

N7 GDD

GDD

Graphics Device Driver Presentation

Maurizio Davanzo: maurizio.davanzo@neonseven.com

NEONSEVEN

Agenda

- Overview
- Block Diagram and interface
- Features and System requirements
- **GDD structure**
 - GDD process Layer (upper)
 - GDD Middle layer
 - GDD Low Level Layer (GTL) (lower)
- GDD Start-up sequence
- Phone tool interface
- Interface



Overview

- This presentation will give you a description about the generic GDD driver (from DWD) and then explain the specific solution used by N7 in the BP30 GDD driver.
- In the BP30 GDD only the main LCD display is supported, so every references to camera sensor are not implemented in BP30

GDD reference Documentation

- Documentation from N7:
 - BP30 Graphics Device Driver Specification
 - BP30 GDD Interface Specification
 - (BP2 Graphics Device Driver Specification)

- Documentation from DWD:
 - Graphics Transport Abstraction Layer driver – GTL (module specification)

Overview

- The GDD driver is prepared for supporting different kinds of graphics HW. Some of the main features are:
 - Flexible HW interface (Serial/Parallel, 8bit/16bit).
 - High speed data transmission (PEC).
 - Dual LCD configuration.
 - LCD only configuration.
 - Camera module

Features

- The GDD generic driver is made for supporting different kinds of graphics HW and is easily extended.
- The GDD driver is providing a function interface between the application and the graphics hardware for controlling the LCD and camera sensor. The main features are:
 - Updating the LCD displays
 - Setting the LCD contrast
 - Power saving LCD
 - Entering camera mode (viewfinder)
 - Capturing camera picture

NOTE: In BP30 GDD the camera is not supported.

Features

■ Hardware requirements

E-GoldRadio

- Chip select logic (for LCD)
- PEC channel
- SSC
- Core timer

LCD controller

- on-chip display data RAM
- Auto-increment of address
- Possibility to read from display data RAM (only if camera is present)

Features

- Software requirements

Supported operating systems

- OSE

Operating system resources

- Two processes with equal priority (one for the command reception handling and another one for the command execution control.)
- Interrupt routines for core timer

Overview

■ Configuration using LCD and Camera Sensor

N7 GDD generic driver can be configured for applications with a camera sensor and one or two display

Examples:

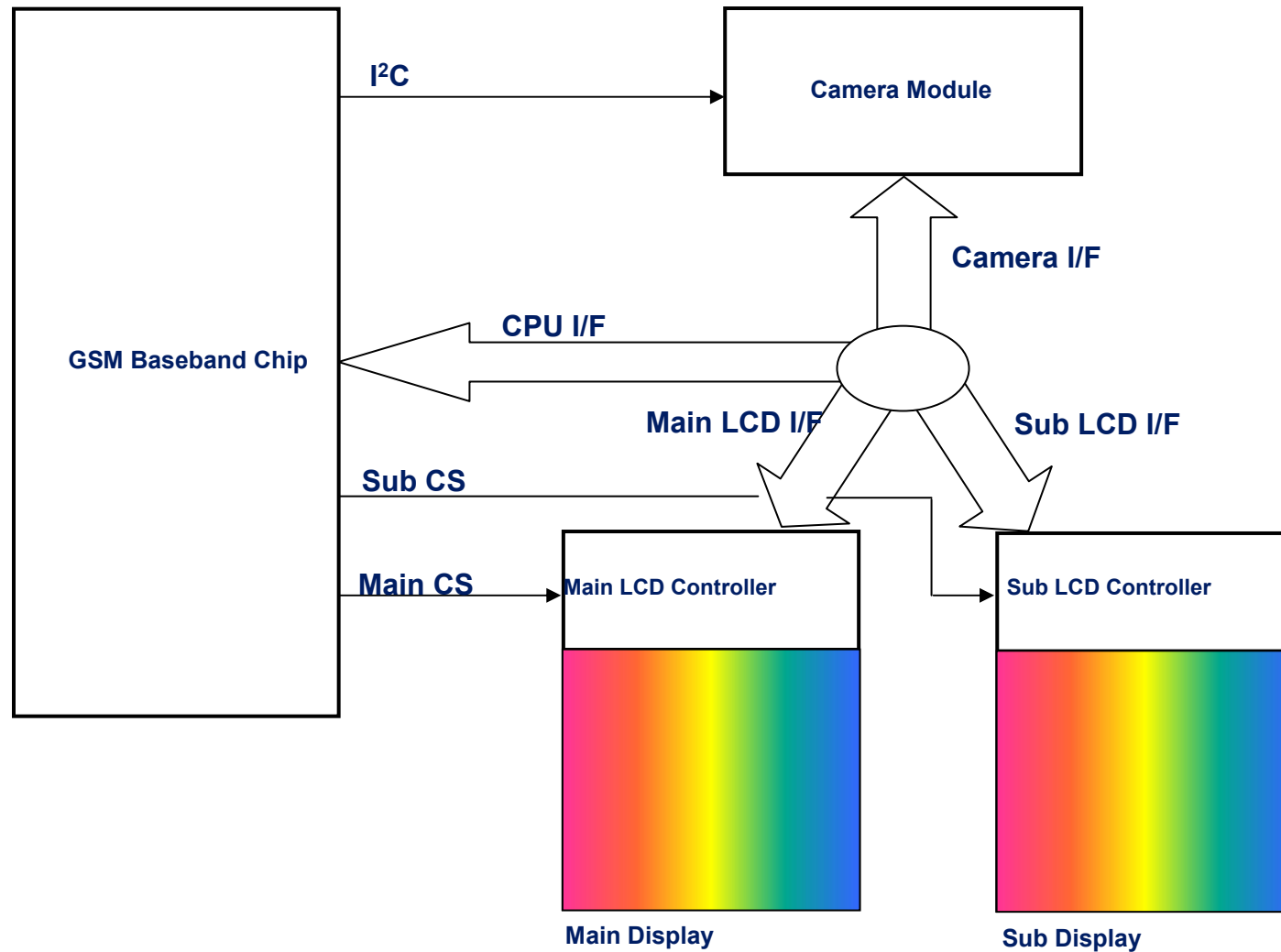
BP2

- I2C connection for programming camera
- Serial connection between camera and display LCD for transferring data
- Serial connection between micro and display LCD for programming and transferring data

