DRM / KMS Magic:
Enabling Displays in Embedded Devices

Date: 2024-03-16

Aradhya Bhatia

Texas Instruments

Introduction to the Speaker

Aradhya Bhatia,
Software Engineer at Texas Instruments, Bangalore.

Aradhya is a Linux-Kernel Engineer and has been working for Texas Instruments (TI) for 2+ years now. It was in TI that he started working in the kernel-space, and had only dabbled with user-space and baremetal development prior to that.
In TI, the focus of his work has been with respect to displays over Linux and the DRM subsystem. He enables, maintains and help architect various display hardware for the TI microprocessor SoCs.

Disclaimers

- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.

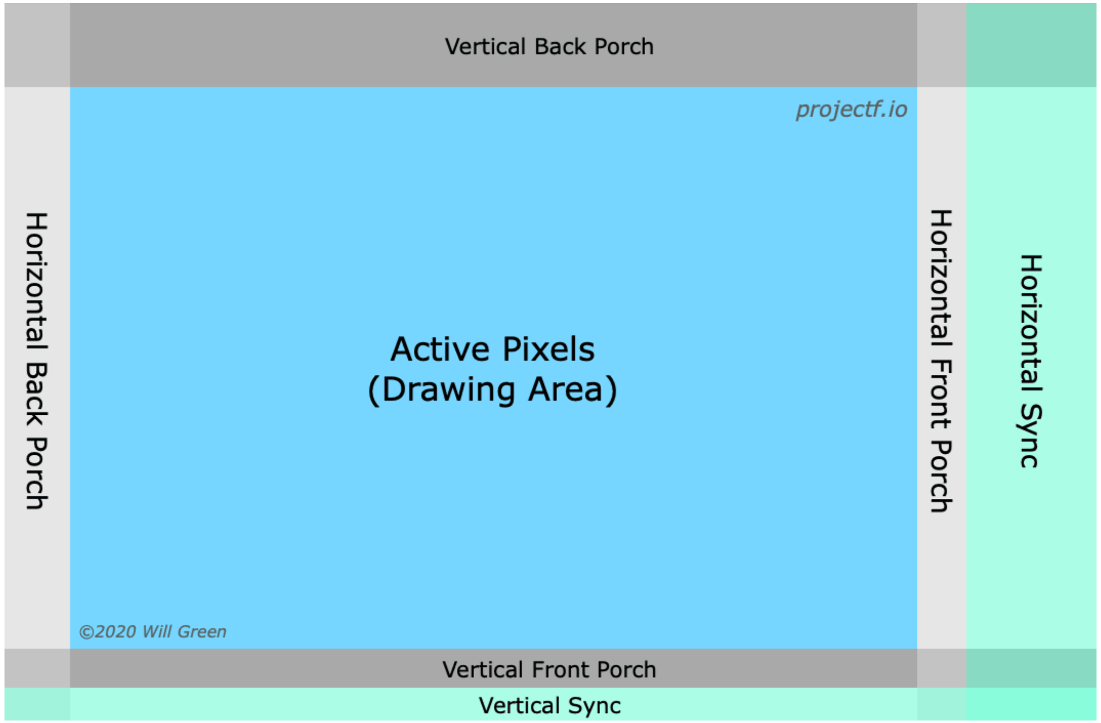
Overview

- Display Basics
- Different Display Interfaces
- Hardware Components: DDR to Display
- DRM and Device-tree
- DRM Objects for Hardware

