Xillybus 5](#_Toc107782819)

[2.1.1. Xillybus là gì 5](#_Toc107782820)

[2.1.2. Luồng dữ liệu 6](#_Toc107782821)

[2.1.3. Clocking và Datawidth 6](#_Toc107782822)

[2.1.4. Giao tiếp thông qua FIFO 7](#_Toc107782823)

[2.1.5. Hành vi của tín hiệu “FULL” và “EMPTY” 7](#_Toc107782824)

[2.2. Board Zybo Z7 7](#_Toc107782825)

[2.3. Các giao thức mạng 9](#_Toc107782826)

[2.3.1. RTSP[4] 9](#_Toc107782827)

[2.3.2. SSH[5] 10](#_Toc107782828)

[2.4. GStreamer[6] 11](#_Toc107782829)

[Chapter 3. THIẾT KẾ VÀ HIỆN THỰC 12](#_Toc107782830)

[3.1. Xây dựng thư viện FFMPEG và OpenCV 12](#_Toc107782831)

[3.1.1. Cài đặt các package cần thiết 12](#_Toc107782832)

[3.1.2. Xây dựng thư viện FFMPEG 12](#_Toc107782833)

[3.1.3. Xây dựng thư viện libfaac-dev 13](#_Toc107782834)

[3.1.4. Xây dựng thư viện OpenCV 13](#_Toc107782835)

[3.2. Cài đặt GStreamer và RTSP 15](#_Toc107782836)

[3.2.1. Cài đặt các package cần thiết 15](#_Toc107782837)

[3.2.2. Xây dựng thư viện GStreamer và RTSP 15](#_Toc107782838)

[Chapter 4. ĐÁNH GIÁ KẾT QUẢ 15](#_Toc107782839)

[4.1. Cài đặt hệ điều hành xillinux-2.0 lên board và nghiên cứu cách giao tiếp giữa OS và phần FPGA 15](#_Toc107782840)

[4.2. Kiểm tra các thư viện đã cài đặt 17](#_Toc107782841)

[4.2.1. OpenCV 17](#_Toc107782842)

[4.2.2. Gstreamer + RTSP 19](#_Toc107782843)

[Chapter 5. HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN 20](#_Toc107782844)

[5.1. Hạn chế 20](#_Toc107782845)

[5.2. Hướng phát triển 20](#_Toc107782846)

[TÀI LIỆU THAM KHẢO 21](#_Toc107782847)

**DANH MỤC HÌNH ẢNH**

[Hình 2.1 Sơ đồ luồng dữ liệu 6](#_Toc107782848)

[Hình 2.2 Board Zybo Z7 8](#_Toc107782849)

[Hình 2.3 Kiến trúc của dòng Zynq 7000 8](#_Toc107782850)

[Hình 2.4 Giao diện PuTTY 10](#_Toc107782851)

[Hình 2.5 Cấu trúc của GStreamer 11](#_Toc107782852)

[Hình 4.1 Board thực tế 15](#_Toc107782853)

[Hình 4.2 Giao diện trên board Zybo Z7 16](#_Toc107782854)

[Hình 4.3 Giao tiếp giữa máy tính cá nhân và board 16](#_Toc107782855)

[Hình 4.4 Đọc giá trị FIFO và lưu vào read.txt 16](#_Toc107782856)

[Hình 4.5 Ghi giá trị vào FIFO 17](#_Toc107782857)

[Hình 4.6 Đọc giá trị ra từ file read.txt 17](#_Toc107782858)

[Hình 4.7 Ví dụ về ghi giá trị của FIFO 17](#_Toc107782859)

[Hình 4.8 Kết quả đọc giá trị của FIFO 17](#_Toc107782860)

[Hình 4.9 OpenCV 2.4.9 18](#_Toc107782861)

[Hình 4.10 OpenCV Camera 19](#_Toc107782862)

[Hình 4.11 GStreamer 1.8.0 19](#_Toc107782863)

[Hình 4.12 Video Streaming từ luồng 20](#_Toc107782864)

**DANH MỤC BẢNG**

[Bảng 2.1 Đặc điểm của Zybo Z7 9](#_Toc107782865)

[Bảng 4.1 Source code Capture camera sử dụng OpenCV 18](#_Toc107782866)

# GIỚI THIỆU TỔNG QUAN

Với sự bùng nổ của cuộc Cách mạng Công nghiệp 4.0, sự ra đời của các thiết bị thông minh ngày càng giúp cho đời sống của con người trở nên tiện lợi và dễ dàng hơn. Trong đó có camera, một thiết bị không thể thiếu trong việc phục vụ vấn đề an ninh an toàn. Trên thị trường hiện nay không khó bắt gặp nhiều loại camera với nhiều mẫu mã và chức năng khác nhau, kèm theo đó là giá thành cũng đi kèm với chất lượng sản phẩm.

