Product Overview

如何在 AM62 SK EVM 上使用 Flutter™ 进行开发

Google 的 Flutter™ 软件开发套件可用于进行简单的界面编程。本产品概述是由 TI 和 TI 的 Flutter 第三方合作伙伴 Klarälvdalens Datakonsult AB (KDAB) 共同编写的分步指南，其中包含使用 TI AM62x 入门套件 (SK) 评估模块 (EVM) 进行基于 Flutter 的人机界面 (HMI) 开发的说明。

Flutter 是否是合适我的 GUI 工具套件？-工程师视角

在当今世界，人们倾向于花更多时间通过显示屏与机器交互，而不是通过机械方式控制机器。例如，许多新型汽车或家用电器的主屏幕上都运行着图形用户界面 (GUI)。通过 GUI，用户可以控制设备并激活音乐、广播、电话、温度控制等功能。同样，在机场或餐厅，公众可通过自助式售货亭与相关公司进行通信。售货亭在显示设备上运行 GUI，客户使用售货亭触摸屏，通过机器即可满足要求。总之，业界的趋势是终端消费者越来越趋向于通过与 GUI 进行交互，便捷高效地完成任务。

从开发人员的角度而言，打造美观的 GUI 至关重要，因为 GUI 是客户与产品之间通信的关键要素之一。要在 AM62 等嵌入式器件上运行 GUI，有多个 GUI 框架可供选择。每个框架都是独特的，并支持以下部分特性：

1. 触控：大多数 GUI 框架支持电阻式或电容式触控，可连接显示器。
2. 美观性：外观更现代或功能更易用的 GUI 可吸引更多客户。
3. 硬件 (HW) 加速：一些嵌入式器件具有图形处理单元 (GPU)，能够执行各种任务，例如 α 混合、颜色转换、缩放和旋转。重要的是，GUI 可利用 GPU 渲染复杂场景，并减轻中央处理器 (CPU) 的负载，以便处理其他任务。
4. 存储器：开发人员使用的存储器空间有限，因此必须确保 GUI 及其软件栈符合空间占用要求。
5. 可扩展性：根据项目的不同，开发人员更倾向于使用可跨多个平台移植的 GUI。使用可用于多个器件和操作系统 (OS) 的框架可能会很有用。
6. 许可证：软件许可也是需要考虑的一个关键方面。

如何在 AM6254 Arm® 处理器系列上启用 Flutter

尽管选择 GUI 框架具有挑战性，但在嵌入式平台上启用和优化 GUI 框架是一项要求更高的任务。本产品概述旨在帮助开发人员在 TI 平台上启用和优化 Flutter 框架。TI 的下一代器件 (AM625) 已支持多种现代 GUI。有关更多信息，请参阅 AM62x 设计库。借助 TI 器件和随附的软件开发套件 (SDK)，开发人员可灵活选择任何框架。

Flutter 最近获得了广泛关注，它支持触控，提供现代 GUI，并具有更小的存储器栈和较宽松的许可条款。如果开发人员有兴趣使用 Flutter 并希望在 TI 的嵌入式平台上进行开发，可以按照以下流程操作，具体步骤如下：

1. 完成适用于 AM62x 的 Processor SDK Linux 列出的一次性设置先决条件中的要求。
2. 按顺序在控制台中输入以下命令：
   a. `git clone https://git.ti.com/git/arago-project/oe-layersetup.git tisdk`
   b. `cd tisdk`
   c. `cp configs/processor-sdk/processor-sdk-08.05.00.21-config.txt configs/flutter-config.txt`
   d. `echo "meta-flutter,https://github.com/ardera/meta-flutter.git,dunfell,HEAD" >> configs/flutter-config.txt`
   e. `./oe-layertool-setup.sh -f configs/flutter-config.txt`
   f. `cd build/`
   g. `export TOOLCHAIN_PATH_ARMV7=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf`
   h. `export TOOLCHAIN_PATH_ARMV8=$HOME/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu`
   i. `echo 'IMAGE_INSTALL_append = " flutter-pi-runtimedebug flutter-gallery-runtimedebug "' >> conf/local.conf`
   j. `echo 'TOOLCHAIN_HOST_TASK:append = " nativesdk-flutter-sdk"' >> conf/local.conf`
   k. `echo 'FLUTTER_SDK_TAG = "stable"' >> conf/local.conf`
   l. `echo 'SRCREV_pn-flutter-gallery-runtimedebug = "9776b9fd916635e10a32bd426fcd7a20c3841faf"' >> conf/local.conf`
   m. `. conf/setenv`
   n. `MACHINE=am62xx-evm bitbake-layers add-layer ../sources/meta-flutter`
   o. `MACHINE=am62xx-evm bitbake tisdk-default-image`
3. 将以下映像刷写到 SD™ 卡：
   - `arago-tmp-external-arm-glibc/deploy/images/am62xx-evm/tisdk-base-image-am62xx-evm.wic.xz`
4. 在评估模块上运行以下命令：
   - `flutter-pi /usr/share/flutter/gallery/`