Display Basics

- Timing Representation of a Frame
  - Active vs Blanking Intervals

- Types of color spaces
  - RGB
  - YUV



Texas Instruments

Display Interfaces

- MIPI DPI – Display Pixel Interface
- OLDI – Open LVDS Display Interface

Display Interfaces

MIPI DPI - Display Pixel Interface

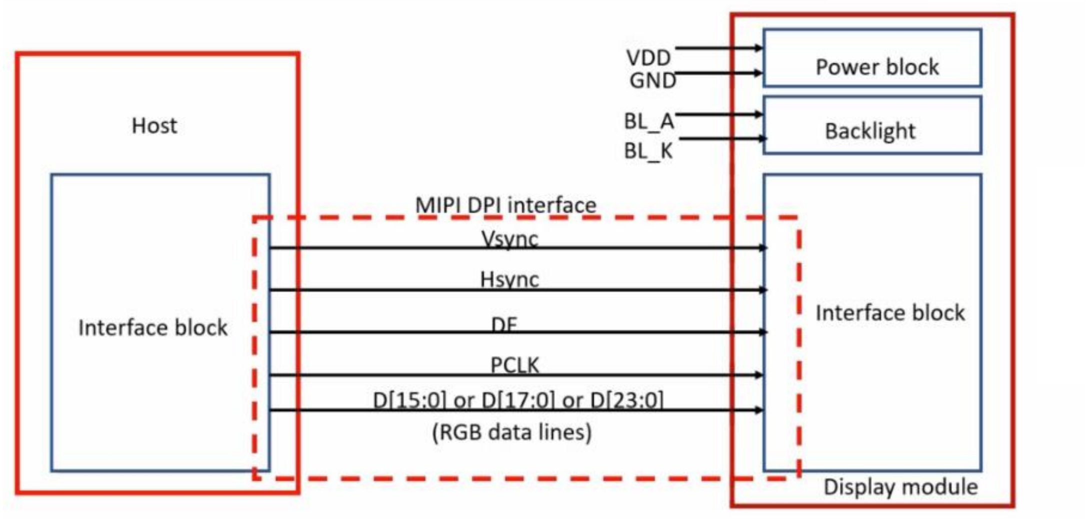


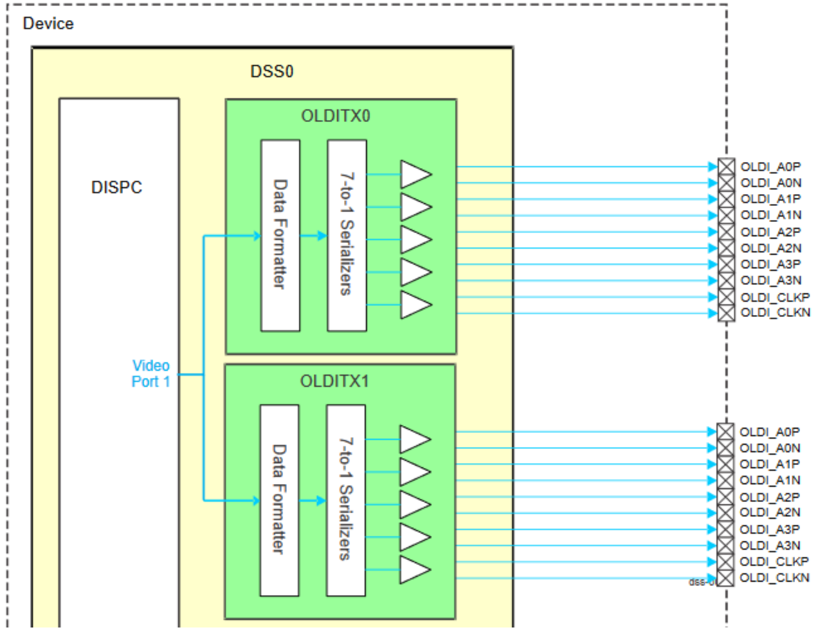
Figure shows a display module connected to the Host(Microcontroller) using MIPI DPI interface

- Simplest Interface (Parallel RGB)
- Useful for when Display module has no G-RAM
- Every clock-pulse a pixel is transmitted.
- Pixel format here can be of multiple types, based on
  - Color Space (RGB/YUV)
  - Pixel Size
- RGB888, RGB565, YUV422

