Qt Overlay on Video

Example Code Introduction for 64/32-bit NuMicro® Family

Document information

Application	This example code demonstrates how to write a Qt application that simultaneously plays a video at the bottom layer of the screen and overlays interactive elements on top, allowing the video playback to be controlled through these interactive components.
BSP version	Linux-5.10.x
Hardware	NuMaker-HMI-MA35D1-S1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

Compositing is a computer graphics technique that combines multiple images or layers into a single output image, as illustrated in Figure 1-1. It is commonly used in video and image processing, such as video and image compositing.

This example code demonstrates how to simultaneously play a video and run a Qt widget in a Qt application while displaying them on the screen. The video and Qt application are output to different frame buffers, such as Frame Buffer 0 and Frame Buffer 1, and then blended using the blending function provided by the display driver before being displayed on the screen. The MA35D1 display driver offers two types of blending functions: Color Key and Alpha Blending, and this example code uses Alpha Blending.

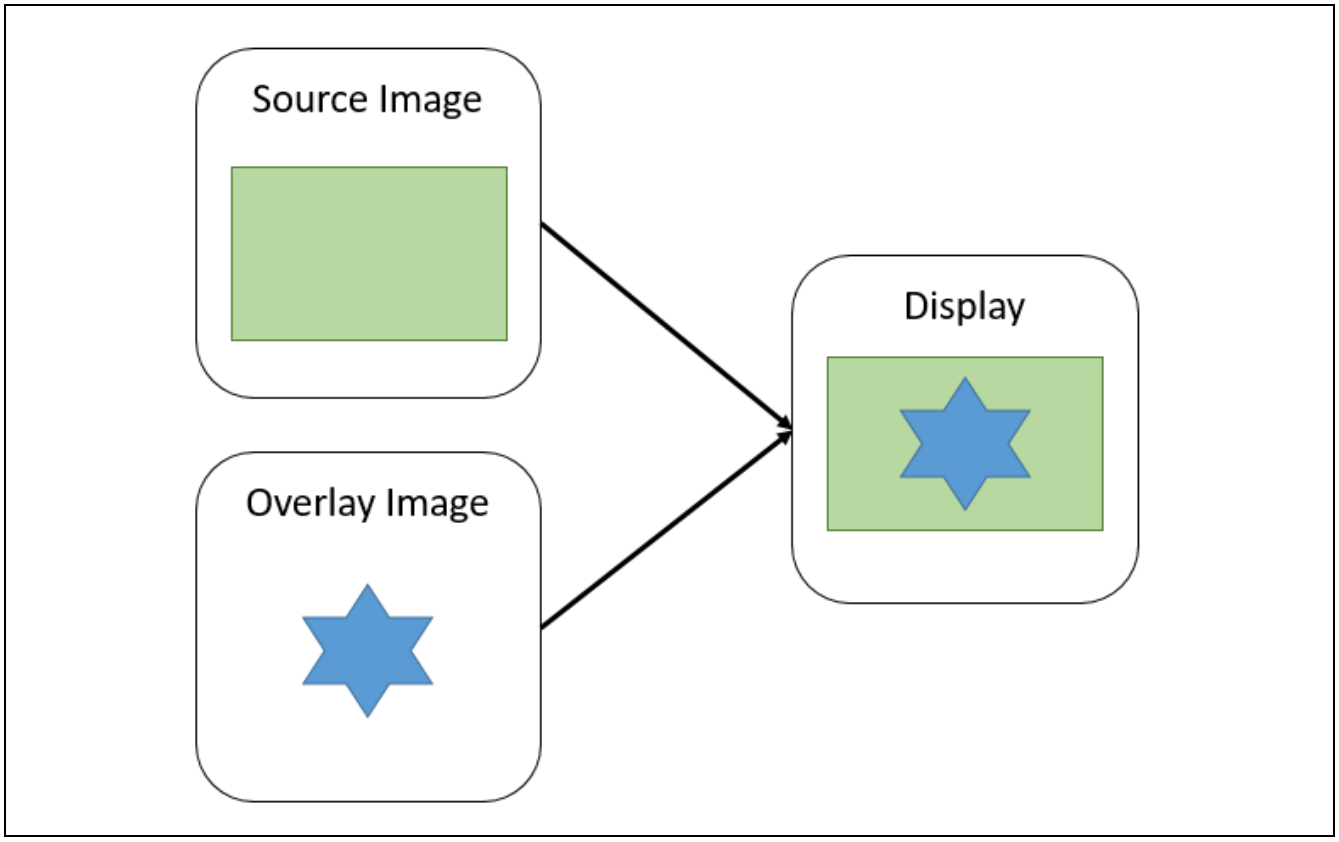

Figure 1-1 Compositing Diagram

1.1 Software Architecture

