Optimizing Power Consumption in Embedded Linux: Techniques and Tradeoffs

2026-05-19

Kendall Willis, Texas Instruments

About us: TI Processors and Open source

Decades of contribution and collaboration

Ingrained culture to give back to the community

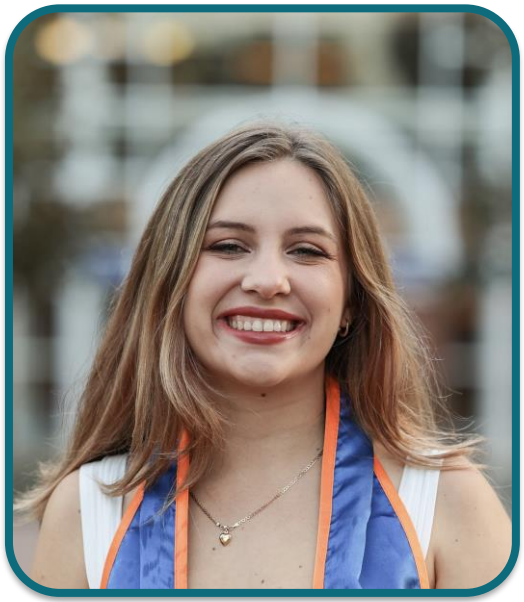
Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Introduction to the Speaker



Kendall Willis

Software Engineer at Texas Instruments

Dallas, TX, USA

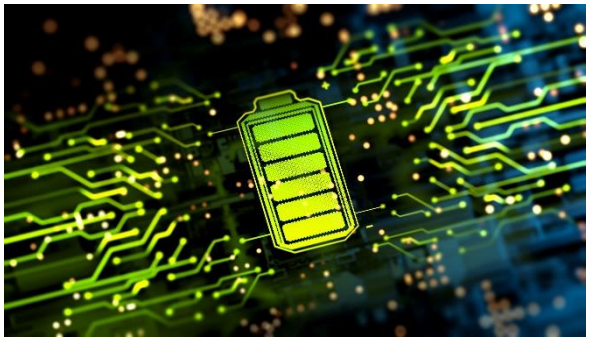
Linux Developer for Power Management on Sitara Embedded Processors

Overview

- Why does power matter on embedded devices?
- Tuning Knobs of Power Optimizations
- Linux Power Frameworks
- Case Study: AM62L Low Power HMI Application
- Optimizations
  - System
  - CPU
  - Display
  - DDR
- Conclusions

Why does power matter on embedded devices?

- Battery operated devices
- Cost savings
- Energy efficiency standards from governing bodies
  - USA: Code of Federal Regulations
  - EU: Standby, networked standby and Off mode - Energy Efficient Products



Tuning Knobs of Power Optimizations

Power ∝ Frequency * Voltage^2

Frequency Scaling

- Slow Clock Frequencies
- CPUFreq
- DevFreq
- Slow Display Refresh Rate
- Slow DDR Transfer Speed