Texas Instruments

Display Interfaces

OLDI - Open LVDS Display Interface



- LVDS signaling based (+/- differential pair)
  - Remote Display Setups
  - Low Noise Interference
- Packs pixels (RGB666/888) format into OLDI packets
- 3 or 4 pairs for video + 1 for clock
- Single-Link vs Dual-Link Modes
  - Reduces frequency load
- Automotive & Industrial markets

Display Interfaces

Summary

- MIPI DPI – Display Pixel Interface
- OLDI – Open LVDS Display Interface

-- Other popular Interfaces --

- MIPI DSI – Display Serial Interface
- HDMI
- DP/eDP

Hardware: DDR to Display

- Memory
- Display Controller
  - Video Pipes
  - Overlayers
  - Video Ports
- Bridges
- Connectors / Always-Connected Panels

Hardware: DDR to Display

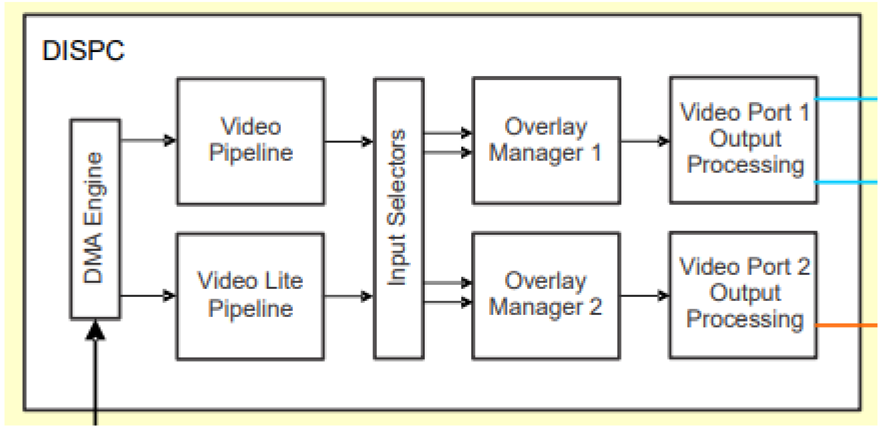
Memory and Framebuffers

- Framebuffer: Memory region(s) where user-space apps store rendered pixel data
- Types of Memories
  - Dedicated Memory in Graphics Card (video RAM)
    - Not an embedded use-case
  - Shared Memory with the system
    - With / Without IOMMUs

Texas Instruments

Hardware: DDR to Display

Display Controller: Overview



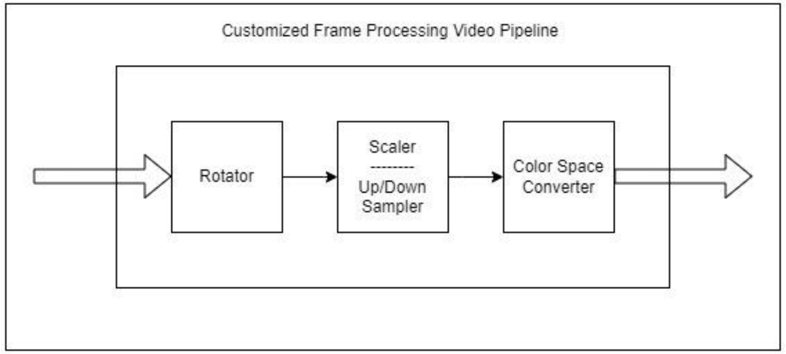
- The key hardware block that generates the video signals from the pixels stored inside the memory.
- Tasked with numerous operations
  - Video processing of the input frame
  - Plane Overlaying
  - Generating video signals
  - Managing “scan-out”

Texas Instruments

Hardware: DDR to Display

Display Controller

- Video Pipes (Planes)