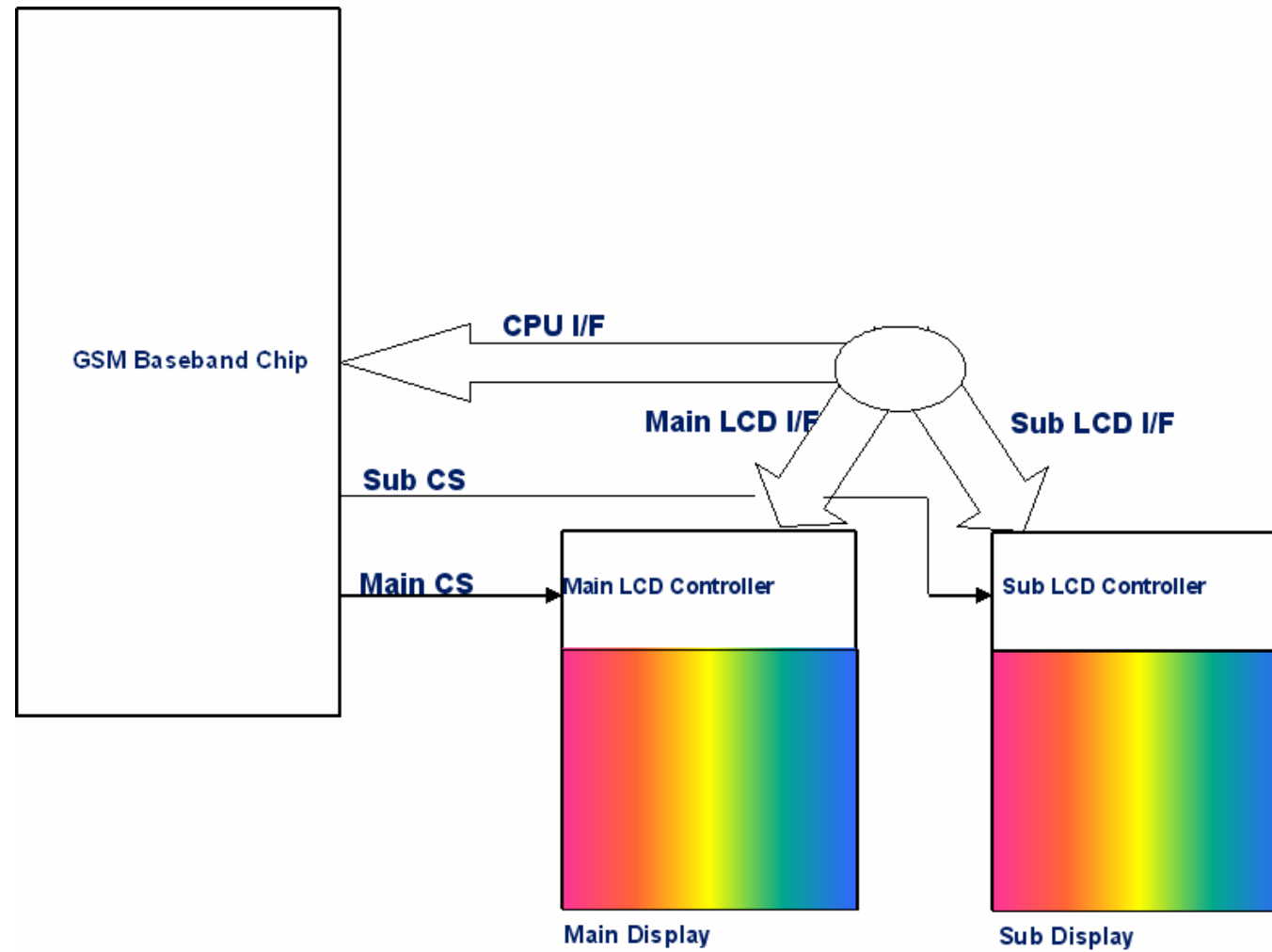
BP30

- Serial connection between micro and display LCD for programming and transferring data