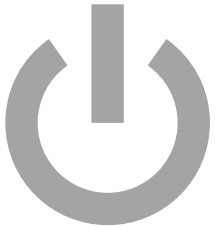
Idle States

- CPUIdle
- DDR Auto Self Refresh

Power Domains

- Runtime PM
- Device Tree Optimizations
- CPU Hotplug

Linux Power Frameworks



Static

Long idle periods

Ignorant to workloads

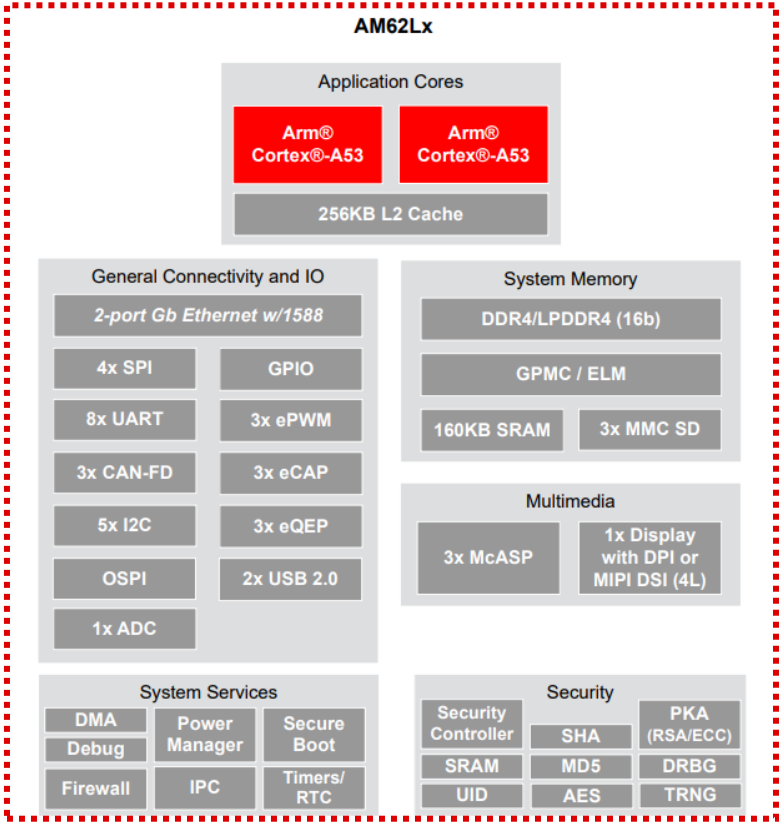
Dynamic

Short idle periods

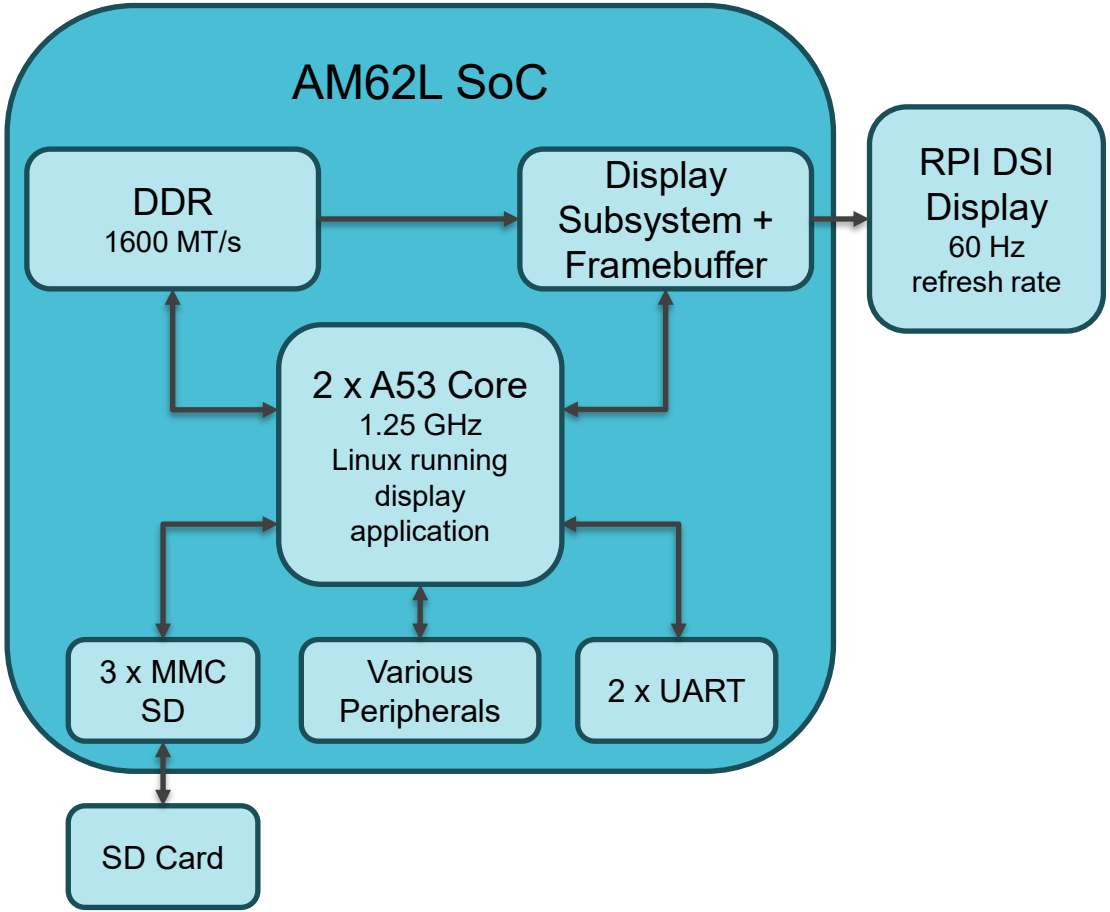
Workload aware

Case Study: AM62L Low Power HMI Application

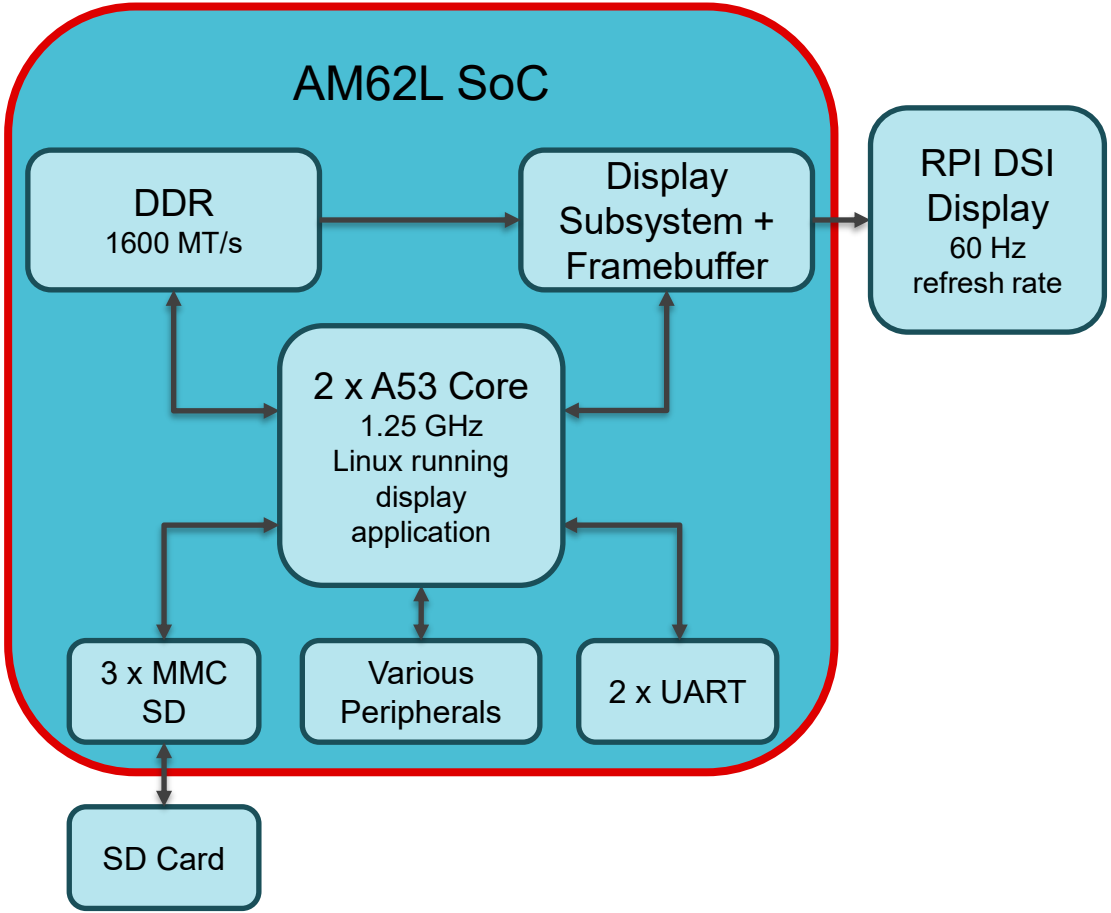
Goal	Low power consumption while running a display human-machine interface (HMI) application on the AM62L SoC
Constraint	Display is functional
Software	TI Processor SDK 12.0



Power Consumption:
448.18 mW



Power Consumption:
448.18 mW



System Suspend

Power Optimization	Framework
All	Static

- Suspend entire system to a low power state
- Several mode offerings with different levels of system power savings
  - Suspend to idle
  - Standby
  - Suspend to RAM
  - Hibernate
- More power savings = higher latency
- Low power states are platform specific

- Use if system allows for long idle times and all devices needed are still ON during suspend

Case study:

- Not used since currently there is no low power state on AM62L which allows the display subsystem (DSS) to stay ON

Runtime PM

Power Optimization	Framework
Power Domain	Dynamic

- Framework allows drivers to **independently** suspend their devices during runtime if they are not being used
- Automatic integration by device driver
- Consider removing a device from using runtime PM if a device is constantly switching runtime states or if the performance is negatively impacted

```
root@am62lxx-evm:~# cat /sys/devices/platform/bus@f0000/20000000.i2c/power/control
auto
root@am62lxx-evm:~# cat /sys/devices/platform/bus@f0000/20000000.i2c/power/runtime_status
active
root@am62lxx-evm:~# cat /sys/devices/platform/bus@f0000/20000000.i2c/power/runtime_suspended_time
10423
root@am62lxx-evm:~# cat /sys/devices/platform/bus@f0000/20000000.i2c/power/autosuspend_delay_ms
1000
```

Device Tree Optimization

Power Optimization	Framework
Power Domain	Static

- Remove any devices which are not needed for the application
- Disabling a device node in the device tree leaves the device in reset since the driver is not probed
- Nodes are enabled by default
- Make sure nodes that are disabled don't have interdependencies

Case Study:

- Removed all nodes that do not have to do with DSS or UART0
- **Power savings: 18mW**

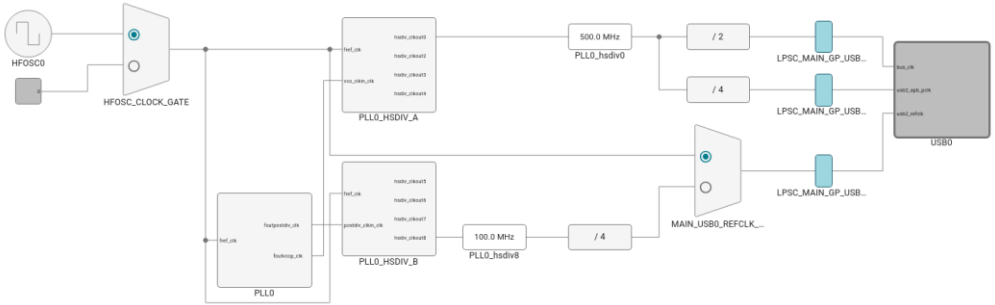
Disable status

```
&uart1 {
        status = "disabled";
};
```

Slow Clock Frequency

Power Optimization	Framework
Frequency Scaling	Static

- Reduce all clock frequencies for inactive devices
- Reduce clocks of active devices without affecting performance
- Enable auto clock gating for all peripherals
  - If not done by default
- Platform specific
  - Firmware may not expose clocks to userspace
  - Consult data sheet and block diagram for the platform to determine how clocks are used