AM62xx 的应用特定性能改进

在为 AM62xx 平台优化 Flutter 应用时，GPU 是主要考虑因素。AM62x 上的 GPU AXE-1-16M 是分块延迟渲染器 (TBDR)，与较常见的即时模式渲染器 (IMR) 运行方式不同。有关 TBDR 的更多信息，请参阅以下文章 ***了解 PowerVR 图形架构*** :

- *分块渲染*
- *延迟渲染*

开发 GPU 架构旨在将对存储器的读取和写入保持在最低水平，这是实现高效操作的基本要求。部分操作会妨碍 GPU 将存储器流量保持在最低水平。GPU 的存储器带宽也受限，以便实现低功耗操作。

Imagination Technologies™ 提供了对 PowerVR 硬件进行性能优化的指南，但并未提供 Flutter 环境内的很多具体详细信息。有关使用 AM62 硬件开发 Flutter 应用的建议如下：

默认情况下，所有 Flutter 图像都启用了 α 混合。尽可能避免 α 混合。α 混合是通过将图像与背景结合实现透明效果。使用以下提示和技巧，可确保尽量少使用 α 混合。

- 尽可能不使用 GPU 进行 *α 混合*。例如，假设图像具有白色背景，顶部具有蓝色元素。如果要使蓝色元素显示透明效果，最好将颜色定义为较浅的蓝色，而不是提供蓝色元素的透明度并让 GPU 确定颜色。
- 如果必须使用 α 混合，请尽量减少需要进行 α 混合的像素数量。不要绘制透明对象或像素，而应将其完全删除。下面给出了如何使用图像实现这一点的示例。
- KDAB 提供的此示例使用图像子集工具从图像中删除所有透明像素。这样可确保仅将可见像素发送到 GPU 进行处理，并减少所需进行的 α 混合。此示例的自述文件介绍了如何使用示例中提供的图像子集工具。

- 通过遵循 Flutter 文档要求并指定 BlendMode.src（而不是默认的 BlendMode.srcOver），绘制不使用 α 混合的图像。这无法通过默认图像小部件来实现，而是需要使用 CustomPainter 部署的定制图像小部件。

避免使用纹理和图像也*可以*提高效率和性能。分块渲染 GPU 可非常高效地绘制纯色三角形。使用经光栅化的元素时，通常需要更多的存储器带宽。

- 可以使用 CustomPainter（而不是图像）来绘制顶点或路径。
- 存储器带宽要求取决于纹理插值技术。使用最近邻插值可高效节省带宽，而使用 FilterQuality.none 则需使用最近邻插值。请参阅 Flutter 文档，了解更多信息。

以下示例说明了如何在不使用 α 混和和最近邻插值的情况下绘制图像。

```
import 'package:flutter/rendering.dart';
import 'package:flutter/services.dart';
import 'package:flutter/widgets.dart';
import 'dart:ui' as ui;

/// Draw the background image with a custom painter.
class BackgroundImagePainter extends CustomPainter {
  BackgroundImage(this.image);

  final ui.Image image;

  final Paint imagePaint = Paint()
    ..blendMode = BlendMode.src // Disables alpha blending.
    ..filterQuality = FilterQuality.none; // Nearest neighbor interpolation.

  @override
  void paint(Canvas canvas, Size size) {
    canvas.drawImage(image, Offset.zero, imagePaint);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) {
    return oldDelegate.image != image;
  }
}

class BackgroundImage extends StatelessWidget {
  const BackgroundImage({super.key, this.image});

  final ui.Image image;

  @override
  Widget build(BuildContext context) {
    return CustomPaint(
      painter: BackgroundImage(
        image: image
      ),
    );
  }
}
```

总结

GUI 是人机交互的重要方面。选择合适的 GUI 可能问题重重，但是无论开发人员如何选择，TI 的下一代器件 AM62x 均能运行任何现代 GUI。如需更多信息，请参阅在 TI AM6254 使用 Flutter 演示。

关于 KDAB

KDAB 可在产品开发周期的任何阶段为客户提供帮助，包括全栈开发计划和架构、现代开发流程、工具和持续集成。在嵌入式器件方面，KDAB 提供了特别深入的专业知识，涵盖软件栈的所有部分，特别是操作系统（通常是嵌入式 Linux™）、UI 框架（Qt、Flutter 等）以及给定硬件上的性能优化。

作者

KRUNAL BHARGAV 是 TI 的系统应用工程师，主要从事图形和显示支持工作。自 2017 年起，Krunal 一直在 TI 工作，参与设计嵌入式 HMI 系统中的产品并提供支持。Krunal 拥有美国新泽西州新泽西理工学院的硕士学位。

Hannes Winkler 自 2021 年起担任 KDAB 的软件工程师，拥有马格德堡大学计算机科学学士学位。他是适用于嵌入式 Linux 的 Flutter 引擎嵌入器 Flutter-pi 的开发者，也为部分 Flutter 引擎和工具功能的开发做出了卓越贡献。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265