Exammple: BP2 configuration



Example: BP30 configuration

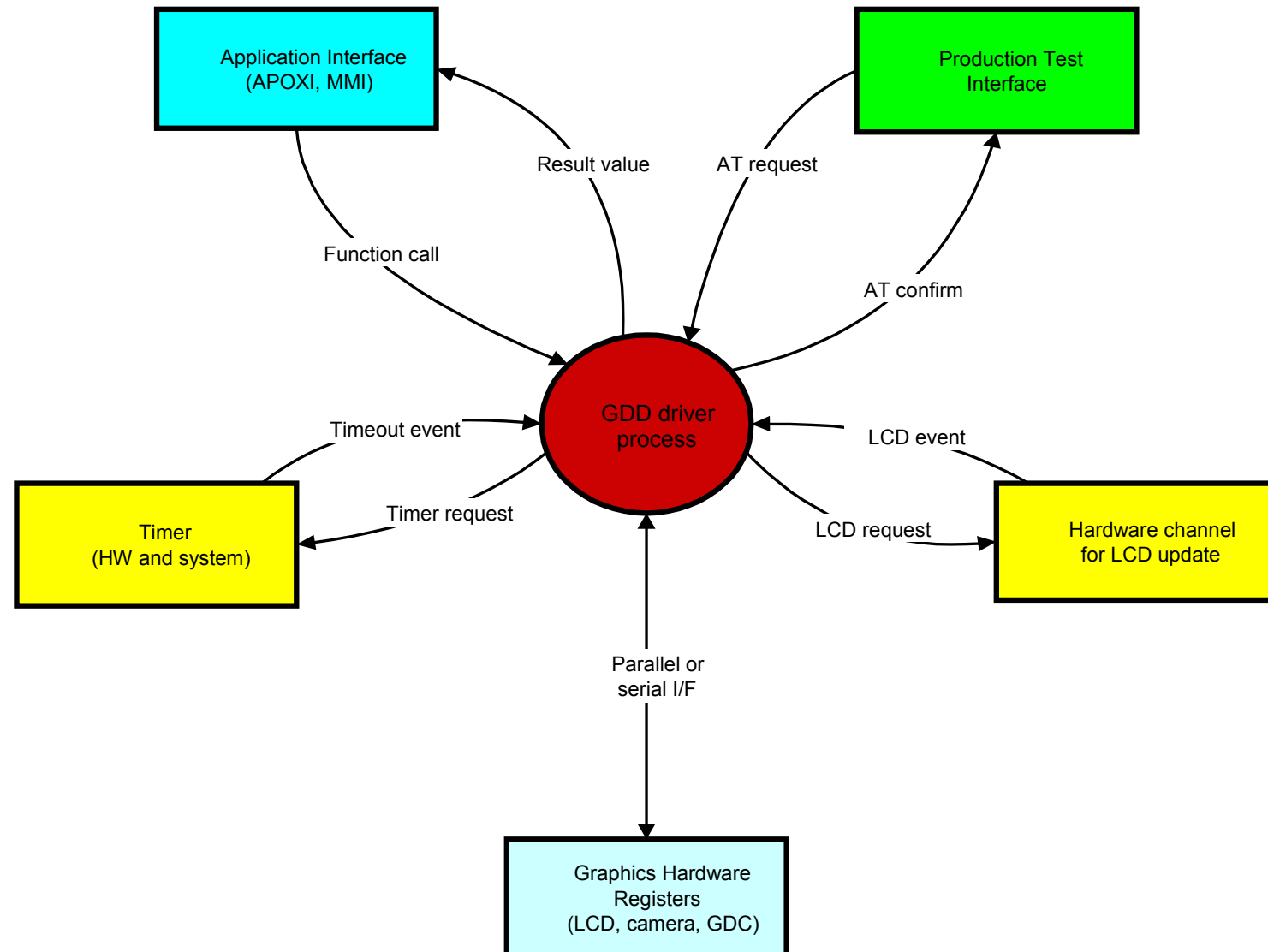


GDD terminators

The GDD driver has several terminators:

- The application calls the GDD driver interface routines, and a result is often returned.
- In production test, a PC program is requesting LCD commands using a serial adapter to the mobile station, and responses are sent back by the GDD driver.
- The GDD driver is using hardware timer when accessing the graphics hardware.
- A dedicated hardware channel is used when updating the LCD image.
- The graphics hardware registers are accessed during LCD or camera operations.

