Application note

STM32MP13x product lines system power consumption

Introduction

STM32MP13x product line devices are built on an Arm® Cortex®-A7 with single core MPU subsystem.

STM32MP13x product line datasheets present the devices power consumption values calculated using bare metal software (not using Linux operating system). The values are available for basic Run modes and for various low-power modes such as CSleep, CStop, Stop, LP-Stop, LPLV-Stop, LPLV-Stop2 and Standby well described in the RM0475.

This application note provides power consumption values on various use cases measured on a STM32MP135F device mounted on a STM32MP135F-DK discovery board and running on STM32 MPU OpenSTLinux Distribution.

The values provided in this document are indicative typical values measured on one sample at room temperature. User may measure different values depending on device characteristics (slow, typical, fast) and ambient temperature.

For further information on STM32MP13x product line devices, refer to the following documents and deliverables available on *www.st.com*:

- STM32MP13x product line reference manual (RM0475)
- STM32MP13x product line datasheets
- STPMIC1x datasheet
- *STM32MP13x lines using low-power modes* application note (AN5565)
- *Getting started with STM32MP13x lines hardware development* application note (AN5474)
- *STM32MP13x MPU line and STPMIC1D / STPMIC1A integration on a wall adapter supply* application note (AN5587)
- STM32CubeMP1
- STM32MP13x product line embedded software
- STM32MP1 wiki article: https://wiki.st.com/stm32mpu/wiki/Category:Getting_started_with_STM32MP1_boards

1 General information

This document applies to STM32MP13x product line microcontrollers Arm®Cortex®-based devices.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

2 Overview

This document is applicable to all the devices of the STM32MP13x product lines. The table below describes the main characteristics of all products belonging to STM32MP13x product lines.

The present document assumes a full featured device (for example STM32MP135F).

Table 1. Configurations of the STM32MP13x product line devices

Product line	Reference manual	TFT	DCMIPP	FDCAN
STM32MP131	RM0475	No	No	No
STM32MP133		No	No	Yes
STM32MP135		Yes	Yes	Yes

3 STM32MP135F-DK board

The STM32MP135F-DK board is identified as "Discovery board" (MB1635B).

Detailed description of the board can be found in the *Getting started with STM32MP1 boards* wiki web page *https://wiki.st.com/stm32mpu/wiki/Category:Getting_started_with_STM32MP1_boards* on the article “STM32MP13 boards”.

3.1 Measurement points

On the STM32MP135F-DK board one cannot directly measure the consumption of the various voltage domains: V_{DD}, V_{DD_CORE}, V_{DD_DDR}, V_{DD_CPU}.