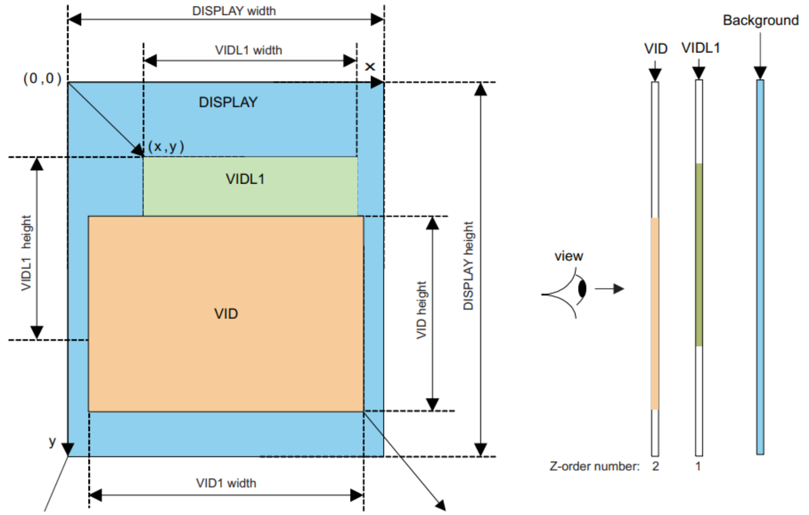


- Planes are framebuffers with attributes assigned.
- Video pipes take the pixels (one line at a time, unless a 2D-tiler is involved) and process them
- Perform all kinds of ops
  - Frame rotation
  - Scaling
  - Color conversions
- Saving Memory bandwidth

Hardware: DDR to Display

Display Controller

- Overlayers (CRTC)

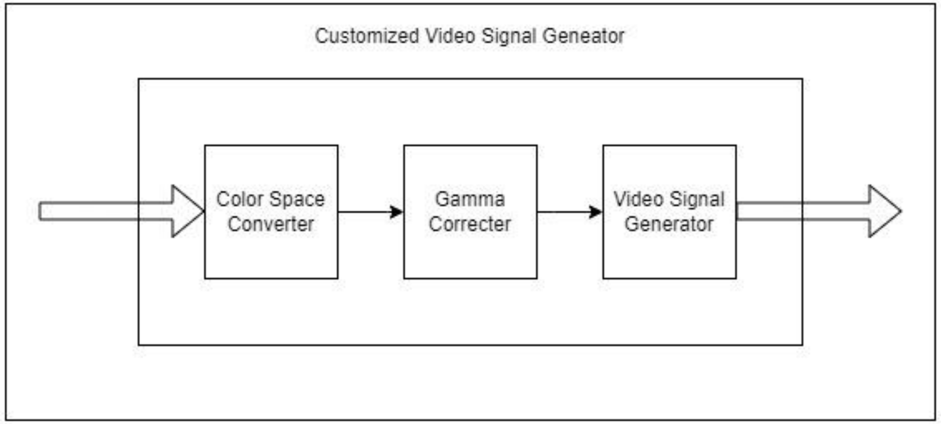


- Multiple planes layered on top of one another
- Displayed as a single frame in a single scan-out
- Z-positions

Hardware: DDR to Display

Display Controller

- Video Ports (CRTC + Encoder)



- This entity generates the DPI / OLDI signals that we saw earlier
- Other video ports are also capable to generating signals directly for other interfaces