GDD terminators

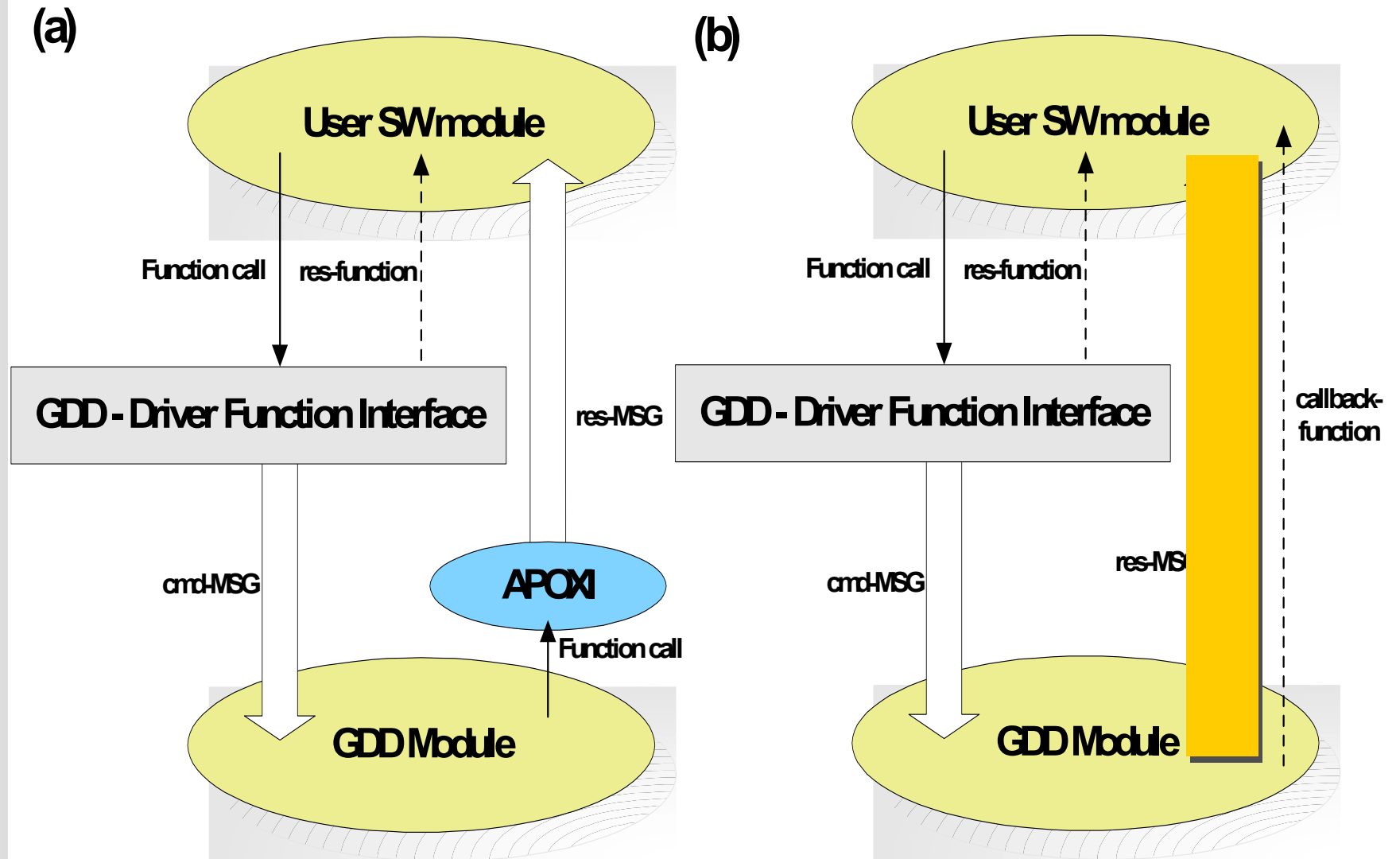


GDD Driver Function Interface

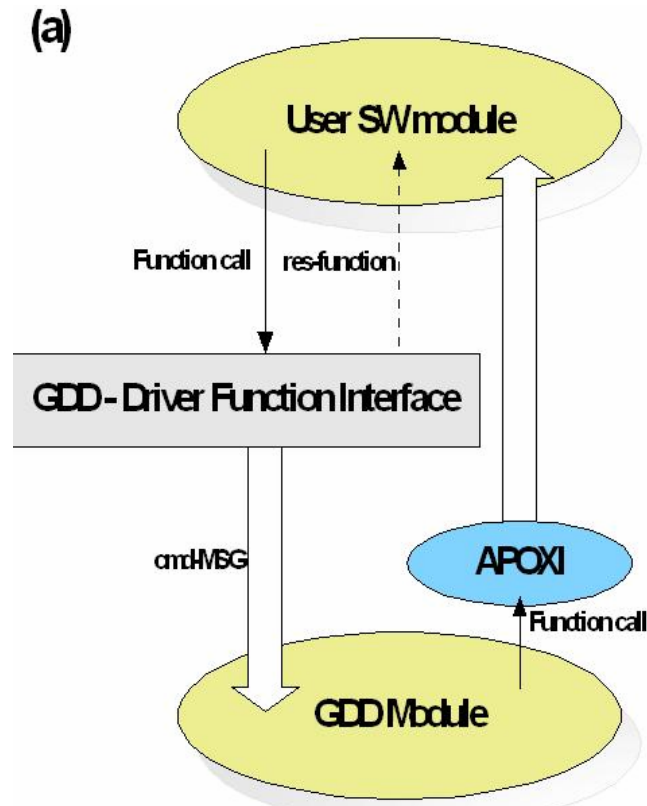
BP30 GDD:

- gdd_result_type **GDD_lcd_init**(gdd_hw_id_type hw_id)
- void **GDD_lcd_update**(SDL Pid proc_id, ushort res_code, gdd_hw_id_type hw_id, ubyte shuge *vram, ushort left, ushort top, ushort right, ushort bottom)
- gdd_result_type **GDD_lcd_set_contrast**(gdd_hw_id_type hw_id, sshort level)
- gdd_result_type **GDD_lcd_get_contrast_limits**(gdd_hw_id_type hw_id, gdd_limits_type *dst)
- gdd_result_type **GDD_lcd_set_power_save_mode**(gdd_hw_id_type hw_id, gdd_lcd_power_save_mode_type mode)

GDD Generic Function Interface Interaction



Function Interface Interaction (a)



- The application call the GDD function interface.
- The function interface generates a message with the requested command. This message is protected by using the semaphore *gdd_api_semaphore*.

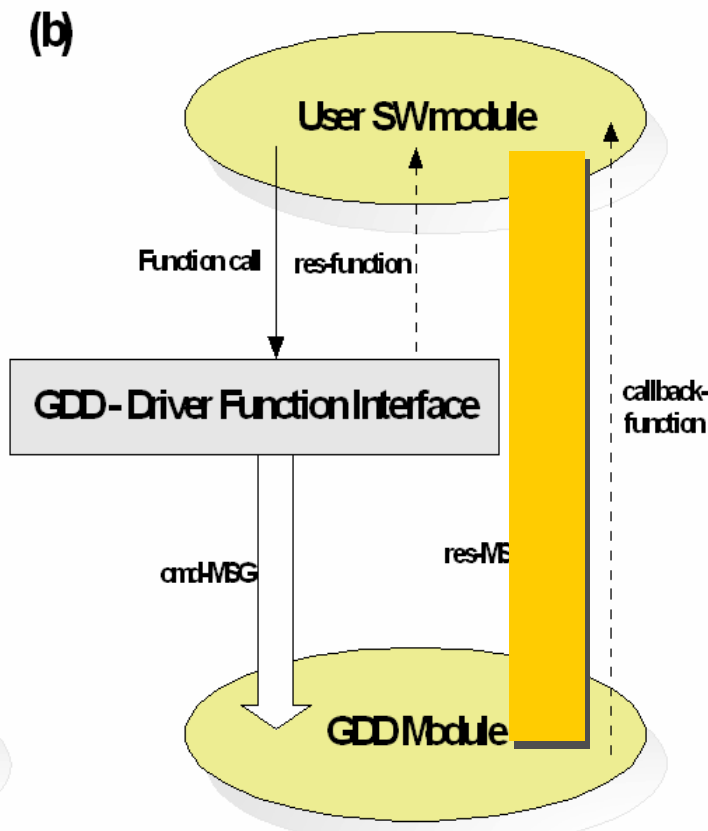
Function without timing critical:

- The Function Interface will hold back the application until the result has been stored by the GDD module. The hold back mechanism is implemented by using the semaphore *gdd_api_result_semaphore*.

