Product Overview

How to Develop With Flutter™ on the AM62 SK EVM

The Flutter™ software development kit from Google is used for simple interface programming. This product overview is written by TI and TI's Flutter third-party partner Klarälvdalens Datakonsult AB (KDAB), providing a step-by-step guide with instructions for initiating a Flutter-based development for human-machine interface (HMI) using the TI AM62x starter kit (SK) evaluation module (EVM).

Is Flutter the Right GUI Toolkit For Me?: An Engineer's Perspective

In today's world, people tend to spend more time interacting with machines through a display screen rather than mechanically controlling machines. As an example, there is a graphical user interface (GUI) running on the main screen of many new automotive vehicles or home appliances. Through the GUI, users can control devices and activate features, such as music, radio, phone, temperature control, and more. Similarly, at the airport or at restaurants, the public can communicate with the companies through a self-serving kiosk. The kiosk runs a GUI on a display device and through the kiosk touchscreen, machines can fulfill the customer's request. In summary, there is an increasing trend in the industry for the end consumer to interact with GUIs in a convenient and efficient manner to accomplish tasks.

From a developer's perspective, creating an elegant GUI is vital because GUIs are one of the key components of communication between the customer and the product. There are many GUI frameworks to select from for running GUIs on embedded devices like the AM62. Each framework is unique and supports some of the following features:

1. Touch: Most GUI frameworks connect a display through resistive or capacitive touch support.
2. Aesthetics: GUIs with a more modern appearance or user-friendly features attracts more customers.
3. Hardware (HW) acceleration: Several embedded devices have a graphics processing unit (GPU) and are capable of performing various tasks like alpha blending, color conversion, scaling, and rotation. Crucially, the GUI utilizes the GPU for rendering complex scenes and offloads the central processing unit (CPU) for other tasks.
4. Memory: Developers are working with a limited memory footprint so developers must verify that the GUI and their software stack fits within the required footprint.
5. Scalability: Depending on the project, developers prefer to have a GUI that is portable across multiple platforms. Having a framework that scales across multiple devices and operating system (OS) can be beneficial.
6. License: Software licensing is also a key aspect to consider.

How to Enable Flutter on the AM6254 Arm® Processor Family

Although choosing a GUI framework is challenging, enabling and optimizing the GUI framework on an embedded platform is an even more demanding task. The objective of this product overview is to assist developers in enabling and optimizing the Flutter framework on TI platforms. Several modern GUIs are already enabled for TI's next generation device (AM625). See the AM62x Design Gallery for more information. TI devices and the accompanying software development kit (SDK) offer flexibility to the developers to choose from any framework.

Flutter has recently gained traction and has support for touch, includes contemporary GUI, contains smaller memory stack, and has a less stringent licensing clause. Developers that are interested in using Flutter and want to develop on TI's embedded platform, can follow the process described in the following steps:

1. Complete the one-time setup prerequisites listed in the Processor SDK Linux for AM62x.
2. Enter the following commands, in order, into the console:
   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. Perform a flash on an SD™ card with the following image:
   - `arago-tmp-external-arm-glibc/deploy/images/am62xx-evm/tisdk-base-image-am62xx-evm.wic.xz`
4. On the evaluation module run the following:
   - `flutter-pi /usr/share/flutter/gallery/`

Application-Specific Performance Improvements For AM62xx

The GPU is a primary concern when optimizing Flutter applications for the AM62xx platform. The GPU on AM62x, AXE-1-16M, is a tile-based deferred renderer (TBDR), and functions in a different way than the more common immediate mode renderers (IMRs). For more information about TBDR, see the following ***A look at the PowerVR graphics architecture*** articles:

- *Tile-based rendering*
- *Deferred rendering*

The GPU architecture was developed to keep reading and writing to and from memory to a minimum, this is a fundamental part of the efficient operation. Some operations hinder the ability of the GPU to keep memory traffic to a minimum. The memory bandwidth of the GPU is also limited to provide low-power operation.

Imagination Technologies™ provides guidelines for performance optimizations on PowerVR hardware; however, many low-level details within the Flutter environment are not accessible. Recommendations for developing Flutter applications with AM62 hardware follow:

By default, all Flutter images have alpha-blending enabled. Avoid alpha blending, where possible. Alpha blending is when an image is combined with a background to provide a transparency effect. Use the following tips and tricks to keep alpha blending to a minimum.

- Where possible *alpha blend* without using the GPU. As an example, consider an image with a white background and a blue element on top. If the blue element is to appear transparent it is better to define the color as a lighter blue, rather than to give the blue element transparency and let the GPU determine the color.

- If alpha blending is unavoidable, keep the amount of pixels that need to be alpha blended to the minimum. Do not draw transparent objects or pixels, rather remove them completely. The next point gives an example of how to do this with images.
- This example by KDAB uses a image subsetting tool to remove all transparent pixels from an image. This makes sure that only visible pixels are sent off to the GPU for processing and reduces the amount of alpha blending that must be done. The README of the example describes how to use the image subsetting tool that is provided with the example.
- Draw the images without alpha blending by following the Flutter documentation and specifying BlendMode.src instead of the default BlendMode.srcOver. This specification is not possible through the default image widget and requires that a custom image widget be implemented using a CustomPainter.

Avoiding textures and images *can* also improve power efficiency and performance. The tiling GPU draws solid color triangles very efficiently. Working with elements that have already been rasterized often requires more memory bandwidth.

- Vertices or Paths can be drawn using a CustomPainter instead of using a image.
- Memory bandwidth requirements depend upon texture interpolation techniques. Using nearest-neighbor interpolation is the most efficient technique for saving bandwidth and using FilterQuality.none uses nearest-neighbor interpolation. See the Flutter documentation for more information.

The following example shows how to draw an image without alpha blending and nearest-neighbor interpolation.

```
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
      ),
    );
  }
}
```

Summary

GUIs are an essential aspect of interaction with machines. Selecting an appropriate GUI can be problematic but regardless of the GUIs chosen by the developer, TI's next generation device, the AM62x, is capable of running any modern GUI. For additional information, see the Flutter on the TI AM6254 demonstration.

About KDAB

KDAB helps customers in any stage of their product development cycle from planning and architecture over full-stack development to modern development processes, tooling, and continuous integration. When it comes to embedded devices, KDAB provides particular in-depth expertise in all parts of the software stack, especially the operating system (usually Embedded Linux™), the UI framework (Qt, Flutter and others), and performance optimization on a given hardware.

Authors

KRUNAL BHARGAV is a System Application Engineer at TI where he primarily supports graphics and display. Krunal has been with TI since 2017 and is involved in designing and supporting products in Embedded HMI Systems. Krunal earned his Master's degree from the New Jersey Institute of Technology, New Jersey, USA.

HANNES WINKLER is a software engineer for KDAB since 2021 and bachelor student of computer science at Otto-von-Guericke Universität Magdeburg. He is the author of flutter-pi, a Flutter engine embedder for embedded Linux, and contributor of some Flutter engine and tool features.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265