Figure 1. Board with cables

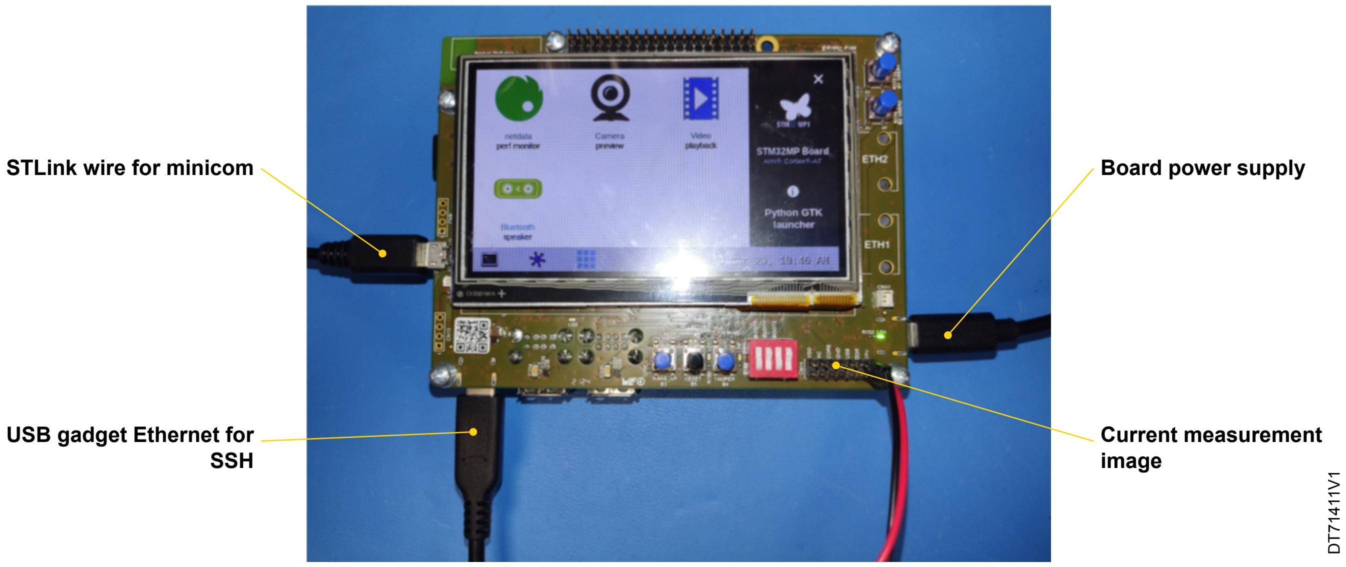