Với mong muốn sử dụng các thiết bị thông minh - trong đó có camera, có chức năng ổn định, giá cả hợp lý. Nhóm đề xuất ý tưởng thiết kế một IP Camera có chức năng phát luông video trực tiếp qua mạng với giao thức RTSP.

Trong giới hạn đề tài, nhóm ưu tiên sử dụng các board tích hợp Xilinx Chip để nghiên cứu và hiện thực đề tài. Đơn cử như dòng board Zypo Z7 (Zynq-7000 ARM/FPGA SoC Development Board)**.**

# TÌM HIỂU LÝ THUYẾT

## Xillybus

### Xillybus là gì

Là một giải pháp chìa khóa trao tay từ đầu đến cuối đơn giản, trực quan, hiệu quả dựa trên Direct Memory Access (DMA) – hay Truy cập dữ liệu trực tiếp, để truyền dữ liệu giữa FPGA và một máy chủ chạy Linux hoặc Windows. Trong đó, DMA là cơ chế của hệ thống máy tính cho phép một thành phần phần cứng truy cập đến bộ nhớ dữ liệu chính (như RAM) một cách độc lập với CPU[1].

Xillybus hỗ trợ các loại FPGA của Xilinx như Virtex-5T, Spartan-6T, Virtex-6T, các dòng Series – 7, tất cả dòng Ultrascale và Ultrascale+, Zynq-7000/Ultrascale+; của Intel FPGA như Cyclone, Stratix,… Hiện nay các bản phân phối Linux đã bao gồm có Xillybus Driver, đối với Windows, Xillybus hỗ trợ các phiên bản Windows 7, 8 và 10 (32 và 64 bit).

### Luồng dữ liệu

Hình 2. Sơ đồ luồng dữ liệu

Hình bên trên mô tả một sơ đồ khối đơn giản cho thấy kết nối của một luồng dữ liệu theo mỗi hướng. Xillybus IP Core (Lõi IP Xillybus) giao tiếp dữ liệu với người dùng thông qua các FIFO tiêu chuẩn. Các nhà thiết kế FPGA có thể tự do quyết định độ sâu của FIFO và việc giao tiếp của nó với ứng dụng logic.[2]

Việc ghi dữ liệu vào FIFO bên dưới làm cho lõi IP Xillybus nhận ra dữ liệu đã sẵn có để truyền ở đầu kia của FIFO. Sau đó, Xillybus đọc dữ liệu từ FIFO và gửi đến máy chủ lưu trữ, làm cho nó có thể đọc được bởi phần mềm không gian người dùng. Lõi Xillybus IP thực hiện luồng dữ liệu sử dụng bus AXI, tạo ra các yêu cầu DMA trên bus của lõi bộ xử lý.

### Clocking và Datawidth

Tất cả các tín hiệu từ và đến lõi Xillybus IP phải được xử lý xung nhịp (clocking) bằng bus clk do chính lõi đó cung cấp. Clock này được tạo PCIe hoặc lõi bộ xử lý và có tần số tùy thuộc vào các nền tảng khác nhau. Trong hầu hết các trường hợp đều có khả năng thay đổi tần số clock với một danh sách lựa chọn hạn chế, bằng cách kết hợp PCIe hoặc lõi bộ xử lý tạo ra nó.