The software architecture is illustrated in Figure 1-2. The Qt application uses the Gstreamer library to play the video, and Gstreamer uses the VC8000 hardware decoder to decode the video and output it directly to Frame Buffer 0. The Qt application uses the Qt drawing library to draw the components that need to be displayed on the video, which are then output to Frame Buffer 1. The display controller subsequently blends the video and components for display.

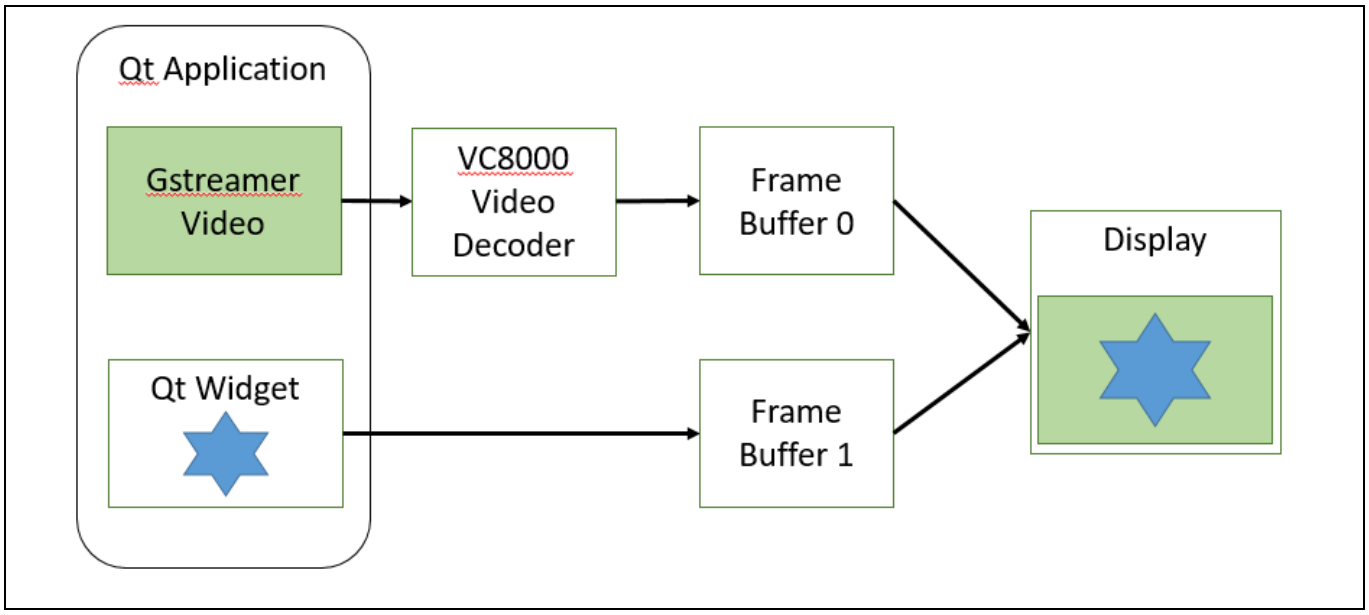

Figure 1-2 Software Architecture

1.2 Driver Settings

1.2.1 Display Device Tree Settings

Confirm the display resolution and enable overlay related options. Users can modify the display driver resolution according to their display. The following demo code uses a resolution of 1024x600.

```
/* Ensure overlay related setting */
&display {
    ....
    overlay-en = <1>; /* Enable overlay function */
    overlay-pixel-fmt = <6>; /* Set RGB888 format */
    overlay-width = <1024>; /* Set overlay resolution width */
    overlay-height = <600>; /* Set overlay resolution height */
    overlay-rect-tlx = <0>; /* Set the starting coordinate of the x-axis */
    overlay-rect-tly = <0>; /* Set the starting coordinate of the y-axis */
    ....
};
```

2. Code Description

To edit and compile the Qt application in this example code, you will need to use Qt Creator. Follow these steps:

1. GStreamer and Qt is a framework for creating streaming media applications and writing multi-platform applications. For the Installation on MA35D1, refer to Environment Setup chapter in MA35D1 Buildroot Quick Start. If you use Yocto, no setup is required because Yocto's default configuration have already included Gstreamer and Qt.
2. Refer to Qt Creator Cross-Compile Environment Setup chapter in MA35D1 HMI GUI Development Environment to set up Qt Creator environment.
3. Open Qt Creator and navigate to the "Open Project" dialog.
4. Copy *EC_MA35D1_Qt_Overlay_on_Video_V1.00\Src\MA35D1_Qt_Overlay_on_Video* project to Linux and open it; you should see a screen similar to Figure 2-1.
5. Double-click the "*MA35D1_Qt_Overlay_on_Video.pro*" file to open it.

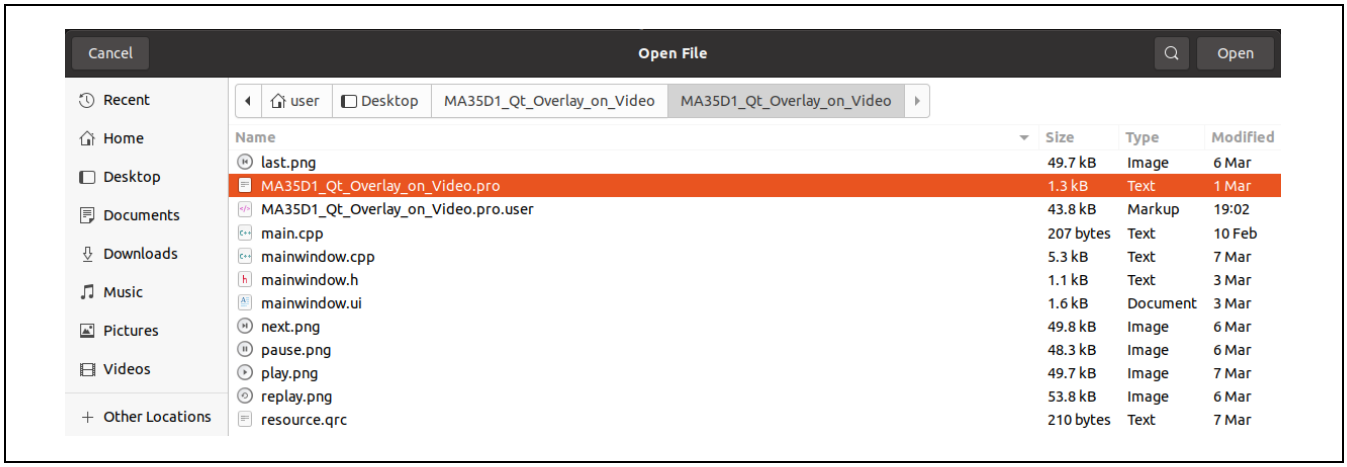

Figure 2-1 Qt Project Dialog

The main behavior of the example code is implemented in the *mainwindow.cpp* file. The example code uses Gstreamer to play videos repeatedly from the path */opt/videos* and draws a Qt widget to display them. The example code provides five buttons for controlling video playback: pause, resume, replay, last, and next. Each button is associated with a specific function, which is triggered when the button is pressed.

```
#include <QApplication>
#include <QDebug>
#include <QDir>
#include <stdio.h>
#include <stdlib.h>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTouchEvent>
#include <QThread>
```

```
#include <QTimer>
#include <gst/gst.h>

/* Define Videos location */
#define VIDEOS_FD_PATH "/opt/videos/"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    /* Gstreamer Setting */
    cur_video_idx = 0;
    QDir dir(QString::fromStdString(VIDEOS_FD_PATH));
    dir.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);
    dir.setSorting(QDir::Size | QDir::Reversed);
    v_list = dir.entryInfoList();
    QString v_path = "gst-launch-1.0 filesrc location=" +
v_list.at(cur_video_idx).filePath() + " ! qtdemux name=demux demux.audio_0 ! queue !
decodebin ! audioconvert ! audioresample ! autoaudiosink demux.video_0 ! queue !
decodebin ! nufbdevsink ! fakesink sync=true ";
    gst_init (NULL, NULL);
    pipeline = gst_parse_launch (v_path.toStdString().c_str(), NULL);

    /* Set Icon */
    ui->play_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/pause.png); background-color: transparent;}");
    ui->replay_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/replay.png); background-color: transparent;background-position: center;}");
    ui->next_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/next.png); background-color: transparent;}");
    ui->last_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/last.png); background-color: transparent;}");

    /* This timer is used to auto replay the video */
    thread = new QThread(this);
    timer = new QTimer(0);
    timer->setInterval(1000);
    timer->moveToThread(thread);
    connect(timer, SIGNAL(timeout()), this, SLOT(video_update()),Qt::DirectConnection);
    connect(thread, SIGNAL(started()), timer,SLOT(start()));
    thread->start();

    /* Transparent Background */
    this->setAttribute(Qt::WA_TranslucentBackground, true);

    /* Start to Play Video */