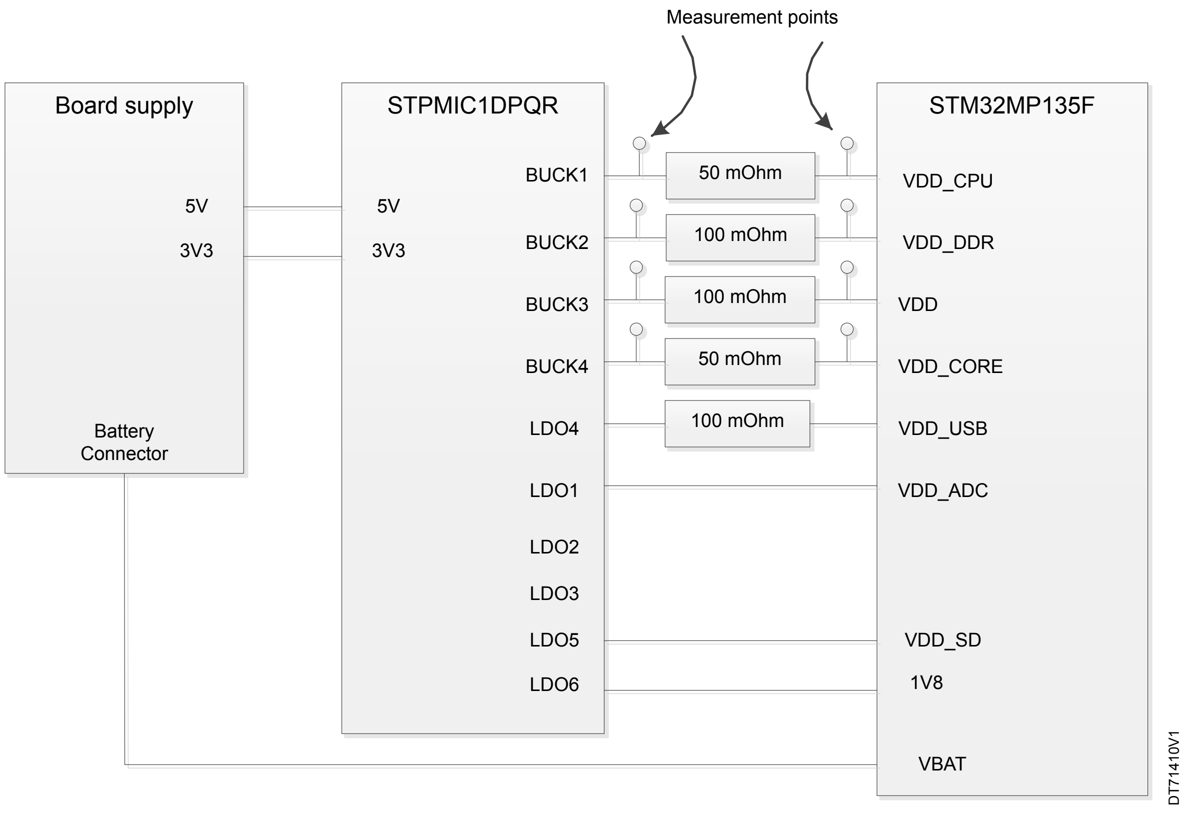
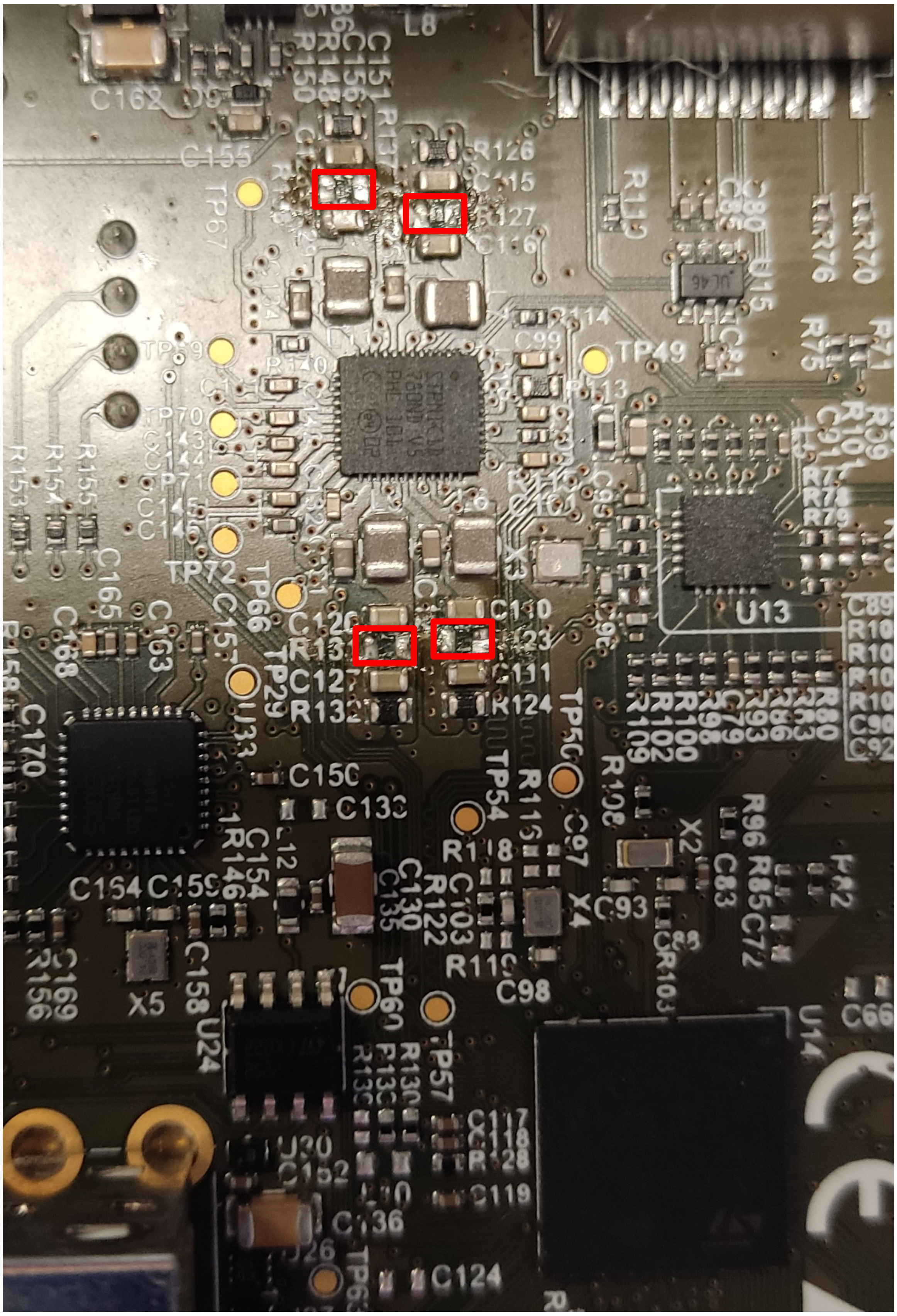