Function with timing critical:

- the application is only hold back until the cmd-MSG has been send to the GDD module.
- When the command is executed, a message is send to the application by a APOXI callback function, which will send a message to the right application.

Function Interface Interaction (b)



Function without GDD interaction:

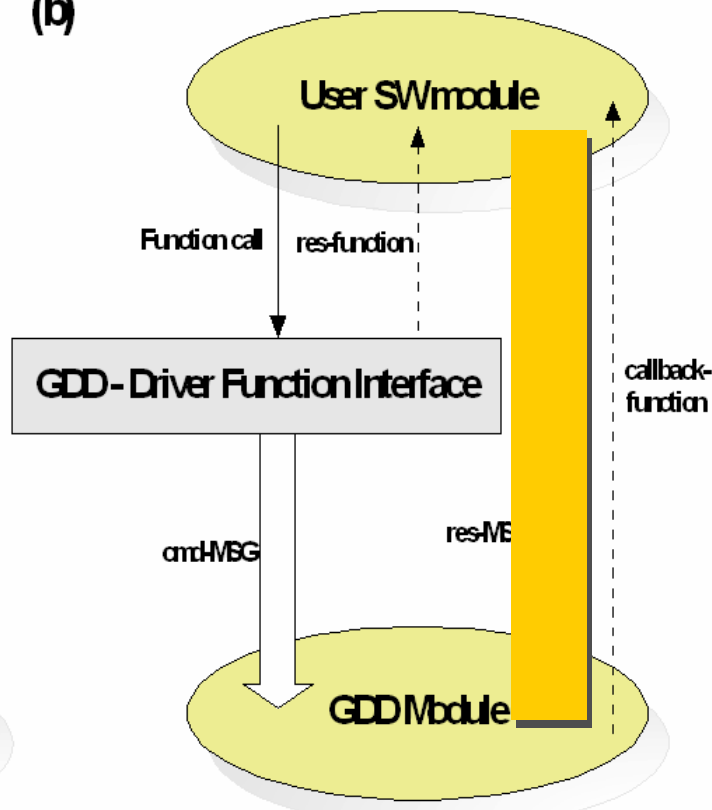
- The application call the GDD function interface.
- The function will return immediately if no interaction with GDD is needed, this is for instance used for `GDD_lcd_get_contrast_limits()`.

Function with GDD interaction, timing critical:

- The application call the GDD function interface.
- The function interface generates a message with the requested command. This message is protected by using the semaphore *gdd_api_semaphore*.
- A callback function is called in the User SW module when the command has been executed. The callback function will be specified by an input parameter in the GDD function.

Function Interface Interaction (BP30)

(b)



- BP30 used the secon approach.

- **Defines configuration:**

- GDD_LCD_UPDATE_CALLBACK >> TRUE
- GDD_SYNC_LCD_UPDATE >> FALSE

NOTE: In BP30 the functions interface Gdd_lcd_update() receive a NULL value as callback paramiter.

GDD Semaphores

Semaphores are used to protect some critical section code

A semaphore must be used like a generic variable:

SEMAPHORE *sem_gdd_api_access;

This instruction define a variable shemaphore

sem_gdd_api_access = create_sem(1);

This instruction initialize it to 1

GDD_request_api_semaphore

This function decrement the semaphore of 1

GDD_release_api_semaphore

This function increment the semaphore of 1

NOTE

When the semaphore become -1, the process who called GDD_request_api_semaphore become blocked until the semaphore switch again to 0