Texas Instruments

Hardware: DDR to Display

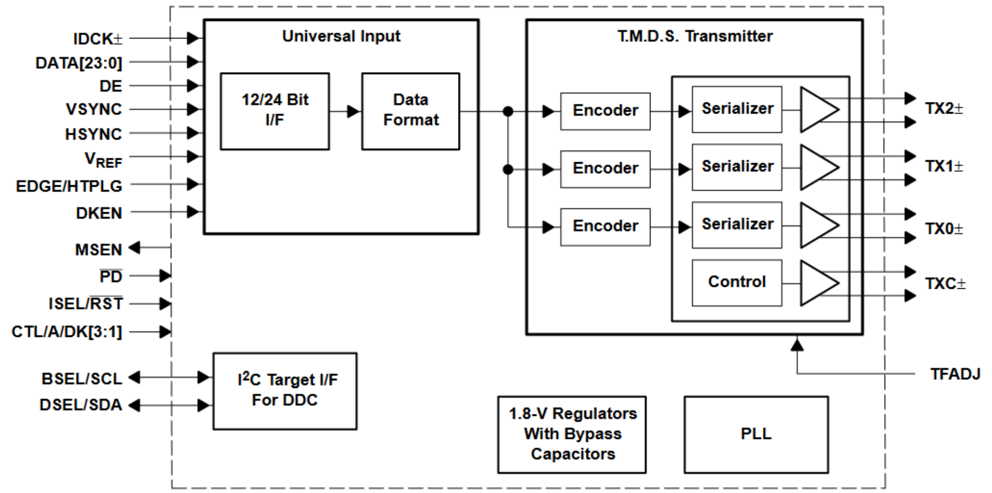
Display Controller: Summary

- Frames in memory to tangible and displayable video signals
- Bare-minimum stage
  - Can directly connect panels that consume these video signals and call it a day
- But, use-cases vary.
  - Not every use-case needs DPI or OLDI
  - Hence, Bridges.

Texas Instruments

Hardware: DDR to Display

Bridges



- How many times you have had to use “DP to HDMI” or “HDMI to VGA” cables!

- Also called the Transcoders
- Converts video signals from protocols to protocols / interfaces to interfaces
- Can be in-soc or external

Hardware: DDR to Display

Connectors / Always-connected Panels

- “Final” entities in the display pipeline
- Monitors get connected to systems via variety of connectors
  - HDMI
  - DP
  - VGA
- Often have an “internal” bridge that decodes the video signals to get pixel values.

Now What?

So, you have all the Hardware!

- As a SoM vendor, or an end-consumer, you have all the hardware.

- What needs to be done to get your display running?

Now What?

So, you have all the Hardware!

- As a SoM vendor, or an end-consumer, you have all the hardware. Custom platforms and Panels.

- What needs to be done to get your display running?

DEVICE-TREE!

Device-tree:

What can get defined in the DT:

- Memory
- Display-controller
- Bridge(s)
- Connector / Panel

Device-tree: Memory

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    /* global cma region */
    linux,cma {
        compatible = "shared-dma-pool";
        reusable;
        size = <0x00 0x8000000>;
        linux,cma-default;
    };

    [ ... ]
};
```

Device-tree: Display Media Graph

```
dss: display-controller@30200000 {
    compatible = "ti,am625-dss";
    [ ... ]

    ports {
        port@0 {
            reg = <0>;
            oldi_out0: endpoint {
                remote-endpoint = <&lcd_in0>;
            };
        };

        port@1 {
            reg = <1>;
            dpi1_out: endpoint {
                remote-endpoint = <&it66121_in>;
            };
        };
    };
};
```

```
lcd {
    compatible = "lincolntech,lcd185-101ct“, “panel-simple”;
    backlight = <&backlight>;
    port {
        lcd_in0: endpoint {
            remote-endpoint = <&oldi_out0>;
        };
    };
};
```

Device-tree: Display Media Graph

```
it66121: bridge-hdmi@4c {
    compatible = "ite,it66121";
    [ ... ]

    ports {
        port@0 {
            reg = <0>;
            it66121_in: endpoint {
                bus-width = <24>;
                remote-endpoint = <&dpi1_out>;
            };
        };

        port@1 {
            reg = <1>;
            it66121_out: endpoint {
                remote-endpoint = <&hdmi_connector_in>;
            };
        };
    };
};
```