Figure 2. **Power control block diagram**



In order to perform the measurements provided in this document, the MB1635B board must be modified. The following 0 Ω shunt resistors: R127, R138, R123, R131 present on the back side of the board must be removed, see Figure 3 to locate their position:

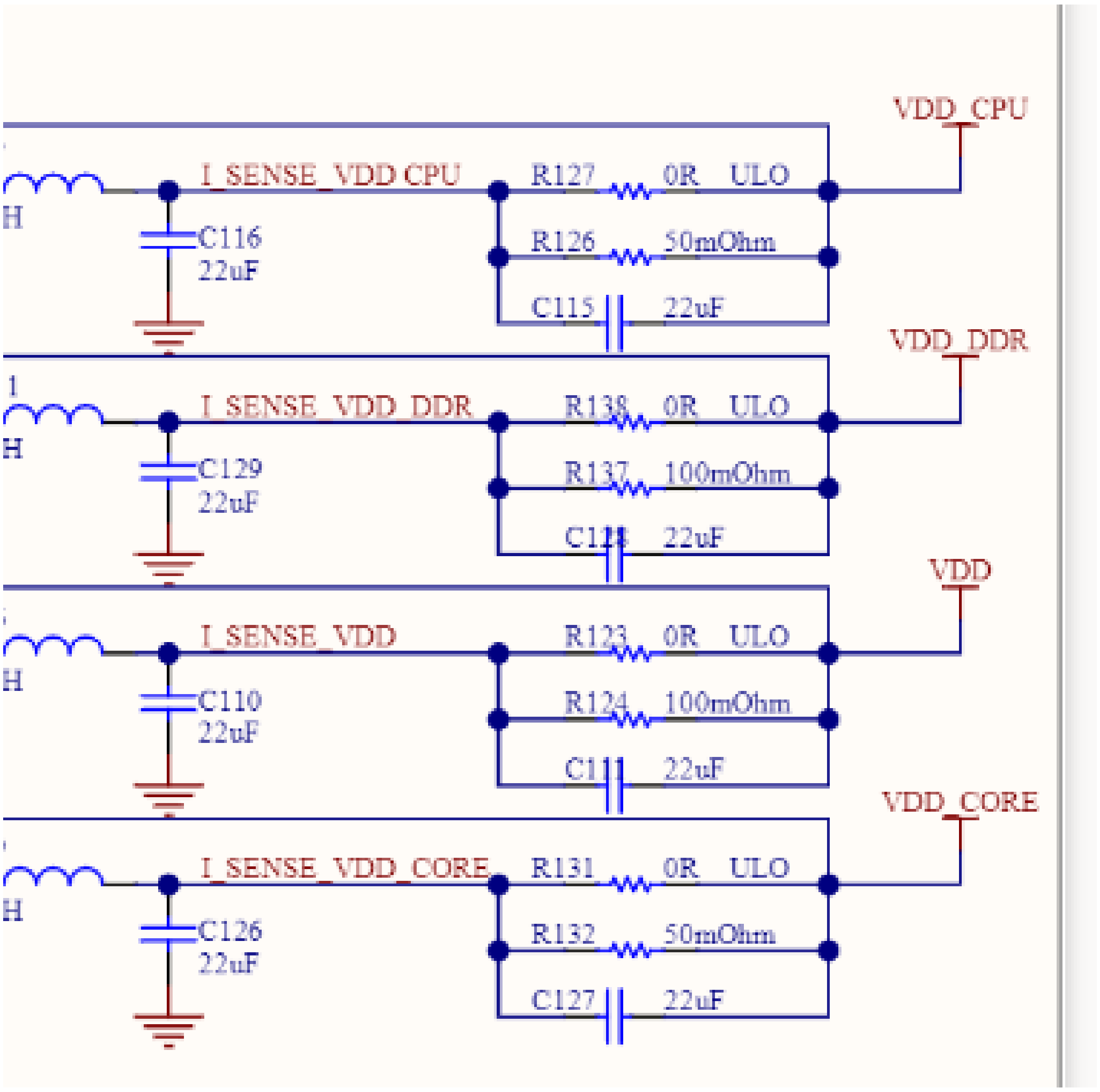
Figure 3. Resistors to remove for current measurement



DT71412V1

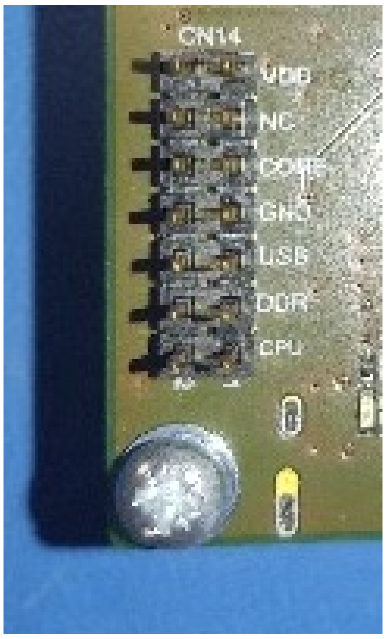
Once the shunt resistors are removed, the V_{DD_CORE}, V_{DD}, V_{DD_DDR} and V_{DD_CPU} supply paths include a serial 0.1 Ω or 0.05 Ohm resistor as shown below:

Figure 4. Power supply lines



The voltage drop across the power supply line serial resistors can be measured using the test points illustrated in the figure below and the current consumption can be deduced.

Figure 5. **Measurement points on the board**



3.2 STM32MP1 OpenSTLinux Distribution

Unless otherwise specified all measurements have been done using the STM32MP1 OpenSTLinux Distribution starter package.