GDD Semaphores

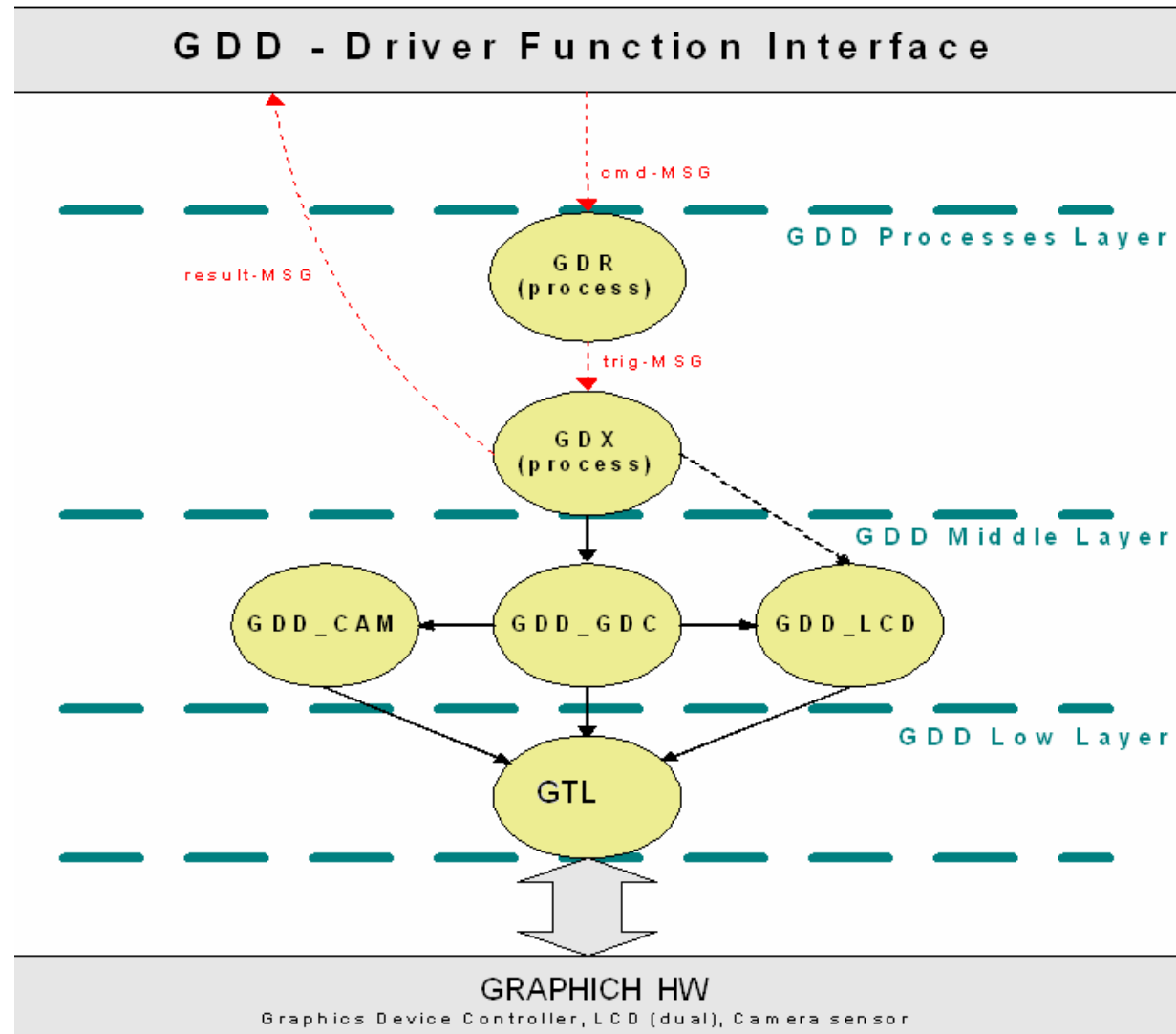
In GDD Driver all semaphores are initialized at boot up by the function:

`GDD_init_rtos_resources()`

Exampe:

See the `GDD_lcd_update()`;

Generic GDD structure



Generic GDD structure

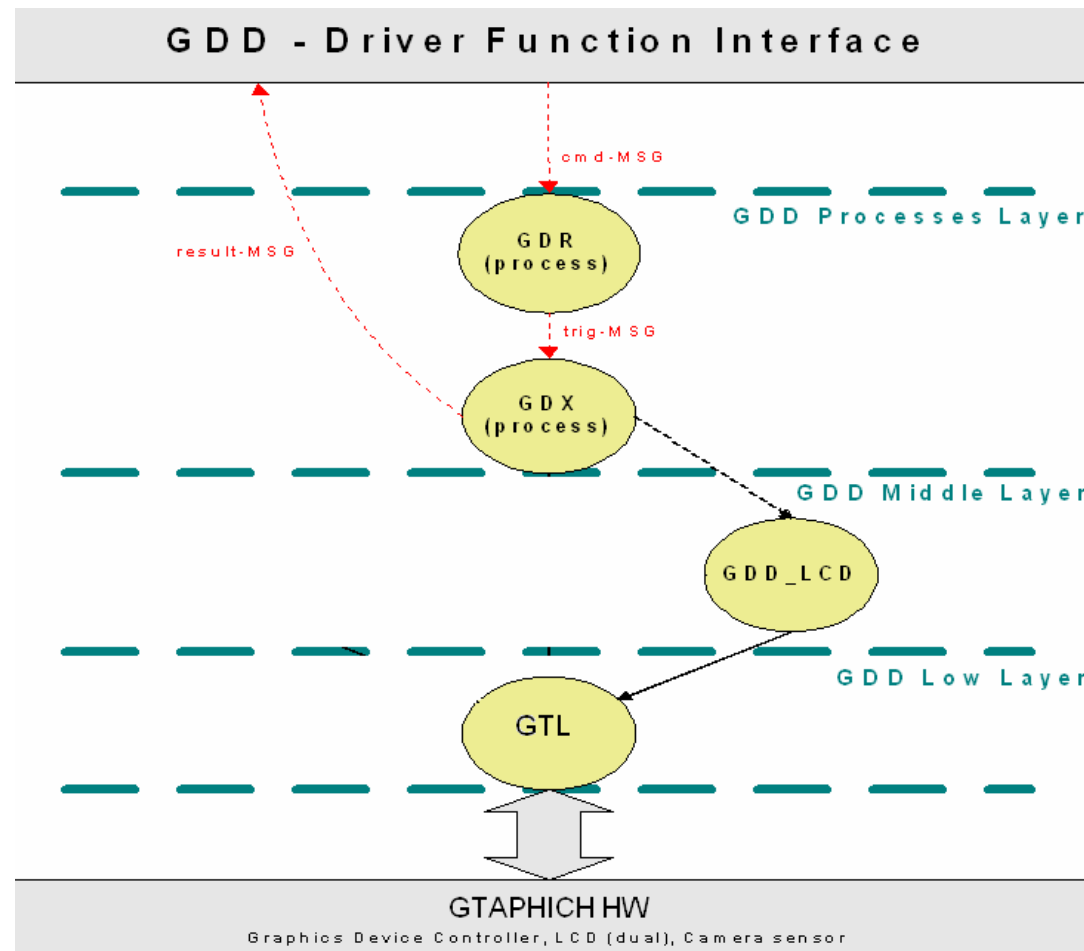
- **GDD Processes Layer**, in this layer there is running two processes, namely GDR (GDd message Receiver) and GDX (GDd message eXecuter). These two processes are responsible for receiving messages from the Driver Function Interface and execute them in a sequential order.
- **GDD Middle Layer**, each one of the middle layer modules abstracts a Graphical HW block. The module contains a state machine and all possible commands for that particular HW block.
- **GDD Low Layer (GTL)**, the low level layer is responsible for interfacing with the HW on the platform