Slow Clock Frequency

Power Optimization	Framework
Frequency Scaling	Static

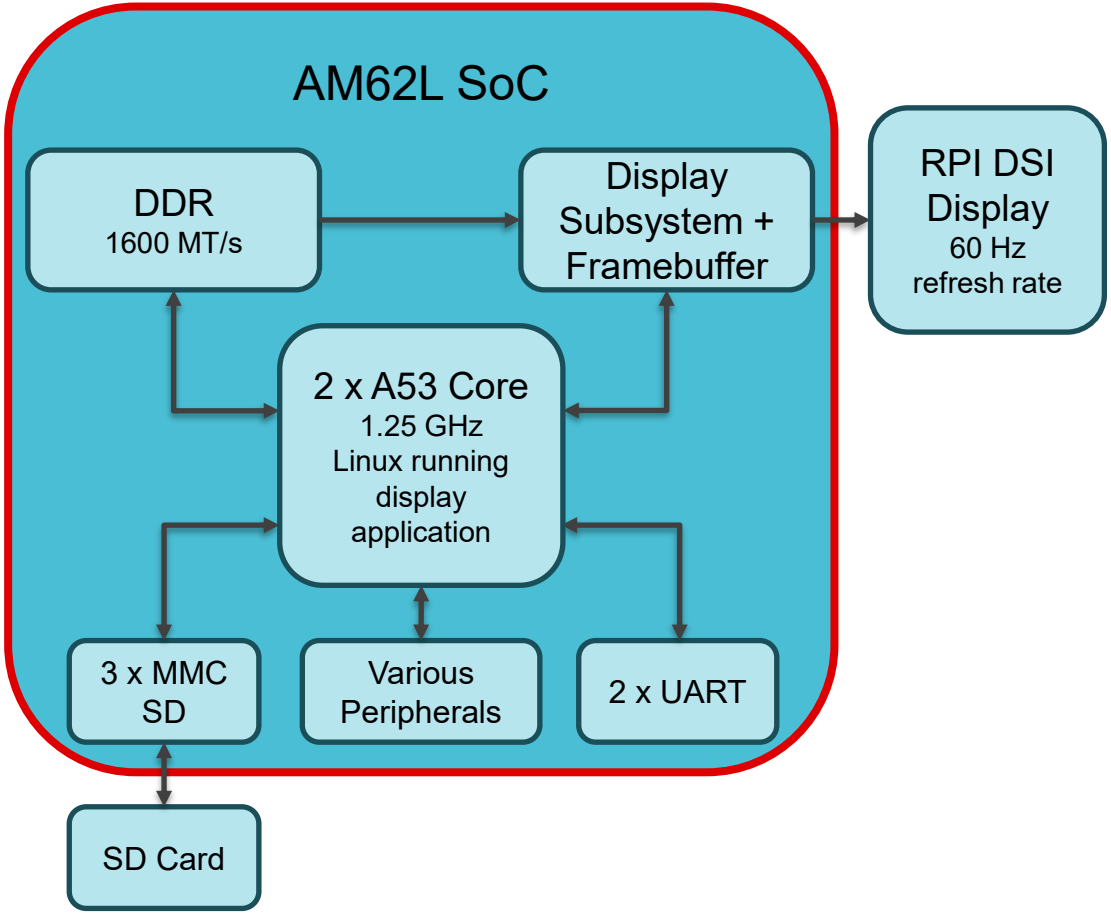
- ARM System Control and Management Interface (SCMI) clocks can be exposed to Linux
  - Sysfs: /sys/kernel/debug/clk/
- Platform specific tools
  - k3conf is a TI specific debug tool that allows the user to disable devices, disable clocks, and change clock frequencies from userspace

Case Study:

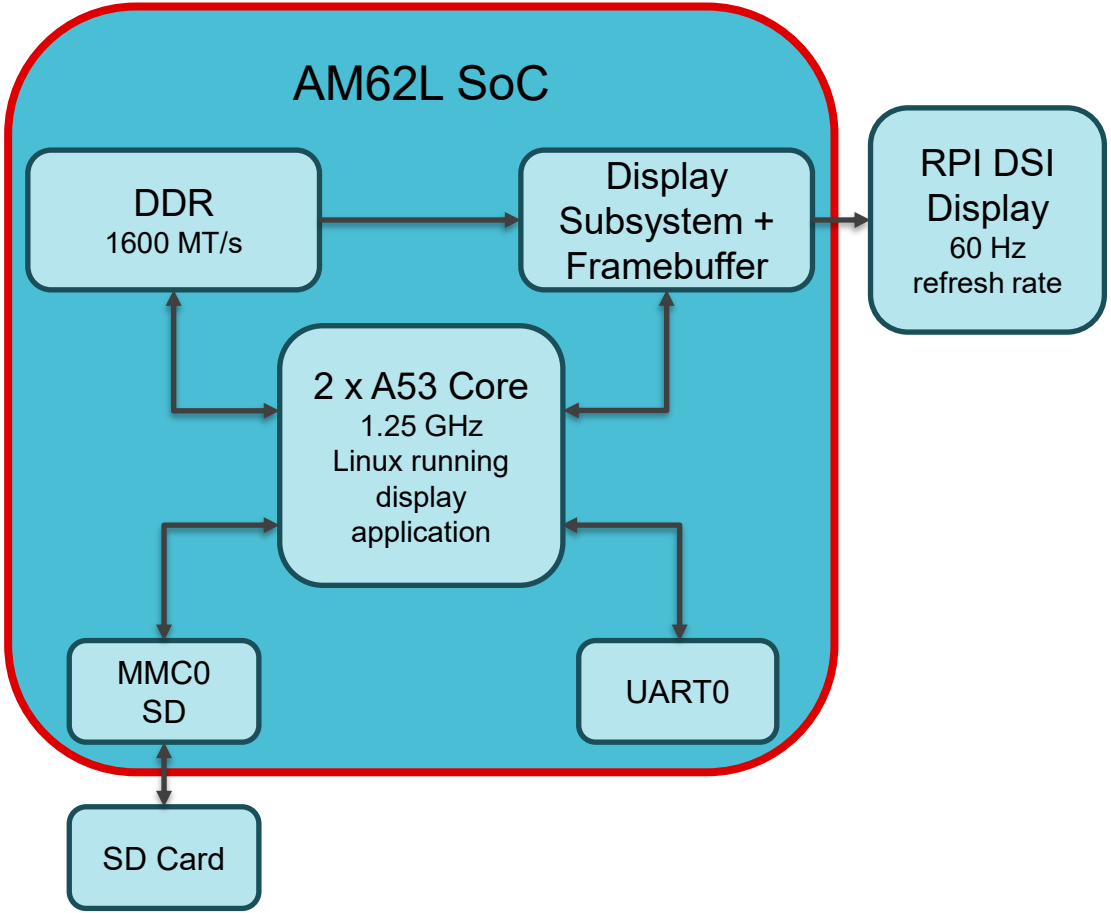
- Lowered frequency of all clocks in WKUP power domain
- **Power savings:** 56 mW

```
k3conf dump clock 0 4

*******************************************************************************

|-------------------------------------------------------------------------------------------|
| Device ID       | Clock ID | Clock Name              | Status          | Clock Frequency |
|-------------------------------------------------------------------------------------------|
| AM62LX_DEV_ADC0 |     4    | AM62LX_DEV_ADC0_ADC_CLK | CLK_STATE_READY | 25000000        |
|-------------------------------------------------------------------------------------------|
```

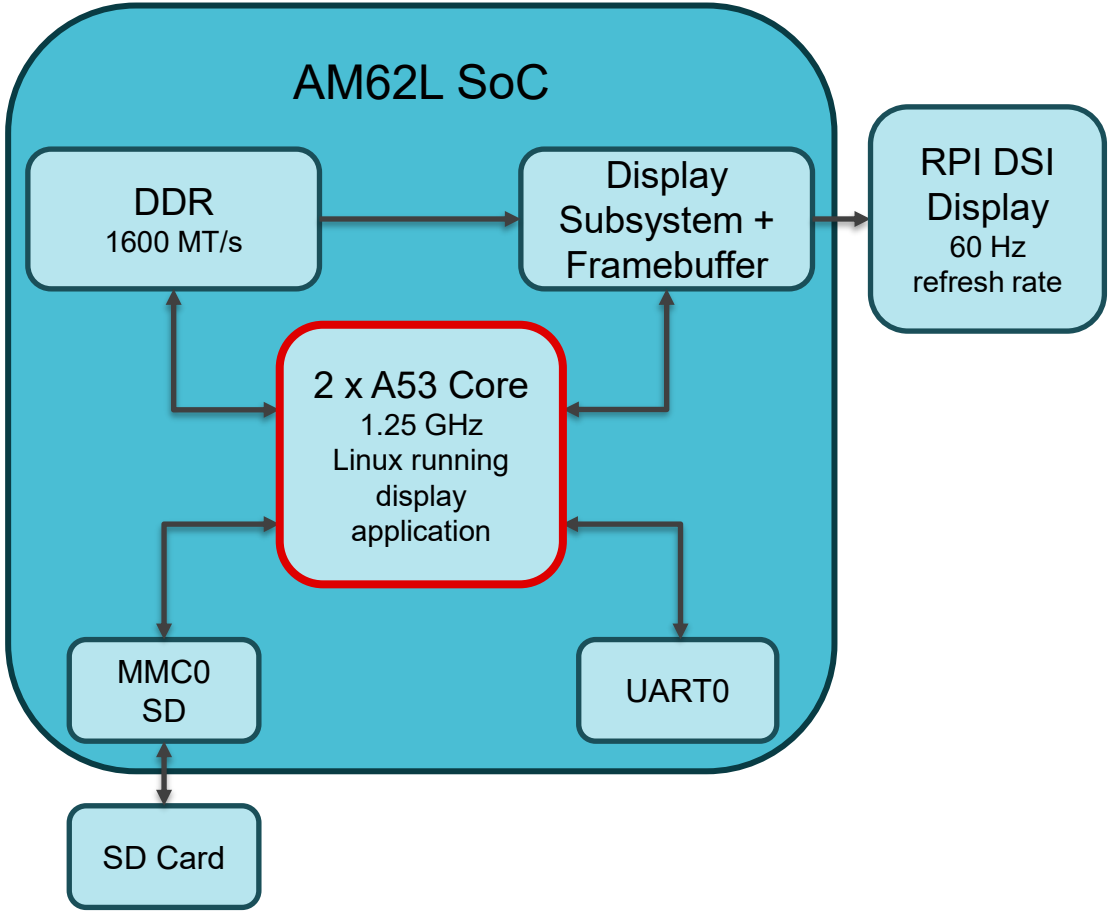
Power Consumption:
448.18 mW



Power Consumption:
373.72 mW



Power Consumption:
373.72 mW



CPUIdle

Power Optimization	Framework
Idle States	Dynamic

```
&{/}  {
        idle-states {
                entry-method = "psci";

                cpu_sleep_0: stby {
                        compatible = "arm,idle-state";
                        idle-state-name = "Standby";
                        arm,psci-suspend-param = <0x00000001>;
                        entry-latency-us = <25>;
                        exit-latency-us = <100>;
                        min-residency-us = <5000>;
                };
        };
};

&cpu0 {
        cpu-idle-states = <&cpu_sleep_0>;
};
```