ST OpenSTLinux - Weston - (A Yocto Project Based Distro) 4.0.1-openstlinux-5.15-yocto-kirkstone-mp1-v22.06.15 version.

4 Power consumption on STM32MP13x product lines

This section provides some use cases of power consumption measurements using the STM32MP135F device mounted on a STM32MP135F-DK board.

4.1 CoreMark® on Linux MPU

First step: prepare CoreMark® executable

The prerequisite is to have the STM32MP1 developer package SDK.
The following commands must be run on the Linux station:

1. Download CoreMark® from EEMBC (embedded microprocessor benchmark consortium)

```
>> mkdir ~/Coremark
>> cd ~/Coremark
>> git clone https://github.com/eembc/coremark.git
```

2. Create specific STM32MP13 folder

```
>> mkdir coremark/STM32MP13_linux
>> cp coremark/linux/* coremark/STM32MP13_linux
>> mkdir coremark/STM32MP1_linux/posix
>> cp coremark/posix/* coremark/STM32MP13_linux/posix
```

3. Update *coremark/STM32MP13_linux/posix/core_portme.mak*
Comment the lines below:

```
#CC? = cc
#LFLAGS_END += -lrt
```

Update the lines with IP@ for Ethernet connection:

```
LOAD = scp $(OUTFILE) root@<IP@>:/home/root/
#RUN = ssh root@<IP@> -c /home/root/
```

Comment the lines below:

```
#LOAD = echo Loading done
#RUN =
```

Note: <IP@> can be found running ifconfig command on the target.

4. Update the makefile

```
>> cd ~/Coremark/coremark
>> gedit Makefile &
```

Remove space between LOAD and OUTFILE in Makefile

```
>> $(LOAD)$(OUTFILE)
```

5. Use the source of the developer package to compile for the STM32MP13 (the path depends on where the sources are installed)

```
>> source ~/Developer-Package/SDK/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

6. Compile CoreMark® and make coremark.exe

```
>> cd ~/Coremark/coremark
>> make PORT_DIR=STM32MP13_linux
```

7. Make *.exe executable if not already the case

```
>> chmod +x coremark.exe coremark_2core.exe
```

8. Copy CoreMark® executable onto the MB1635B SDCard:
Copy file to the target by using scp:

```
>> scp coremark.exe root@<IP@>:/usr/local/
```

Note: Administrator rights are needed on the machine in order to run 'sudo'.

Second step: power measurements

To run CoreMark® on the board run the commands below:

```
>> cd /usr/local
>> ./coremark.exe 0x0 0x0 0x66 0 7 1 2000 > run.log
```

The table below shows the consumption measurements done on CoreMark®.

Table 2. **CoreMark® power consumption on STM32MP135F + MB1635B**

-	Power V_{DD_CPU} (mW)	Result Iteration per second
CoreMark® on MPU core (650 MHz) V_{DD_CPU} = 1.25 V	125	1793
CoreMark® on MPU core (1000 MHz) V_{DD_CPU} = 1.35 V	226	2807

4.2 Graphical use case: 480P video playback

The graphical use case is built with the following settings:

- 480P video playback @650 MHz and 1000 MHz, Linux
- voltages: V_{DD_CPU} = 1.25 V or 1.35 V, V_{DD_CORE} = 1.25 V, V_{DD} = 3.3 V, V_{DD_DDR} = 1.35 V
- Ethernet network

Video file details:

- Video:
  - H264 codec (H264 baseline profile version 1.3)
  - 640×480 pixels
  - 15 fps
  - 1500 kbps bitrate
- Audio: (the video is played without audio: --audiosink = fakesink)
  - AAC codec
  - 44 kHz stereo
  - 128 kbps variable bitrate

About audio and synchronisation settings:

The video is played without audio (--audiosink = fakesink) and without display synchro (sync = false) to reach maximum CPU load and frame rate.

About Ethernet use case:

It is not a real "streaming" use case but rather a "file system read through Ethernet" use case. A real "streaming" use case would have more incidence on the power due to network accesses.

4.2.1 Video file transcoding

The below information helps transcoding the 1080p ST2297 visionv3 ST video to a new video file with the same configuration as provided in Section 4.2 introduction.

1/ Prepare the video file:

```
>> export audio="-c:a aac -ar 44100 -ac 2 -ab 128k" # AAC codec 44 Khz stereo 128 Kbps
variable bitrate
>> export video="-c:v libx264 -profile:v baseline -vf scale=640:480,fps=15 -crf 22" # H264
640x480 pixels baseline 15 fps, ~1500k bitrate (thanks to -crf 22 -preset slow)
>> export extra="-y -tune fastdecode -preset slow" # force overwrite, fastdecode filter,
slower preset for better quality vs bitrate
>> export input_file="ST2297_visionv3_1080p.mp4"
>> export output_file="ST2297_visionv3_480p_15fps_1500kbps.mp4" ffmpeg -i $input_file $video
$audio $extra $output_file
```

2/ Check video size and content

```
>> ls -al $output_file

Response:
-rw-r--r-- 1 frq08942 ggfrqlmeang1-ps 40753417 Jun 20 12:20
ST2297_visionv3_480p_15fps_1500kbps.mp4

>> ffprobe $output_file # or gst-discoverer-1.0 -v $output_file

Response:
Duration: 00:04:02.39, start: 0.000000, bitrate: 1345 kb/s
Stream #0:0(eng): Video: h264 (Constrained Baseline) (avc1 / 0x31637661), yuv420p, 640x480
[SAR 4:3 DAR 16:9], 1209 kb/s, 15 fps, 15 tbr, 15360 tbn, 30 tbc (default)
Stream #0:1(eng): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 132 kb/s
(default)
```

The bitrate varies around 1500kbps.

The video can be downloaded on:

- Weston: *tmp_data/PCO/videos/ST2297_visionv3_480p_15fps_1500kbps.mp4*
- Linux: */users/tmp_data or smb://work01.lme.st.com/tmp_data*
- Windows: *\\work01.lme.st.com\tmp_data*

4.2.2 Preparation and various checks

Here below is a set of commands helping to prepare the various measurements. Only a subset of the commands is required for the power measurements, the other commands help checking the overall configuration.

Start the STM32MP135F-DK with a recent software (dv4)

Enter the following command to disable weston and reboot the board (the board reboots without weston)

```
>> systemctl disable weston-launch
>> reboot
```

Note: *Read for more details.*

By default the MPU clock is adaptive (depending on the bandwidth required by the application).

In order to force the MPU clock as well as the V_{DD_CPU} to one of the two schemes:

- 1.25 V @650 MHz
- 1.35 V @1000 MHz

run the following commands (650 MHz scheme illustrated, replace 650.000 by 1.0000.000 for the second scheme)

```
>> cd /sys/devices/system/cpu/cpu0/cpufreq/
>> echo 650000 > scaling_min_freq
>> echo 650000 > scaling_max_freq
```

Wait for the new prompt, check the drm display chain and check gstreamer without a video file

```
>> modetest -M stm -c # check the connector id is 32
>> modetest -M stm -s 32:480x272 # check the lcd works fine
>> gst-launch-1.0 -e videotestsrc ! kmssink driver-name=stm connector-id=32 force-
modesetting=true # CTRL+C to stop

Response:
GLib-GObject-WARNING **...
```

Note: *Do not pay attention to the above gstreamer warnings.*

Check GStreamer with a video file (refer to https://wiki.st.com/stm32mpu/wiki/Gst-play):

```
>> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm --videosink="kmssink driver-
name=stm connector-id=32 force-modesetting=true sync=false" --audiosink=fakesink # press "q"
to quit, max cpu load (85-95%) and fps (35-55fps)
>>gst-play-1.0 /usr/local/demo/media/ST2297_visionv3.webm --videosink="kmssink driver-
name=stm connector-id=32 force-modesetting=true" --audiosink=fakesink # press "q" to quit,
max cpu load (~50%) and fps (25fps)
```

Plug a USB3.0 cable (USB gadget Ethernet)

On (PC) console1, enter the following command to get the IP address (<IP@>):

```
>> ifconfig
```

On (target) console2, enter the command:

```
>> ssh root@<IP@> # to connect via ssh to the target
```

On console2, at the prompt, make sure the ltdc configuration is YUV (improved consumption) and check during a video playback that plane0 is used with "format = YU12" (see mp13dk_dri0state.log):

```
>> cat /sys/kernel/debug/dri/0/state
```

On console2, measure the fps and the cpu load. Get the cpu load and the display fps every 4 s, enter "fg" then CTRL+C to stop:

```
>> (while true; do \
m=$(mpstat 4 1 | grep "Average: all" | awk -F" " '{print "cpu load " $3 "%"}'); \
echo "-------------------------"; \
export fps=`cat /sys/kernel/debug/dri/0/state | grep fps -m1 | grep -o '[0-9]\+'`; echo
display ${fps}fps; \
echo $m; \
done) &
```

Copy the video file onto the target, only once (refer to https://wiki.st.com/stm32mpu/wiki/Gst-discoverer):