```
hdmi0: connector-hdmi {
    compatible = "hdmi-connector";
    label = "hdmi";
    type = "a";
    port {
        hdmi_connector_in: endpoint {
            remote-endpoint = <&it66121_out>;
        };
    };
};
```

Texas Instruments

DRM Objects

Overview

- DRM Device
  - DRM planes
  - DRM CRTCs
  - DRM encoders
- DRM bridges
- DRM Connectors
- DRM Panels

Texas Instruments

DRM Objects

DRM: Modes / Mode Config

`struct drm_mode_config`

- **display-controller's generic information**
- parameters of a possible framebuffer `{min, max}_{width, height}`
- lists of planes, crtcs, encoders, connectors
- boilerplate and legacy properties like mutexes

`struct drm_display_mode`

- **timing parameters of the display (panel/monitor)**
- `{h,v}display, {h,v}sync, pixel-clock`

DRM Objects

DRM device

- `struct drm_device`
  - blanket structure to represent a display-controller unit.
  - keeps primary info about DCs
    - framebuffer
    - mode-config
    - power-states
    - scan-out details
    - Other boilerplate stuff
  - represented as **`/dev/dri/card`*`n`*** entity in user-space
  - apps query all the information through this card
  - Can contain multiple crtcs/planes/encoders

DRM Objects

DRM device: Planes

- `struct drm_plane`
  - possible crtcs
  - pixel format: RGB, YUV
  - plane type: primary, overlay, cursor
- `struct drm_plane_state`
  - associated crtc
  - associated framebuffer
  - rotation: 90/180/270
  - alpha, scaling params, z-position

- user-space dumps the frame in the framebuffer, sets the properties of the plane
- Plane helper functions:
  `struct drm_plane_helper_funcs`

```
atomic_check()
atomic_update()
atomic_disable()
```

Texas Instruments

DRM Objects

DRM device: CRTCs

- `struct drm_crtc`
  - Properties are legacy. `drm_crtc_state` used instead
    - plane type (primary, cursor)
    - display mode
- `struct drm_crtc_state`
  - display mode
  - vblank events
  - Records of state change
    - planes, connectors, modes

- Responsible for generating video stream, and maintaining the timings
  - Reporting v-blanks and Page-flips for userspace
    `drm_crtc_handle_vblank()`
    `drm_atomic_helper_page_flip()`
- Every crtc has exactly one primary plane, but can use overlay planes too
- CRTC helper functions:
  `struct drm_crtc_helper_funcs`

  `mode_valid()/mode_fixup`

  `atomic_enable/disable()`

Texas Instruments

DRM Objects

DRM device: Encoders

- `struct drm_encoder`
  - encoder_type:
    - DPI, DSI, LVDS, DP
  - possible_crtcs
  - list of bridge_chain

- intermediate between a crtc and a connector
- Encodes the pixel-stream by CRTC to any of the desired outputs.
  - Often kept as a soft entity as CRTCs and Bridges are used prevalently.
- does not have atomic states, and depends on the crtc and connector states

DRM Objects

DRM Bridges

- `struct drm_bridge`
  - ops (EDID, Hot Plug Detect)
  - connected encoder
  - connector_type:
    - DPI, DSI, LVDS, DP
  - timings
  - list of bridge_chain
- `struct drm_bridge_state`
  - input and output bus type

- Doesn’t come under DRM device or get represented in user-space
- But a key component to get the desired display on-going
- Bridge functions:
  `struct drm_bridge_funcs`

```
attach()/detach()
mode_valid()/mode_fixup
get_modes()
atomic_enable/disable()
atomic_get_input/output_bus_fmts()
```

Texas Instruments

DRM Objects

DRM Connectors

- `struct drm_connector`
  - type:
    - DPI, DSI, LVDS, DP
  - Status:
    - connected / disconnected / unknown
  - possible_encoders
  - list of modes
- `struct drm_connector_state`
  - associated crtc/encoder
  - basic properties

- Symbolic representation of the hardware as physical connectors do not require programming
- User-space tells a combination of connector, crtc and plane(s) to get a run a display
- Gets and conveys the modes
- Connector Helper functions:
  `struct drm_connector_helper_funcs`

  ```
  get_modes()
  detect()
  ```

DRM Objects

DRM Panels

- `struct drm_panel`
  - backlight
  - connector_type:
    - DPI, DSI, LVDS, DP

- Fixed panels seen on HMI devices like kiosks in ATMs, shopping markets, digital ad boards.
- Usually have a single mode, hardcoded in the driver or the device-tree
- Panel functions:
  `struct drm_panel_funcs`