Đối với độ rộng dữ liệu (datawidth), với lõi Xillybus IP, mỗi FIFO hoặc giao diện bộ nhớ hoạt động với dữ liệu có độ rộng: 8 bits, 16 bits, 32 bits. Với XillyUSB, có thể hỗ trợ những giao diện có độ rộng dữ liệu lớn hơn. Dữ liệu có độ rộng càng lớn cho phép hiệu suất băng thông càng cao.[3]

### Giao tiếp thông qua FIFO

Tùy thuộc vào hướng dữ liệu, mà các Tín hiệu “empty” hoặc “full” của FIFO được kết nối vào lõi IP Xillybus và được nhận ra để xác định sẽ xảy ra bùng nổ một loạt dữ liệu (data burst) hay không. Một khi xảy ra, các tín hiệu này được nhận ra để đảm bảo lõi IP Xillybus không có gắng đọc từ một FIFO trống hoặc ghi vào một FIFO đã đầy.

FIFO với độ sâu 2kByte là lựa chọn chính xác cho một luồng không đồng bộ, ngay cả đối với tốc độ dữ liệu cao vì lý do lõi Xillybus không có khả năng bỏ qua FIFO đủ lâu để cho phép 2kByte FIFO overflow hoặc underflow.[3]

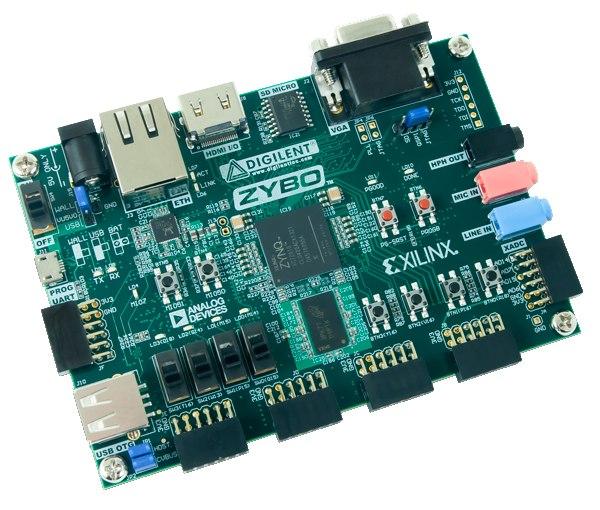
### Hành vi của tín hiệu “FULL” và “EMPTY”

Trong một FIFO hoạt động bình thường, tín hiệu “empty” có thể lên cao chỉ trong 1 chu kì clock sau khi read enable được xác nhận. Tín hiệu “full” có thể lên cao trong 1 chu kì clock sau khi write enable được xác nhận. Hai tín hiệu có thể xuống thấp tại bất cứ lúc nào.

Xillybus IP Core dựa trên hành vi này: Khi FIFO báo hiệu cho lõi rằng nó đã sẵn sàng cho việc truyền dữ liệu (tùy điều kiện áp dụng mà có thể là “empty” hoặc “full”), một máy trạng thái trong lõi có thể bắt đầu một chuỗi sự kiện mà sẽ dẫn đến việc chuyển ít nhất một phần tử dữ liệu (data element), bất kể tín hiệu của FIFO tại thời điểm đó là gì. Tuy việc này về tính toàn vẹn là vô hại, nhưng có thể dẫn đến dữ liệu không mong muốn và không thể dự đoán trước được dữ liệu trong luồng.[3]

## Board Zybo Z7

Board Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board, do Xilinx sản xuất, thuộc họ Zynq-7000. Tích hợp Vi xử lý lõi kép ARM Cortex-A9. Zybo Z7 xoay quanh Zynq với bộ đa phương tiện và việc kết nối các thiết bị ngoại vi phong phú làm cho Zybo Z7 trở nên phổ biến.



Hình 2. Board Zybo Z7

Hình 2. Kiến trúc của dòng Zynq 7000

Bảng 2. Đặc điểm của Zybo Z7

|  |  |
| --- | --- |
| **Đặc điểm** | **Zybo Z7** |
| **Bộ xử lý Zynq** | Lõi kép Cortex-A9  Bộ điều khiển bộ nhớ DDR3L(8 DMA channel và 4 cổng AXI3 Slave) |
| **Tần số** | Tối đa 667MHz |
| **Bộ nhớ** | 1 Gb DDR3L với 32bit bus @1066MHz |
| **Nguồn cấp** | Được cấp nguồn từ USB hoặc các nguồn 5V bên ngoài |
| **Các cổng giao tiếp** | Gigabit Ethernet, USB-JTAG, USB-UART, USB 2.0 |

## Các giao thức mạng

### RTSP[4]

RTSP – Real Time Streaming Protocol, là một giao thức truyền tin thời gian thực ở tầng ứng dụng được thiết kế để sử dụng trong các hệ thống giải trí và truyền thông để điều khiển máy chủ chứa các dữ liệu truyền tin đa phương tiện (streaming media). Giao thức này được sử dụng để thiết lập và điều khiển các phiên truyền thông giữa các trạm cuối.