    gst_element_set_state (pipeline, GST_STATE_PLAYING);
    playing = true;
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::video_update()
{
    bus = gst_element_get_bus(pipeline);
    msg = gst_bus_timed_pop_filtered (bus,
GST_CLOCK_TIME_NONE,(GstMessageType)(GST_MESSAGE_ERROR | GST_MESSAGE_EOS |
GST_MESSAGE_UNKNOWN ));
    switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_ERROR:
        qDebug() << "Stream Error";
        next_video();
        break;

    case GST_MESSAGE_EOS:
        qDebug() << "End-of-Stream";
        gst_element_set_state (pipeline, GST_STATE_NULL);
        gst_object_unref (pipeline);
        gst_message_unref (msg);
        gst_object_unref (bus);
        next_video();
        break;

    default:
        break;
    }

}

/* Play Video Action */

void MainWindow::resume_video()
{
    gst_element_set_state (pipeline, GST_STATE_PLAYING);
}

void MainWindow::pause_video()
{
    gst_element_set_state (pipeline, GST_STATE_PAUSED);
}

void MainWindow::replay_video()
{
    if(v_list.size() == 0)
        return;
    cur_video_idx--;
    cur_video_idx = cur_video_idx < 0 ? v_list.size()-1 : cur_video_idx;
    gst_element_post_message (pipeline, gst_message_new_eos(GST_OBJECT(pipeline)));
}

void MainWindow::next_video()
{
    if(v_list.size() == 0)
        return;
    cur_video_idx++;
    cur_video_idx = (cur_video_idx)%v_list.size();
    play_video();
}
```

```
void MainWindow::last_video()
{
    if(v_list.size() == 0)
        return;
    cur_video_idx-=2;
    if(cur_video_idx <= -2)
        cur_video_idx = v_list.size()-2;
    gst_element_post_message (pipeline, gst_message_new_eos(GST_OBJECT(pipeline)));
}

void MainWindow::play_video()
{
    gst_init (NULL, NULL);
    QString v_path = "gst-launch-1.0 filesrc location=" +
v_list.at(cur_video_idx).filePath() + " ! qtdemux name=demux demux.audio_0 ! queue !
decodebin ! audioconvert ! audioresample ! autoaudiosink demux.video_0 ! queue !
decodebin ! nufbdevsink ! fakesink sync=true ";
    pipeline = gst_parse_launch (v_path.toStdString().c_str(), NULL);
    gst_element_set_state (pipeline, GST_STATE_PLAYING);
}


void MainWindow::on_play_button_clicked()
{

    if(playing == true)
    {
        qDebug() << "Pause";
        pause_video();
        ui->play_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/play.png); background-color: transparent;}");
    }
    else
    {
        qDebug() << "Resume";
        resume_video();
        ui->play_button->setStyleSheet("QPushButton{ border-style: none; border-image:
url(:/pause.png); background-color: transparent;}");
    }
    playing = !playing;
}

void MainWindow::on_replay_button_clicked()
{
    qDebug() << "Replay";
    replay_video();
}

void MainWindow::on_next_button_clicked()
{
    qDebug() << "Next";
    gst_element_post_message (pipeline, gst_message_new_eos(GST_OBJECT(pipeline)));
}

void MainWindow::on_last_button_clicked()
{
```

```
    qDebug() << "Last";
    last_video();
}
```

The UI design for Qt Creator is displayed in Figure 2-2. The buttons are positioned at the bottom of the screen and have a size of 100x100 pixels. The overall resolution of the UI is set to 1024x600 pixels.