```
get_modes/timings()
prepare/unprepare()
enable/disable()
```

DRM Objects

Generic Panel Driver: panel-simple

```
static const struct panel_desc
lincolntech_lcd185_101ct = {
    .modes = &lincolntech_lcd185_101ct_mode,
    .bpc = 8,
    .num_modes = 1,
    .size = {
        .width = 217,
        .height = 136,
    },
    .delay = {
        .prepare = 50,
        .disable = 50,
    },
    .bus_flags = DRM_BUS_FLAG_DE_HIGH,
    .bus_format = MEDIA_BUS_FMT_RGB888_1X7X4_SPWG,
    .connector_type = DRM_MODE_CONNECTOR_LVDS,
};
```

```
static const struct drm_display_mode
lincolntech_lcd185_101ct_mode = {
    .clock = 155127,
    .hdisplay = 1920,
    .hsync_start = 1920 + 128,
    .hsync_end = 1920 + 128 + 20,
    .htotal = 1920 + 128 + 20 + 12,
    .vdisplay = 1200,
    .vsync_start = 1200 + 19,
    .vsync_end = 1200 + 19 + 4,
    .vtotal = 1200 + 19 + 4 + 20,
};
```

```
static const struct of_device_id platform_of_match[] = {
    [ ... ]
    {
        .compatible = "lincolntech,lcd185-101ct",
        .data = &lincolntech_lcd185_101ct,
    },
    [ ... ]
};
```

Thank you!

References

- https://projectf.io/posts/video-timings-vga-720p-1080p/
- https://fastbitlab.com/display-interface-types/
- https://www.computerhope.com/jargon/h/hdmi.htm
- https://git.ti.com/cgit/ti-linux-kernel/ti-linux-kernel/?h=ti-linux-6.1.y-cicd
- https://git.kernel.org/pub/scm/linux/kernel/git/next/linux-next.git
- “A current overview of DRM KMS Driver Side API”, Talk in EOSS

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.
- Freedesktop.org

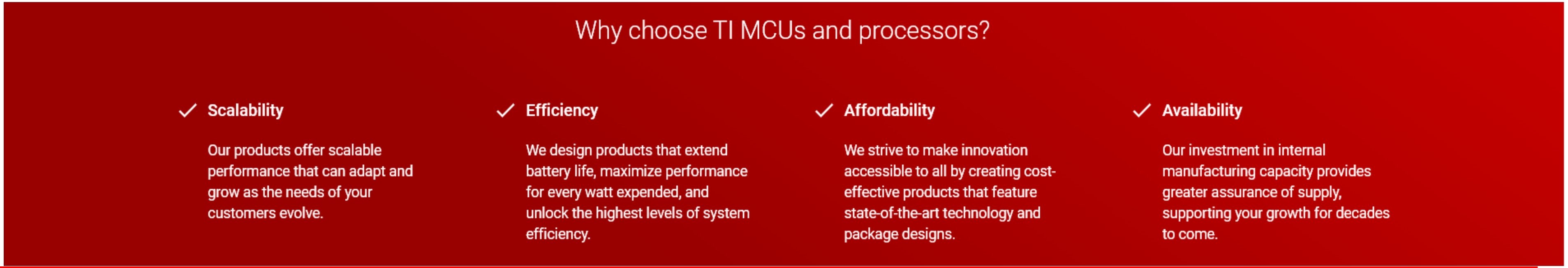
Q&A

- Contact Information:
  - Aradhya Bhatia <a-bhatia1@ti.com>
    - IRC @ aradhya7

- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- http://opensource.ti.com/
- https://www.ti.com/processors
- https://www.ti.com/edgeai



Why choose TI MCUs and processors?

✓ **Scalability**

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

✓ **Efficiency**

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

✓ **Affordability**

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

✓ **Availability**

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.