```
>> scp $output_file root@10.48.0.143:/usr/local/demo/media/
>> gst-discoverer-1.0 -v /usr/local/demo/media/ST2297_visionv3_480p_15fps_1500kbps.mp4
```

4.2.3 The four commands without network

No network, sync, scaling (~40 to 60% cpu load, 15 fps)

```
>> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3_480p_15fps_1500kbps.mp4 --
videosink="kmssink driver-name=stm connector-id=32 force-modesetting=true" --
audiosink=fakesink
```

No network, sync, no scaling (~30 to 50% cpu load, 15 fps)

```
>> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3_480p_15fps_1500kbps.mp4 --
videosink="capsfilter caps="video/x-raw,width=640,height=480" ! videocrop top=208 left=160
right=0 bottom=0 ! kmssink driver-name=stm connector-id=32 force-modesetting=true" --
audiosink=fakesink
```

No network, no sync, scaling (~80 to 95% cpu load, ~30 to 48 fps)

```
>> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3_480p_15fps_1500kbps.mp4 --
videosink="kmssink driver-name=stm connector-id=32 force-modesetting=true sync=false" --
audiosink=fakesink
```

No network, no sync, no scaling (~80 to 90% cpu load, ~30 to 55 fps)

```
>> gst-play-1.0 /usr/local/demo/media/ST2297_visionv3_480p_15fps_1500kbps.mp4 --
videosink="capsfilter caps="video/x-raw,width=640,height=480" ! videocrop top=208 left=160
right=0 bottom=0 ! kmssink driver-name=stm connector-id=32 force-modesetting=true
sync=false" --audiosink=fakesink
```

4.2.4 Network configuration (streaming, not nfs network mount)

Plug a USB3.0 cable (USB gadget Ethernet)

On PC console1, enter the following command to get the IP address (<IP@>):

```
>> ifconfig
```

Copy the video on the web server directory:

```
>> cp ST2297_visionv3_480p_15fps_1500kbps.mp4 /var/www/html/
```

Start a small web server:

```
>> python3 -m http.server 7800
```

Open the following url on your web browser to check the web server *http://localhost:7800/ST2297_visionv3_480p_15fps_1500kbps.mp4*

4.2.5 The four commands with network (streaming)

On target console:

```
>> export VIDEO_FILE=http://<IP@>/ST2297_visionv3_480p_15fps_1500kbps.mp4
```

Network, sync, scaling (~35 to 55% cpu load, 15 fps)

```
>> gst-play-1.0 ${VIDEO_FILE} --videosink="kmssink driver-name=stm connector-id=32 force-
modesetting=true" --audiosink=fakesink
```

Network, sync, no scaling (~25 to 45% cpu load, 15 fps)

```
>> gst-play-1.0 ${VIDEO_FILE} --videosink="capsfilter caps="video/x-
raw,width=640,height=480" ! videocrop top=208 left=160 right=0 bottom=0 ! kmssink driver-
name=stm connector-id=32 force-modesetting=true" --audiosink=fakesink
```

Network, no sync, scaling (~70 to 90% cpu load, ~28 to 45 fps)

```
>> gst-play-1.0 ${VIDEO_FILE} --videosink="kmssink driver-name=stm connector-id=32 force-
modesetting=true sync=false" --audiosink=fakesink
```

Network, no sync, no scaling (~80 to 90% cpu load, ~30 to 55 fps)