Generic GDD structure

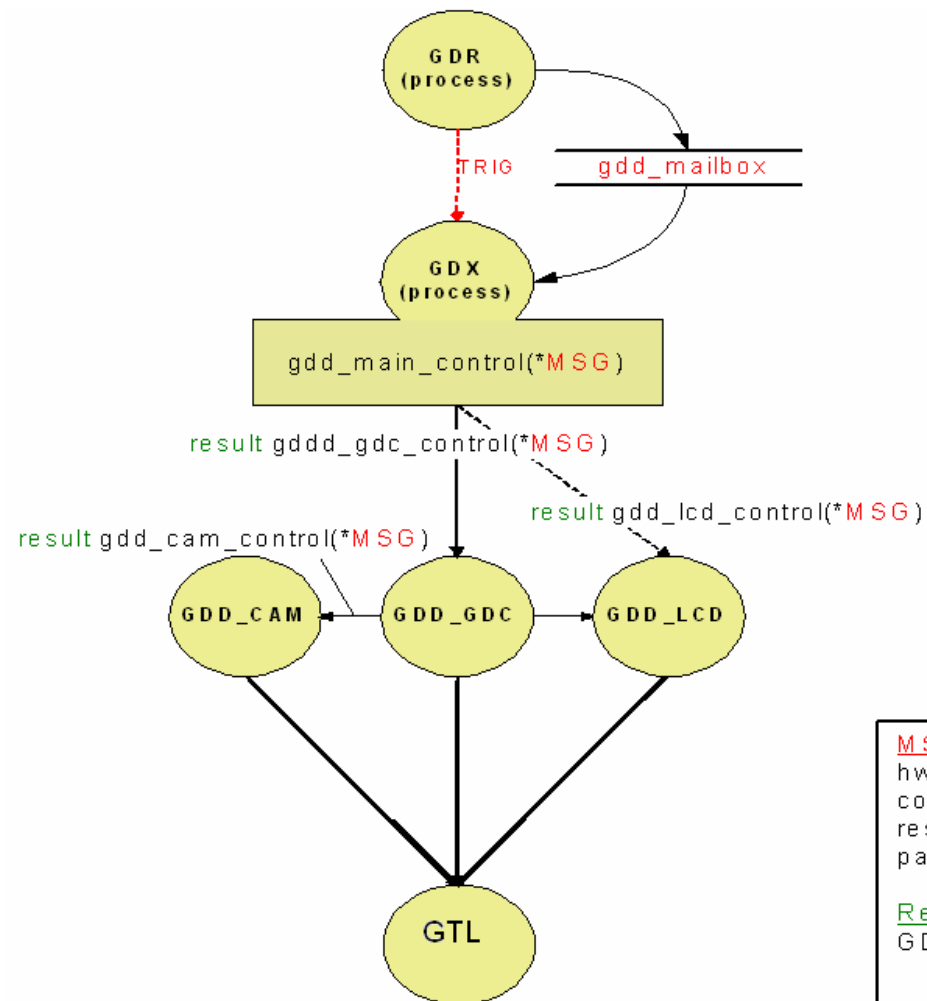
- Whenever a GDD Driver Function is called, a message with the requested command and parameters is generated and sent to the receiver sub module GDR (process).
- The GDR process triggers the GDX module when a message has been received.
- The executer sub module GDX (process) then executes the message by calling the GDC state machine.
- If needed the GDC state function can call routines in the other sub-module.

BP30 GDD structure

- In BP30 project the middle layer contain only the LCD state machine.
- All commands sent by GDX to the display go directly to the LCD state machine.