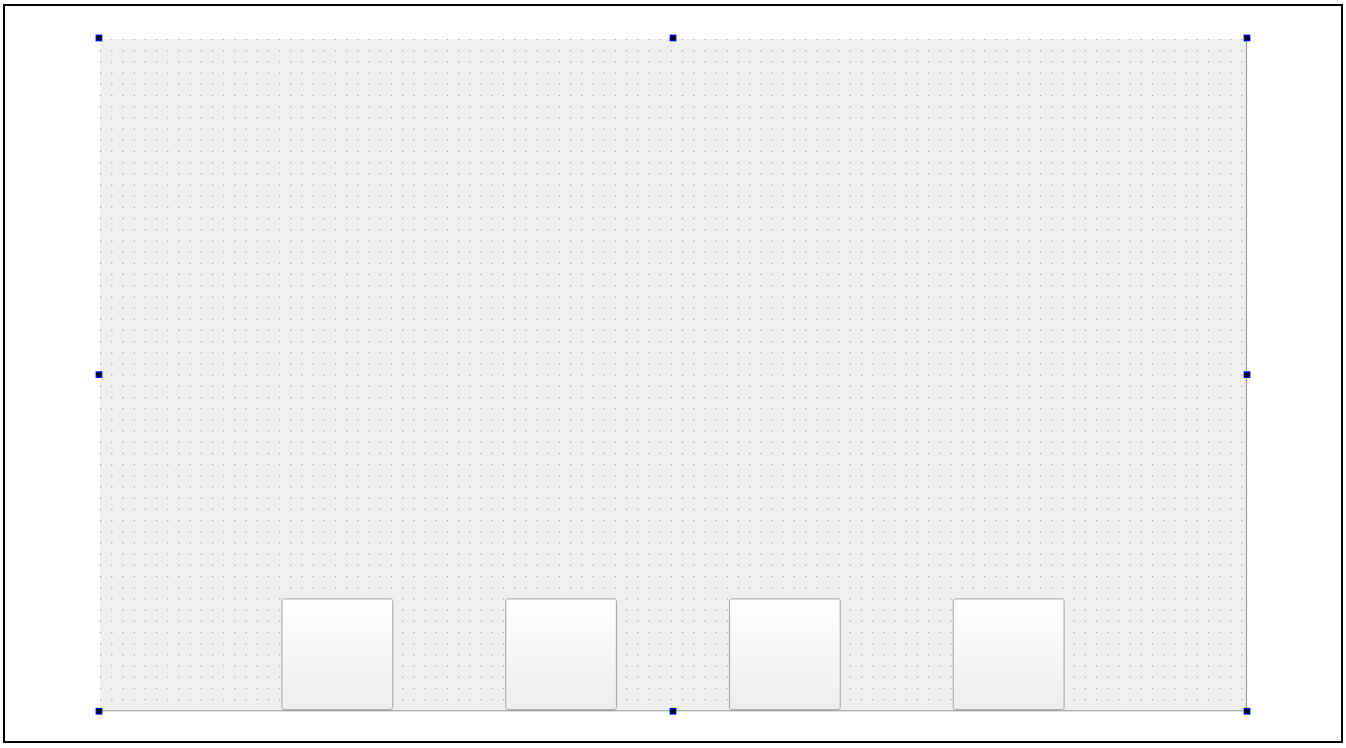

Figure 2-2 mainwindow.ui Diagram

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x

3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1-S1

4. Directory Information

The directory structure is shown below.

EC_MA35D1_Qt_Overlay_on_Video_V1.00	
Src	Source files of Qt project
DT-Binding	Device tree requirements on the contents of driver

Figure 4-1 Directory Information

5. Example Code Execution

1. Set up the display device tree (Refer to section 1.2.1 Display Device Tree Settings).
2. Re-compile MA35D1 Image and program it.
3. After MA35D1 boot, refer to MA35D1 HMI GUI Development Environment to set up Qt and tslib environment variable.
4. Copy MA35D1_Qt_Overlay_on_Video application to the evaluation board.
5. Copy the video to */opt/videos/*.
6. Execute the following command:

```
./MA35D1_Qt_Overlay_on_Video -platform linuxfb:fb=/dev/fb1
```

7. The screen will display as shown below.

Figure 5-1 MA35D1_Qt_Overlay_on_Video

6. Revision History

Date	Revision	Description
2024.02.16	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices , and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.