Việc truyền tải dữ liệu trực tuyến không phải là một nhiệm vụ của RTSP. Hầu hết các máy chủ RTSP sử dụng giao thức truyền tải thời gian thực (RTP) kết hợp với giao thức điều khiển thời gian thực (RTCP) để phân phối luồng truyền thông.

RTSP mang một số đặc điểm nổi bật: Khả năng mở rộng - các phương pháp và các thông số mới có thể dễ dàng thêm vào RTSP; dễ dàng phân tích - RTSP có thể được phân tích theo bộ phân tích cú pháp HTTP hoặc MIME; an toàn - RTSP tái sử dụng những cơ chế bảo mật web. Tất cả các cơ chế chứng thực HTTP như chứng thực cơ bản và chứng thực băm được trực tiếp áp dụng. Ngoài ra còn có khả năng đa máy chủ, độc lập với tầng vận chuyển,…

### SSH[5]

SSH (Secure Shell), là một giao thức mạng mật mã để vận hành các dịch vụ mạng một cách an toàn. Các ứng dụng SSH dựa trên kiến trúc client – server, kết nối một SSH client với một SSH server. SSH hoạt động như một bộ giao thức phân lớp bao gồm ba thành phần phân cấp chính: lớp transport cung cấp tính xác thực, tính toàn vẹn và tính bảo mật server; giao thức xác thực người dùng xác nhận người dùng đến server; và giao thức kết nối ghép các kênh được mã hóa thành nhiều kênh giao tiếp logic.

Sử dụng phần mềm PuTTY để giao tiếp với board thông qua Serial SSH

Graphical user interface

Description automatically generated

Hình 2. Giao diện PuTTY

Các bước thực hiện bao gồm:

**Bước 1:** Lựa chọn giao thức kết nối (Connection type) là Serial.

**Bước 2:** Chọn port kết nối (Serial line) tương ứng với cổng kết nối ngoài.

**Bước 3:** Chọn tốc độ (Speed) - Baudrate phù hợp.

**Bước 4:** Chọn Open để kết nối.

## GStreamer[6]

GStreamer là một framework cho phép người dùng có thể tạo ra các ứng dụng đa phương tiện, có thể làm việc trên hầu hết các hệ điều hành như Linux, Android, Windows, Max OS X, iOS,.. Ngoài ra còn có thể chạy trên các kiến trúc phần cứng như x86, ARM, MIPS, SPARC và PowerPC (32 và 64 bit).

Đặc điểm của GStreamer bao gồm cấu trúc pipeline; dựa trên thư viện Glib 2.0 nhắm đến việc thiết kế hướng đối tượng; pipeline đa luồng; dung lượng chiếm dụng thấp; hệ thống debug hoàn chỉnh,…

Diagram

Description automatically generated

Hình 2. Cấu trúc của GStreamer

# THIẾT KẾ VÀ HIỆN THỰC

## Xây dựng thư viện FFMPEG và OpenCV

### Cài đặt các package cần thiết

$ apt-get update

$ apt-get upgrade

$ apt-get build-essential

$ apt-get pkg-config

$ apt-get install build-essential cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev

$ apt-get install python3.5-dev python3-numpy libtbb2 libtbb-dev

$ apt-get install libjpeg-dev libpng-dev libtiff5-dev libjasper-dev libdc1394-22-dev libeigen3-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev sphinx-common libtbb-dev yasm libopencore-amrnb-dev libopencore-amrwb-dev libopenexr-dev libgstreamer-plugins-base1.0-dev libavutil-dev libavfilter-dev libavresample-dev

$ apt-get install cmake-curses-gui

### Xây dựng thư viện FFMPEG

Tải source code thư viện FFMPEG và build trên board với dòng lệnh sau:

$ ./configure --enable-shared --disable-static --arch=armv71 --prefix=<ffmpeg local path>

$ make

$ make install

Phân biệt sử dụng giữa static lib và shared lib (dynamic lib):

Static lib (các file đuôi .a): chương trình biên dịch với static lib trên máy A, copy chương trình đó qua máy B chạy. Chạy chương trình bình thường.