- Framework that allows inactive CPUs to enter idle state
- Multiple platform defined states that each CPU can enter
- Governor determines state by predicting idle duration and the amount of latency each state has
- Small latency or residency times can increase power since CPU is entering and exiting idle states quickly
  - Iterative tuning to find right latency and residency times

CPUIdle

Power Optimization	Framework
Idle States	Dynamic

```
root@am62lxx-evm:~# ls /sys/devices/system/cpu/cpu0/cpuidle/
state0 state1
root@am62lxx-evm:~# ls -l /sys/devices/system/cpu/cpu0/cpuidle/state1/
desc latency name power time usage
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpuidle/state1/name
Standby
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpuidle/state1/usage
6245738
```

CPUIdle

Power Optimization	Framework
Idle States	Dynamic

Usecase	Any CPUs are expected to be idle for some amount of period
Tradeoff	Verify idle states do not negatively affect the system
Tools	Profiling tools – *perf, ftrace, turbostat*
Case Study	Not used - no power savings Low latency implementation due to display use case

CPU Hotplug

Power Optimization	Framework
Power Domains	Dynamic

- Don’t need a CPU? Turn it off!
- Turns on/off CPU during runtime
- If CPU is turned off, scheduler will not assign the CPU work
- Use if removing CPU doesn’t hinder performance

- Can be better to use only CPUIdle or CPU Hotplug
  - CPUs can reach longer idle times if the work is split up

Case Study:

- Turned off one of two CPU cores
- **Power Savings: Minimal**

```
root@am62lxx-evm:~# echo 0 > /sys/devices/system/cpu/cpu1/online
[ 2009.360852] psci: CPU1 killed (polled 0 ms)
root@am62lxx-evm:~# echo 1 > /sys/devices/system/cpu/cpu1/online
[ 2016.990260] Detected VIPT I-cache on CPU1
[ 2016.994362] GICv3: CPU1: found redistributor 1 region 0:0x0000000001860000
[ 2017.001328] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
```

CPUFreq & DevFreq

Power Optimization	Framework
Frequency Scaling	Dynamic

- Dynamic voltage and frequency scaling (DVFS)
  - CPUFreq for CPUs
  - DevFreq for devices
- Operating Performance Points (OPP) define the frequency and voltage points
- *opp-hz* is the frequency of the device
- *opp-microvolt* is the voltage of the device

```
opp-50-300000000 {
    /* OPP50 */
    opp-hz = /bits/ 64 <300000000>;
    opp-microvolt = <950000 931000 969000>;
    opp-supported-hw = <0x06 0x0010>;
    opp-suspend;
};

opp-100-275000000 {
    /* OPP100-1 */
    opp-hz = /bits/ 64 <275000000>;
    opp-microvolt = <1100000 1078000 1122000>;
    opp-supported-hw = <0x01 0x00FF>;
    opp-suspend;
};
```

CPUFreq & DevFreq

Power Optimization	Framework
Frequency Scaling	Dynamic

- Different governors that choose OPP based off on power/performance goals
  - Powersave governor prioritizes low frequency to save power
  - Ondemand governor uses CPU load to select frequency
  - Schedutil uses the CPU utilization data from the scheduler to select frequency
  - Performance governor prioritizes high frequency to get the best performance
  - Userspace governor allows frequency to be set in userspace
- Sysfs to control governor and view current frequency

```
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
ondemand userspace performance
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
200000 400000 600000 800000 1000000 1250000
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1250000
```

CPUFreq & DevFreq

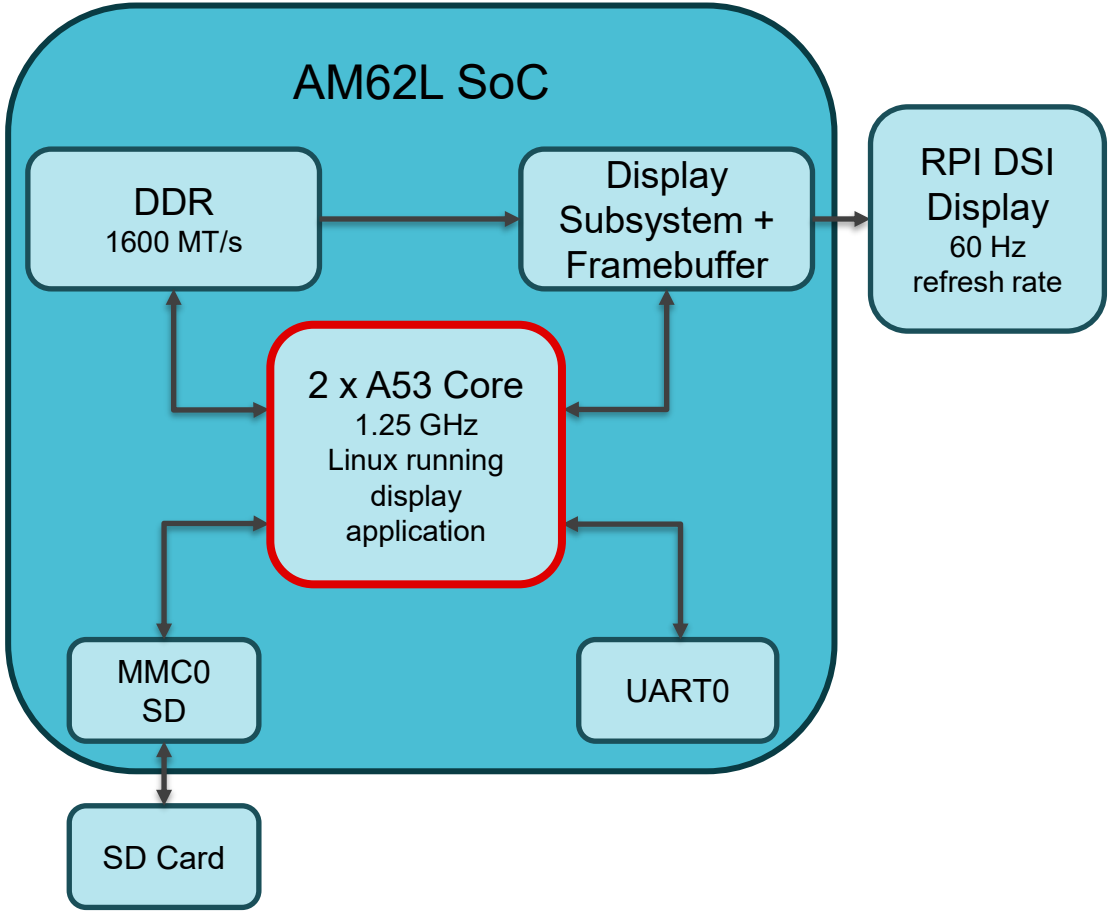
Power Optimization	Framework
Frequency Scaling	Dynamic

Case Study:

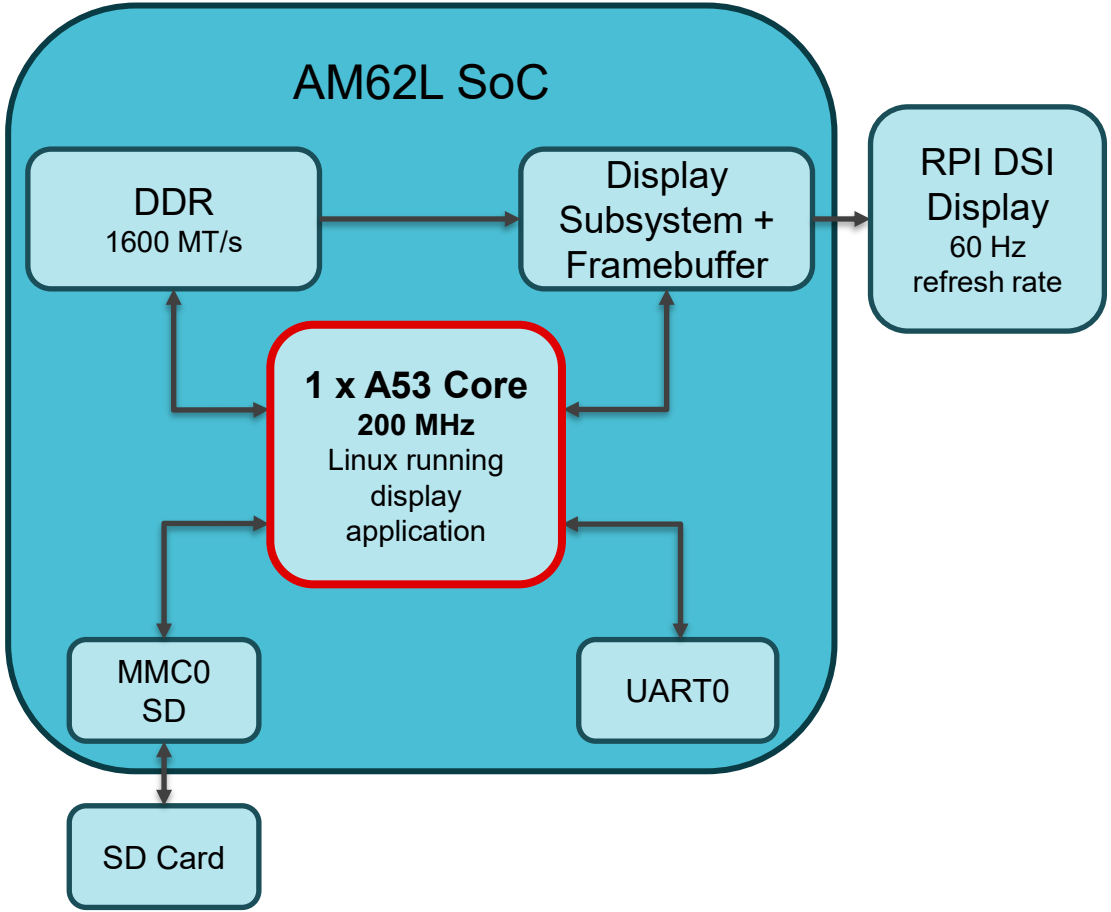
- No devices to available to use DevFreq with
- No voltage scaling available on AM62L
- Set CPUFreq to userspace governor with CPU running at 200MHz
- **Power savings: 8 mW**

```
root@am62lxx-evm:~# echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
userspace
root@am62lxx-evm:~# echo 200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
root@am62lxx-evm:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
200000
```

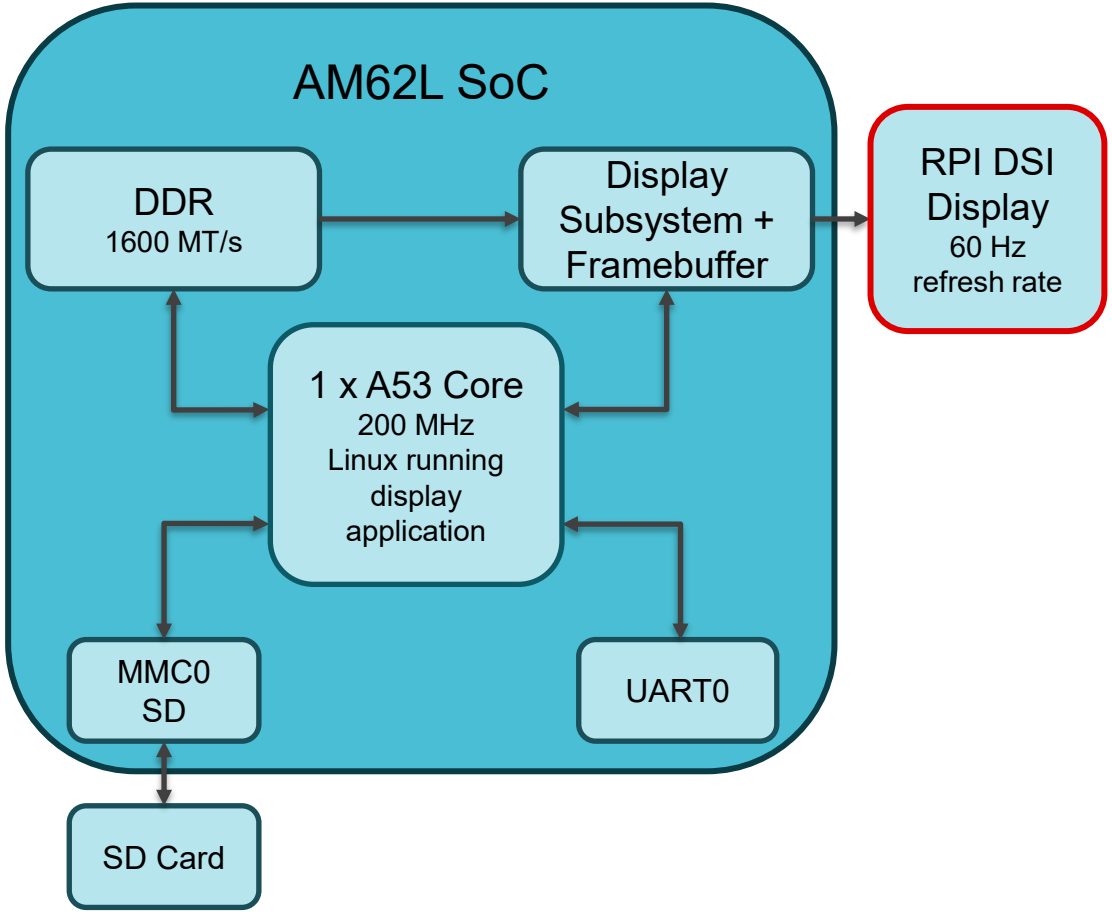
Power Consumption:
373.72 mW



Power Consumption:
364.94 mW



Power Consumption:
364.94 mW



Display Refresh Rate

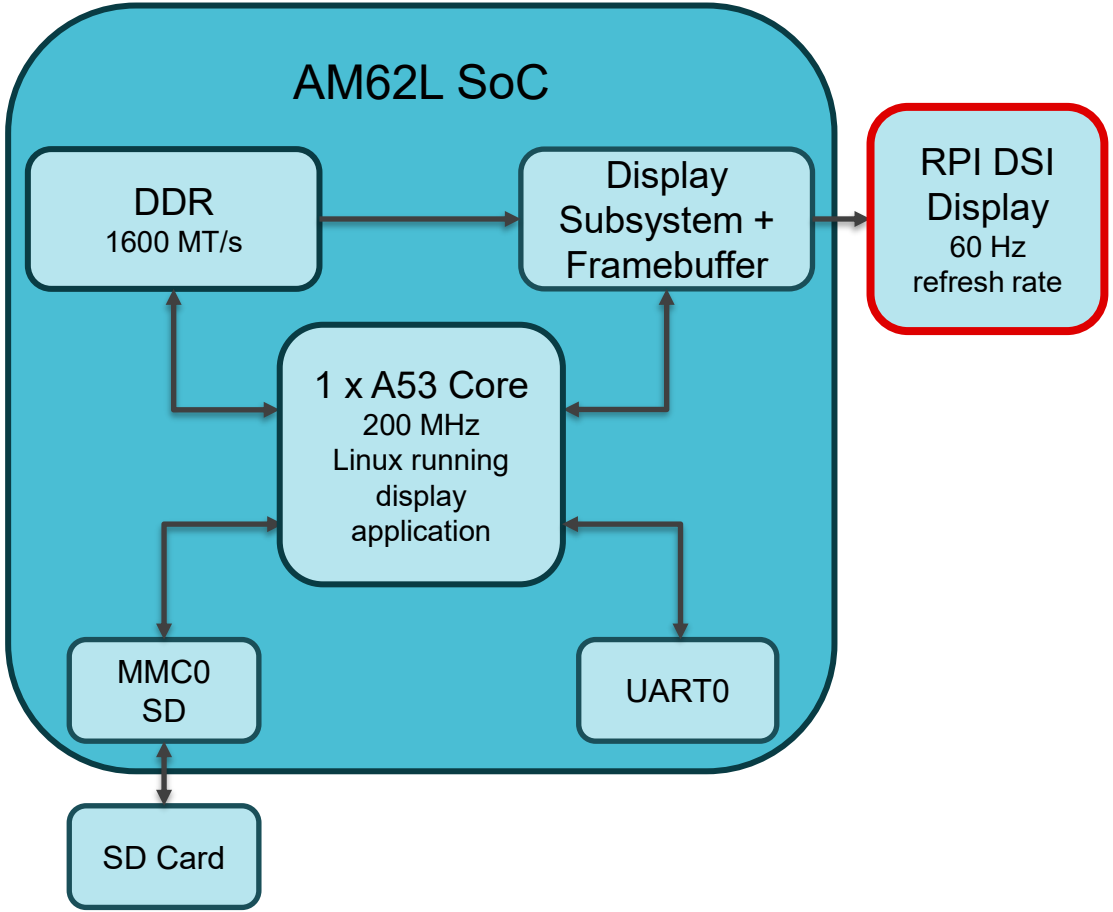
Power Optimization	Framework
Frequency Scaling	Static