```
>> gst-play-1.0 ${VIDEO_FILE} --videosink="capsfilter caps="video/x-
raw,width=640,height=480" ! videocrop top=208 left=160 right=0 bottom=0 ! kmssink driver-
name=stm connector-id=32 force-modesetting=true sync=false" --audiosink=fakesink
```

Note: *Streaming consumes less CPU load.*

4.2.6 Power consumption tables

Table 3 summarizes the consumption of the different use cases with MPU clock @650 MHz:

Table 3. **Power consumptions with MPU clock @650 MHz**

Network	Sync	Scaling	P_TOTAL (mW)
No	Yes	Yes	544
No	Yes	No	538
No	No	Yes	587
No	No	No	537
Yes	Yes	Yes	555
Yes	Yes	No	535
Yes	No	Yes	613
Yes	No	No	621

Table 4 summarizes the consumption of the different use cases with MPU clock @1000 MHz:

Table 4. **Power consumptions with MPU clock @1000 MHz**

Network	Sync	Scaling	P_TOTAL (mW)
No	Yes	Yes	577
No	Yes	No	558
No	No	Yes	678
No	No	No	689
Yes	Yes	Yes	581
Yes	Yes	No	557
Yes	No	Yes	684
Yes	No	No	685

The results are globally coherent:

- Sync = yes means the video is played taking into account its 15 fps framerate → the cpu has time for "sleeping" → lower cpu load → lower consumption.
- Scaling = no means the video is not resized by software → lower cpu load → lower consumption.
- Network = no means less cpu activity → lower cpu load → lower consumption.

4.3 Various low-power modes with OpenST Linux

STM32MP13x product line devices can be used in various power modes, their respective expected consumption is provided in the table at the end of this section.

The measurements are provided on board MB1635B using a “ev1” Weston distribution, which means that some current on V_{DD} supply is dissipated in low-power modes due to components on the board (screen, Ethernet, or other).

Run use cases

1. CRun
   A ‘For’ loop that takes about 100% of one A7 core is launched using below commands.
   ```
   >>while true; do echo “” > /dev/null;done
   ```
2. Stop
   This mode is not enabled on the provided distribution. LP-Stop is used instead.
3. LP-Stop
   The activation of a wake-up source that prevents entering into the deeper low-power mode is needed. UART wake-up source can be used.
   ```
   >>echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
   >>echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
   >>echo mem > /sys/power/state
   ```
4. LPLV-Stop/LPLV-Stop2
   This measurement is not available at this time using the provided distribution.
   It requires to have GPIO wakeup activation implemented in the device tree.
   Measurement not provided.
5. Standby DDR in Self Refresh
   Reset the board (to cleanup any unwanted wake-up source settings)
   ```
   >>echo mem > /sys/power/state **A7 in CStop and System in Standby, DDR in SR**
   ```
6. Standby DDR OFF
   Reset the board or press wakeup button
   ```
   >>shutdown –h 0 **A7 in CStop with PDDS=1 and System in Standby, DDR Off**
   ```

Table 5. **Power consumption in various power mode with STM32MP135F + MB1635B**

System power modes	Power VDD@3.3 V (mW)	Power VDD_CORE@1.27 V (mW)	Power VDD_DDR@1.35 V (mW)	Power VDD_CPU@1.25V (mW)(1)	Power Total (mW)
CSleep	132	247	54	23	455
CRun adaptive MPU clock	132	255	65	185	636
CRun MPU clock @650 MHz	132	249	61	113	555
CRun MPU clock @1000 MHz	135	257	63	197	653
LP-Stop	43	4	9	0.03	56
Standby DDR in SR	36	0.3	9	0.002	46
Standby DDR OFF	0.02	0.02	0.01	0.02	0.07

1. *V_{DD_CPU}@1.35 V when MPU clock @1000 MHz*

For more details on power modes of the STM32MP13x product lines, refer to [AN5565].

Revision history

Table 6. Document revision history

Date	Version	Changes
13-Jan-2023	1	Initial release.

Contents

1 General information 2

2 Overview 3

3 STM32MP135F-DK board 4

3.1 Measurement points 4

3.2 STM32MP1 OpenSTLinux Distribution 8

4 Power consumption on STM32MP13x product lines 9

4.1 CoreMark® on Linux MPU 9

4.2 Graphical use case: 480P video playback 10

4.2.1 Video file transcoding 10

4.2.2 Preparation and various checks 11

4.2.3 The four commands without network 12

4.2.4 Network configuration (streaming, not nfs network mount) 13

4.2.5 The four commands with network (streaming) 13

4.2.6 Power consumption tables 14

4.3 Various low-power modes with OpenST Linux 15

Revision history 16

List of tables 18

List of figures 19

List of tables

Table 1. Configurations of the STM32MP13x product line devices . . . 3
Table 2. CoreMark® power consumption on STM32MP135F + MB1635B . . . 10
Table 3. Power consumptions with MPU clock @650 MHz. . . . 14
Table 4. Power consumptions with MPU clock @1000 MHz . . . 14
Table 5. Power consumption in various power mode with STM32MP135F + MB1635B . . . 15
Table 6. Document revision history . . . 16

List of figures

Figure 1. Board with cables 4
Figure 2. Power control block diagram 5
Figure 3. Resistors to remove for current measurement. 6
Figure 4. Power supply lines 7
Figure 5. Measurement points on the board 8