Shared lib (các file đuôi .so): chương trình biên dịch với shared lib trên máy A, copy chương trình đó qua máy B chạy. Chương trình không chạy được do thiếu shared lib. Do đó phải copy cả các shared lib đã dùng khi biên dịch trên máy A qua máy B

Trong trường hợp này, ta đang build trực tiếp trên board nên sẽ chọn Shared lib để tiết kiệm thời gian.

### Xây dựng thư viện libfaac-dev

$ wget -O fdk-aac.tar.gz https://github.com/mstorsjo/fdk-aac/tarball/master

$ tar xzvf fdk-aac.tar.gz

$ cd mstorsjo-fdk-aac\*

$ autoreconf -fiv

$ ./configure --prefix="$HOME" --disable-shared

$ make

$ make install

### Xây dựng thư viện OpenCV

Thêm đường dẫn thư viện và file header vào môi trường, sử dụng các lệnh:

$ export LD\_LIBRARY\_PATH=<ffmpeg local path>/lib:${LD\_LIBRARY\_PATH}

$ export C\_INCLUDE\_PATH=<ffmpeg local path>/include:${C\_INCLUDE\_PATH}

$ export CPLUS\_INCLUDE\_PATH=<ffmpeg local path>/include:${CPLUS\_INCLUDE\_PATH}

$ export PKG\_CONFIG\_PATH=<ffmpeg local path>/lib/pkgconfig:${PKG\_CONFIG\_PATH}

Sau khi giải nén OpenCV, tiến hành tạo 1 folder có tên “build”, 1 file text có tên “toolchain.make” có nội dung:

set( CMAKE\_SYSTEM\_NAME Linux )

set( CMAKE\_SYSTEM\_PROCESSOR arm )

set( CMAKE\_C\_COMPILER arm-linux-gnueabihf-gcc )

set( CMAKE\_CXX\_COMPILER arm-linux-gnueabihf-g++ )

set( CMAKE\_INSTALL\_PREFIX /usr/local )

set( CMAKE\_FIND\_ROOT\_PATH /usr/include )

Tiếp theo tạo folder tên “lib” ở trong folder “build” bên trên, chép tất cả các file đuôi “.so” trong folder lib của folder cài đặt ffmpeg (<ffmpeg local path>/lib) vào đó. Nếu không thì khi build OpenCV sẽ không nhận thư viện FFMPEG.

Chuyển đường dẫn đến folder “build”, sau đó chạy lệnh:

$ cmake –D CMAKE\_TOOLCHAIN\_FILE=toolchain.make –D BUILD\_opencv\_nonfree=OFF ..

Dẫn thư viện từ hệ thống vào thư mục build:

$ mkdir/usr/include/sys

$ cp \

/usr/local/include/opencv2/videoio/videoio\_c.h/ \

/usr/include/sys/

$ sudo ln -s /usr/include/sys/videoio\_c.h/ \

usr/include/sys/videoio.h

Để kiểm tra các package mà máy đã cài, nếu đã cài đầy đủ, chạy lệnh

$ ccmake .

Tiến hành cài đặt một số tùy chọn, bao gồm: WITH\_GSTREAMER, WITH\_JPEG, WITH\_PNG, WITH\_V4L. Sau khi tùy chỉnh xong, nhấn ‘c’ và sau đó nhấn ‘g’. Sau khi nhấn ‘c’ hoặc ‘g’ nếu không có bất kì warning nào, bỏ qua bằng cách nhấn ‘e’.

Mở file “CmakeCache.txt” trong folder “build” để kiểm tra config, kiểm tra WITH\_FFMPEG đã ON hay chưa.