- Refresh rate is how many times per second the image is updated on the display
- Higher refresh rates consume more power
- Use tools such as *modetest* to change the refresh rate
- Refresh Rate = Pixel Clock / (Horizontal Total × Vertical Total)

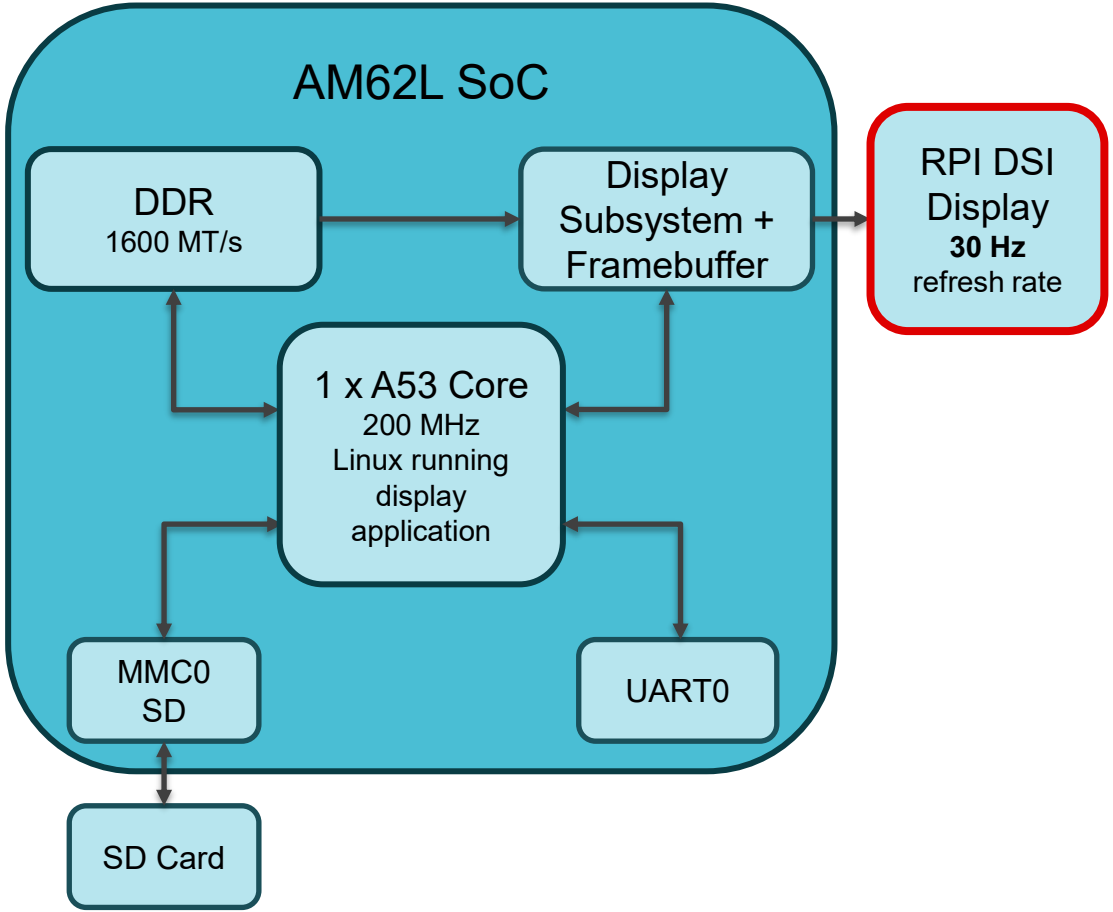
Case study

- Refresh rate of RPI DSI display is 60 Hz
- Change refresh rate from 60 Hz to 30 Hz by halving the pixel clock value
- Pixel clock is a static value in the *drivers/gpu/drm/panel/panel-simple.c* driver
- **Power savings**: **12 mW**

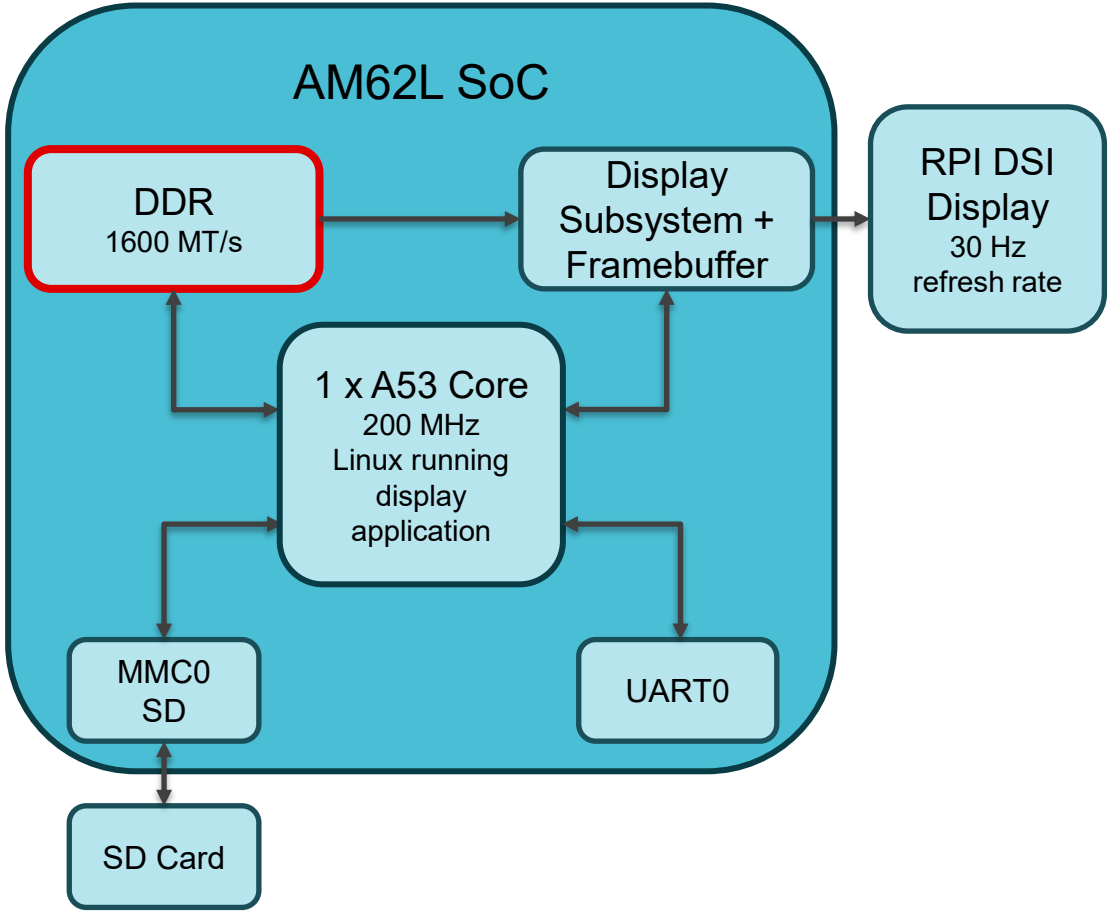
Power Consumption:
364.94 mW



Power Consumption:
353.04 mW



Power Consumption:
353.04 mW



DDR Transfer Speed

Power Optimization	Framework
Frequency Scaling	Static

- Change clock frequency for DDR which changes the MT/s
- Fewer transfers are happening over the same amount of time

Case Study:

- Clock frequency not exposed in userspace
- DDR clock frequency is set in Arm Trusted Firmware (TFA)
- 400 MHz to 200 MHz clock frequency
- 1600 MT/s to 800 MT/s
- **Power Savings: 50 mW**

```
#define DDRSS_PLL_FHS_CNT 3
#define DDRSS_PLL_FREQUENCY_1 200000000
#define DDRSS_PLL_FREQUENCY_2 200000000
#define DDRSS_SDRAM_IDX 15
#define DDRSS_REGION_IDX 15
```

DDR Auto Self Refresh

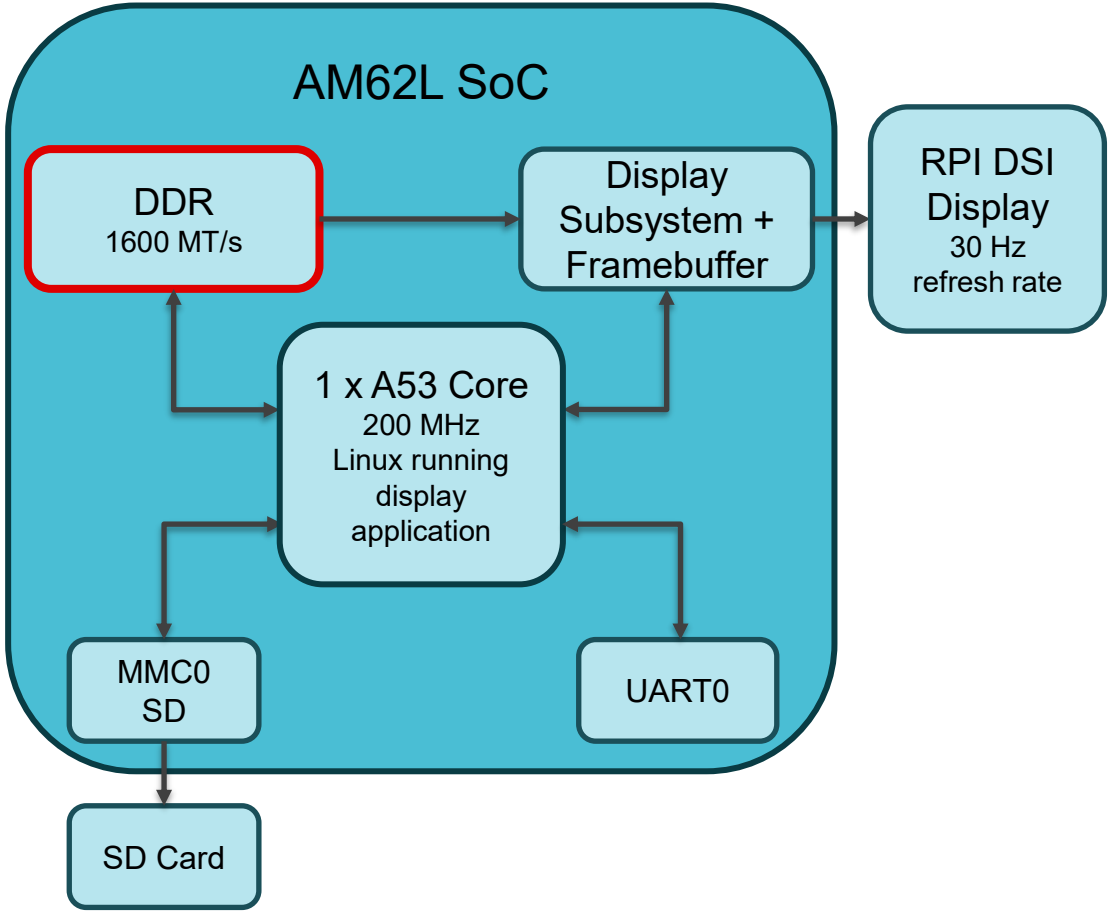
Power Optimization	Framework
Idle States	Dynamic

- DDR self refresh is a low power state that enables the DDR to refresh contents without using an external clock
- DDR auto self refresh allows the DDR controller to put DDR into self refresh after some idle time and DDR will leave idle when a transaction occurs

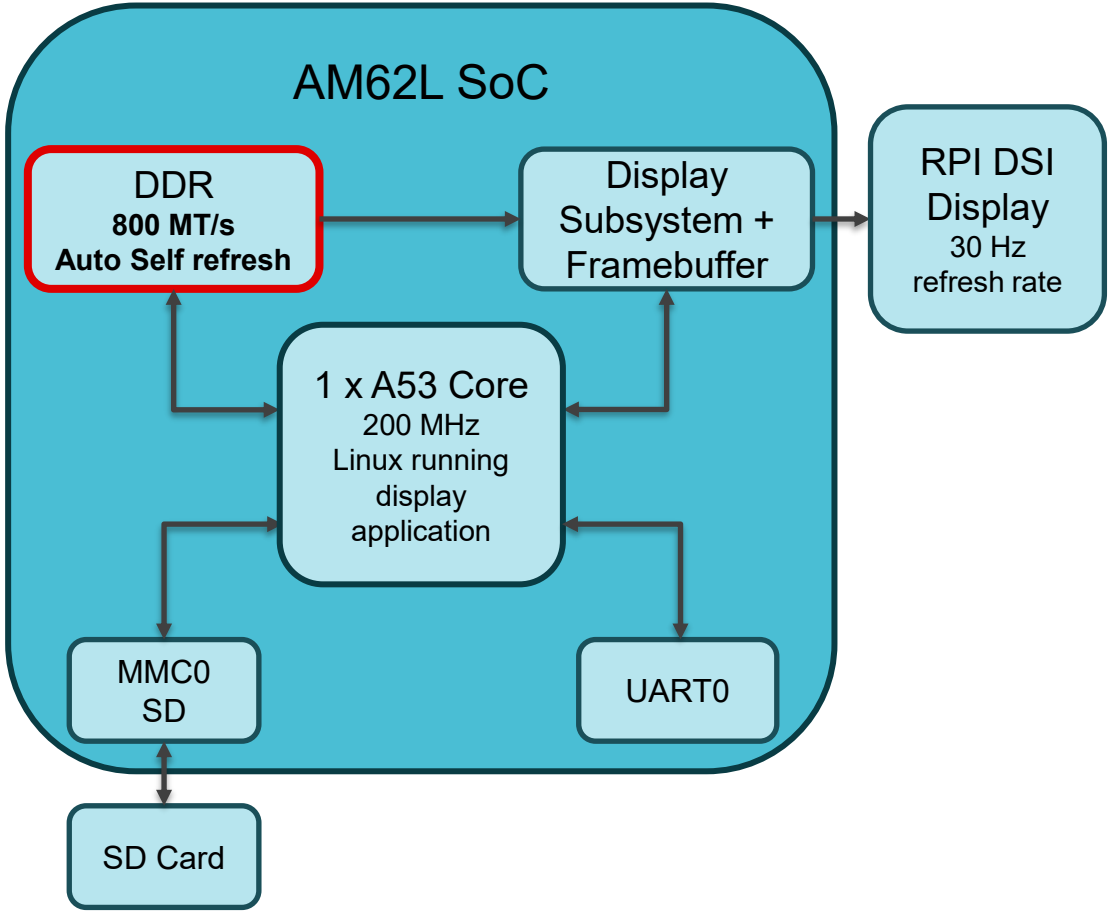
Case study:

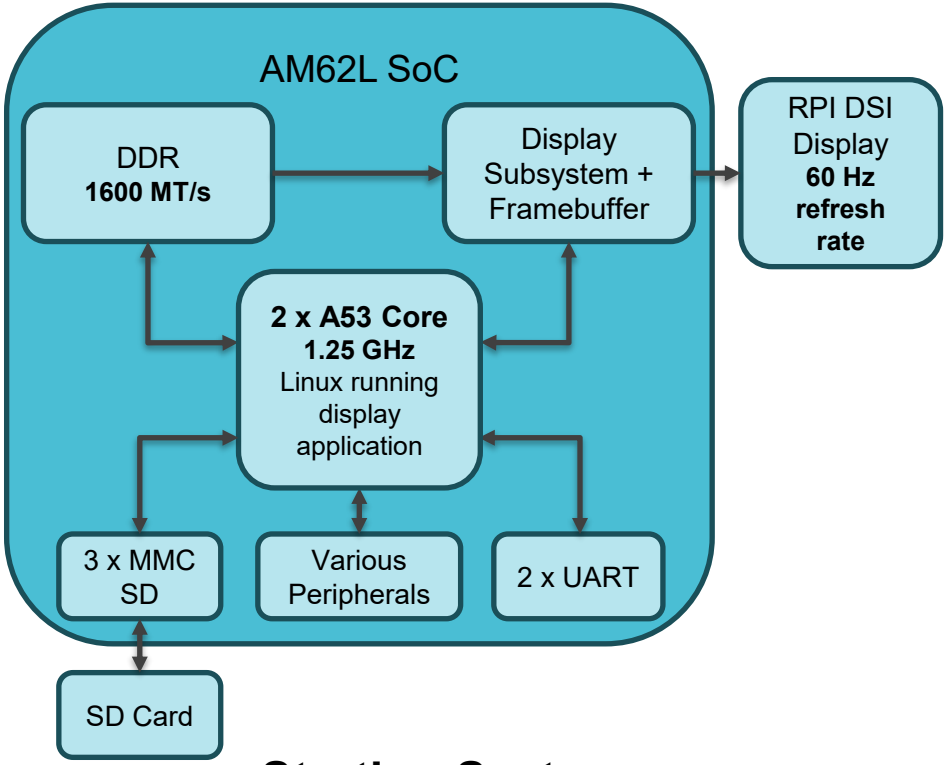
- Executed through direct register writes to the DDR controller
- Values are found in the AM62L Technical Reference Manual
- **Power savings: 45 mW**

Power Consumption:
353.04 mW

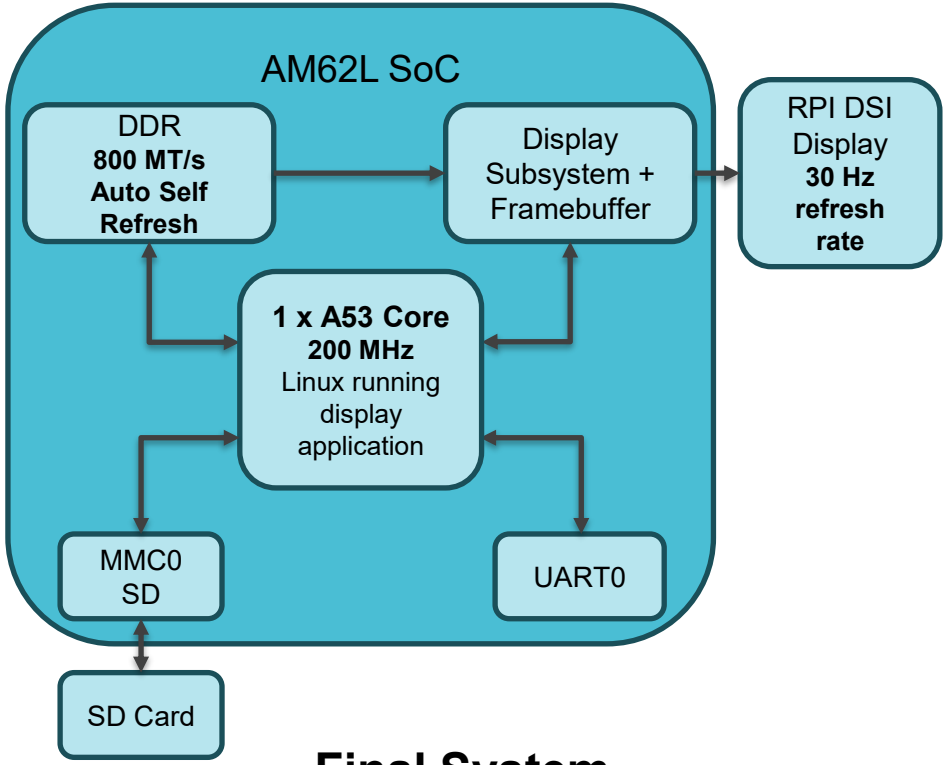


Power Consumption:
256.96 mW



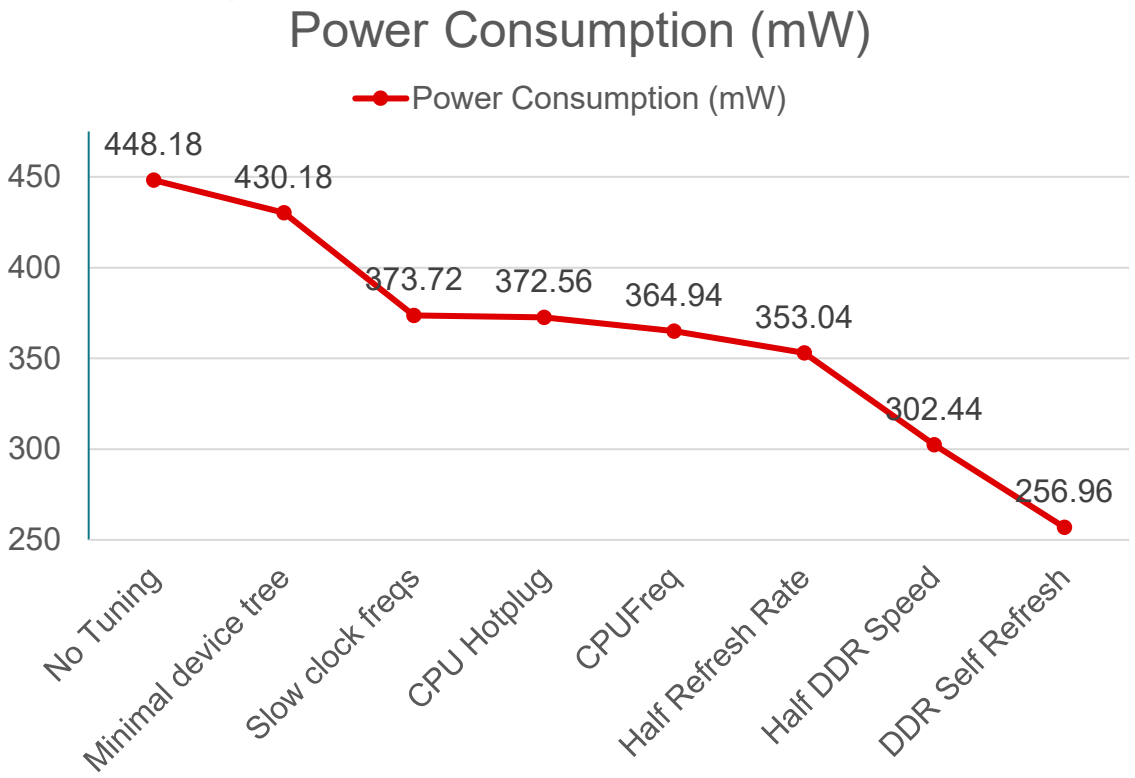


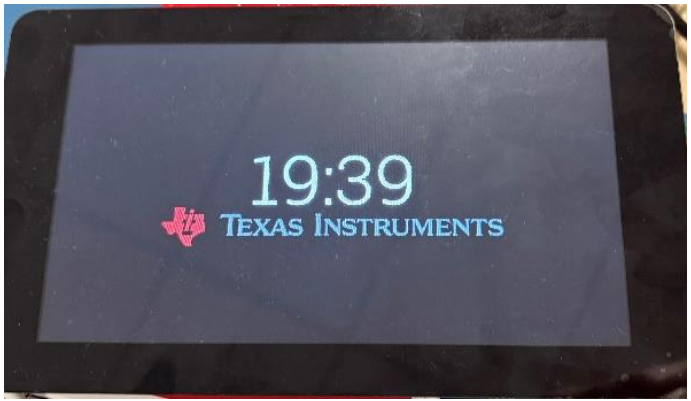
Starting System Power Consumption:
448.18 mW



Final System Power Consumption:
256.96 mW

Final System



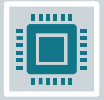


*Each techniques power savings builds off the previous power savings.

Conclusions

Iterative approach

Start with whole system optimizations, then component specific



Examine how power optimizations impact performance

Consider how individual components interact with the system

References

- Runtime Power Management Framework for I/O Devices — The Linux Kernel documentation
- CPU Idle Time Management — The Linux Kernel documentation
- CPU hotplug in the Kernel — The Linux Kernel documentation
- CPU Performance Scaling — The Linux Kernel documentation
- AM62L Technical Reference Manual
- 1. Overview — Linux SDK for AM62L Documentation
- AM335x Low Power Design Guide (Rev. A)

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.
- Anshu Madwesh
- Dhruva Gole
- Scaria Kochidanadu

Q&A

- Contact Information:
  - Kendall Willis k-willis@ti.com
- LinkedIn

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- AM62L EVM

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.