Chạy lần lượt lệnh:

$ make

$ make install

## Cài đặt GStreamer và RTSP

### Cài đặt các package cần thiết

$ sudo apt-get install bison

$ sudo apt-get install flex

$ apt-get install gstreamer1.0-tools gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly

### Xây dựng thư viện GStreamer và RTSP

Bước 1: Tải lần lượt các source code gstreamer, gst-plugins-base, gst-plugins-good, gst-rtsp-server

Bước 2: Giải nén các file

Bước 3: Truy cập vào từng thư mục đã giải nén và cài đặt với câu lệnh sau

$ ./configure

$ make

$ make install

# ĐÁNH GIÁ KẾT QUẢ

## Cài đặt hệ điều hành xillinux-2.0 lên board và nghiên cứu cách giao tiếp giữa OS và phần FPGA

* Hệ điều hành xillinx-2.0: Hệ điều hành chạy ổn định trên board và có khả năng xuất hình ảnh qua cổng VGA hoặc HDMI.



Hình 4. Board thực tế

A picture containing text, monitor, electronics, indoor

Description automatically generated

Hình 4. Giao diện trên board Zybo Z7

Đã giao tiếp được giữa máy tính cá nhân và board thông qua giao thức SSH sử dụng cổng kết nối Serial.

A screenshot of a computer

Description automatically generated

Hình 4. Giao tiếp giữa máy tính cá nhân và board

* Giao tiếp giữa OS và FPGA:

Loopback dữ liệu trên cùng 1 FIFO:



Hình 4. Đọc giá trị FIFO và lưu vào read.txt

Text

Description automatically generated with low confidence

Hình 4. Ghi giá trị vào FIFO

Text

Description automatically generated

Hình 4. Đọc giá trị ra từ file read.txt

Sử dụng 2 FIFO và giá trị sau khi được đọc bởi phần cứng sẽ được cộng 1:

Text

Description automatically generated

Hình 4. Ví dụ về ghi giá trị của FIFO

Text

Description automatically generated

Hình 4. Kết quả đọc giá trị của FIFO

## Kiểm tra các thư viện đã cài đặt

### OpenCV

* Kiểm tra cài đặt:

Text

Description automatically generated

Hình 4. OpenCV 2.4.9

* Chạy thử nghiệm:

Bảng 4. Source code Capture camera sử dụng OpenCV

|  |
| --- |
| #include "opencv2/opencv.hpp"  #include "opencv2/highgui/highgui.hpp"  #include "opencv2/core/core.hpp"  #include "opencv2/imgproc/imgproc.hpp"  #include "iostream"  int main(int, char\*\*) {  cv::VideoCapture camera(0);  if (!camera.isOpened()) {  std::cerr << "ERROR: Could not open camera" << std::endl;  return 1;  }  cv::namedWindow("Webcam", CV\_WINDOW\_AUTOSIZE);  cv::Mat frame;  while (1) {  // show the image on the window  camera >> frame;  cv::imshow("Webcam", frame);  if (cv::waitKey(10) >= 0)  break;  }  return 0;  } |

Compile:

g++ capture.cpp -o capture `pkg-config --cflags --libs opencv`

Kết quả:

A picture containing text, monitor, electronics, display

Description automatically generated

Hình 4. OpenCV Camera

### Gstreamer + RTSP

* Kiểm tra cài đặt:

Graphical user interface

Description automatically generated with medium confidence

Hình 4. GStreamer 1.8.0

* Chạy thử nghiệm:

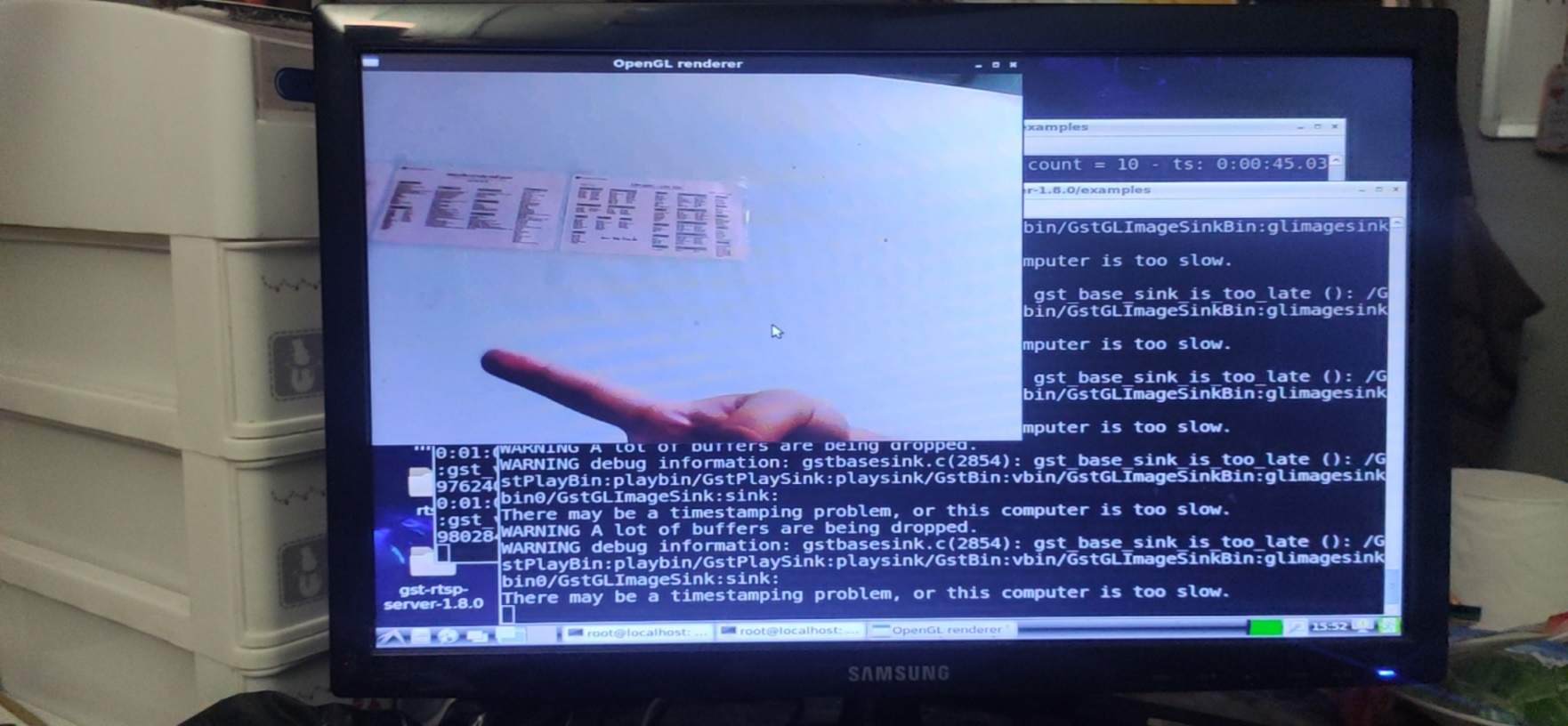
Sử dụng example của gstreamer

Sử dụng câu lệnh cho sender:

./test-launch --gst-debug=3 '( v4l2src device=/dev/video0 ! video/x-raw,width=640,height=480,framerate=25/1 ! x264enc tune="zerolatency" byte-stream=true bitrate=800 ! rtph264pay name=pay0 pt=96 )'

Sử dụng câu lệnh cho receiver:

gst-play-1.0 rtsp://127.0.0.1:8554/test



Hình 4. Video Streaming từ luồng

# HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN

## Hạn chế

Chỉ dừng ở việc loopback giá trị khi giao tiếp giữa FPGA và OS. Chưa thể tự xây dựng lại kernel phù hợp với ứng dụng, chỉ đang sử dụng lại xillinux do nhà phát hành cung cấp. Tốc độ khi streaming video là thấp và chưa tìm được cách public IP để đưa luồng dữ liệu ra bên ngoài.

## Hướng phát triển

Nâng cấp camera có khả năng tự encode để giảm tải áp lực lên board hoặc sử dụng các board có hiệu năng cao hơn.

Thiết kế phần cứng có thể giao tiếp được với luồng dữ liệu được đọc ra từ usb camera.

Thiết kế website có giao diện để dẫn luồng streaming camera.

Xây dựng kernel riêng với các thư viện được cài đặt sẵn để tăng độ tương thích.

# TÀI LIỆU THAM KHẢO

[1] Xillybus Product Brief

[2] Xillybus Getting Started Zynq

[3] Xillybus FPGA API

[4] <https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol> [Online]

[5] <https://en.wikipedia.org/wiki/Secure_Shell> [Online]

[6] <https://gstreamer.freedesktop.org/features/index.html> [Online]