Generic GDD messages flow



Generic GDD messages flow

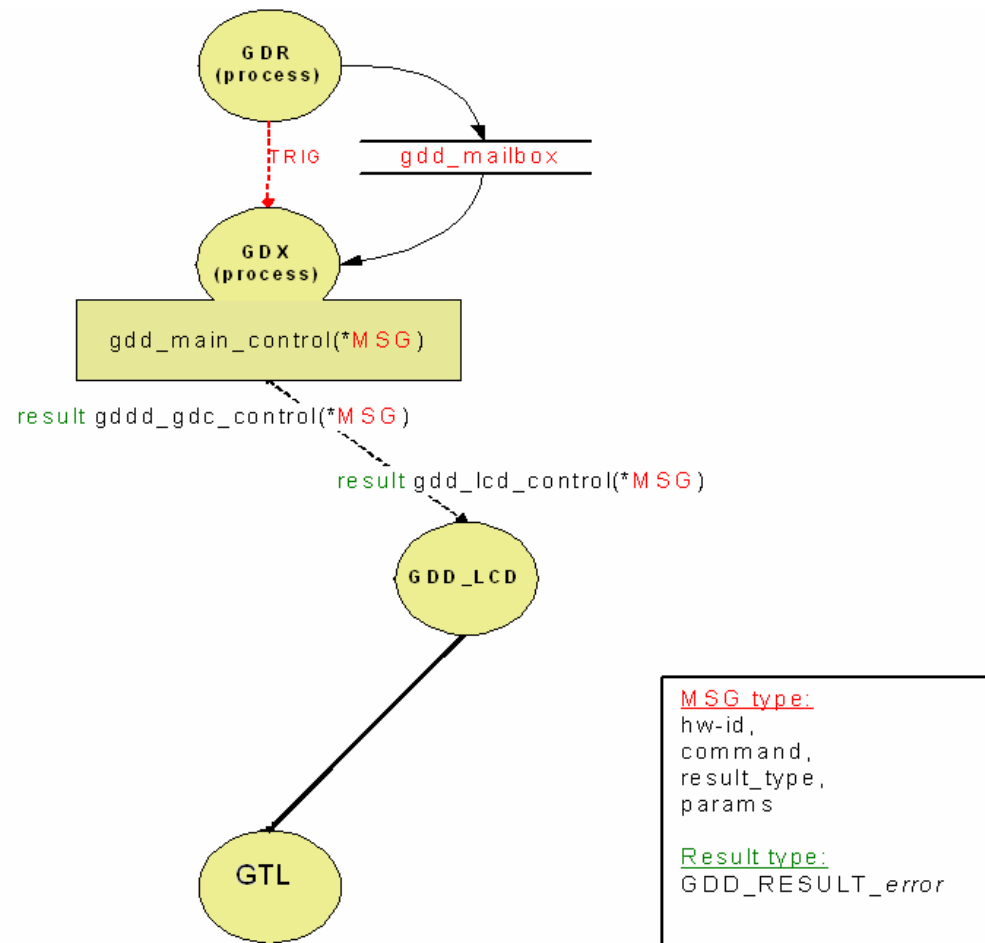
- The GDR process receives the message from the GDD interface and stores it inside the mailbox
- As soon as the mailbox is not empty, the GDR process triggers the GDX process
- GDX receives the message and executes it by the `gdd_main_control()`.
- `gdd_main_control()` forwards the incoming message to the GDC sub-module (LCD sub-module in BP30)
- GDC sub-module can send commands to other sub-modules
- Each middle layer module contains a command control structure, which contains all the possible commands for that specific module. So, by the GTL layer, the command is executed

NOTE

An important feature of the chosen structure is that all incoming requests are handled sequentially, where one command is executed at a time

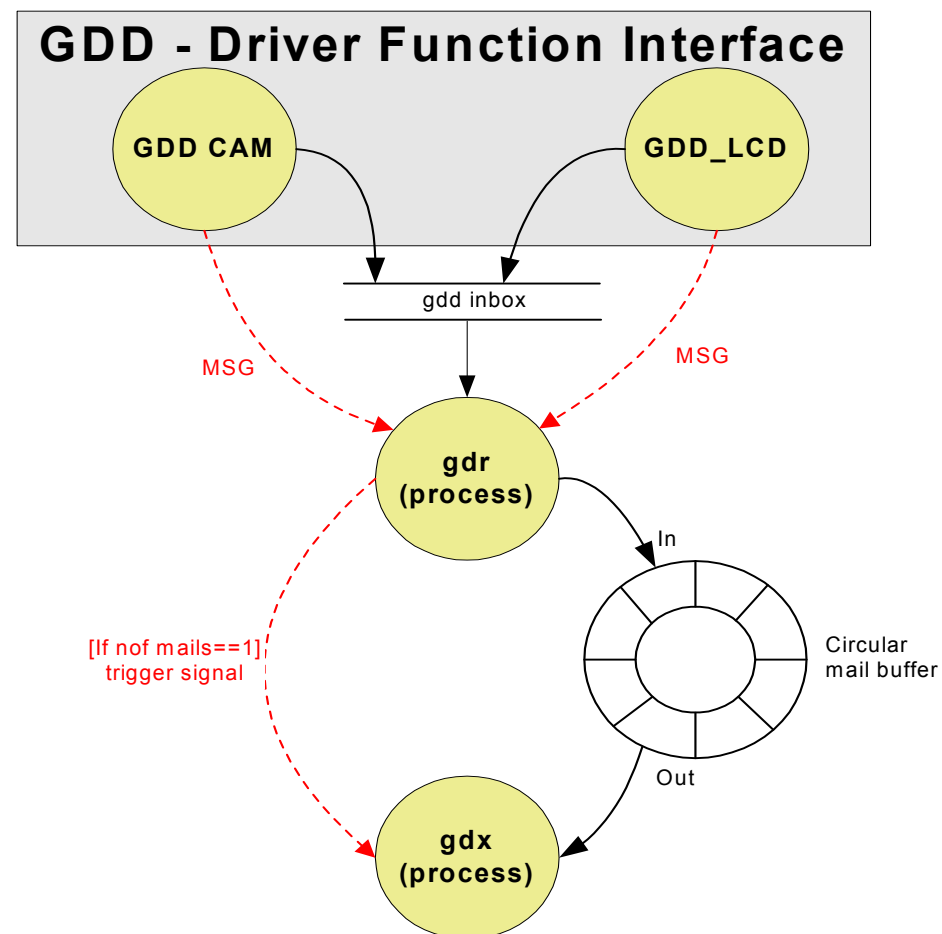
BP30 GDD messages flow

- On BP30 platform the function `gdd_main_control()` forwards the incoming message directly to LCD sub-module



GDD process Layer

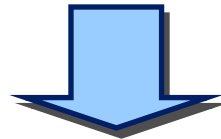
Interaction between the Driver Function Interface, the GDR process and the GDX process



GDD process Layer

The process layer is made of two process:

- **GDR:** is responsible for receiving messages from the Driver Function Interface. The messages are copied to the circular mail buffer and the calling application can continue . The GDR process will continue to fetch messages as long as there are messages in queue, but when the GDD inbox is empty, it will suspend itself.



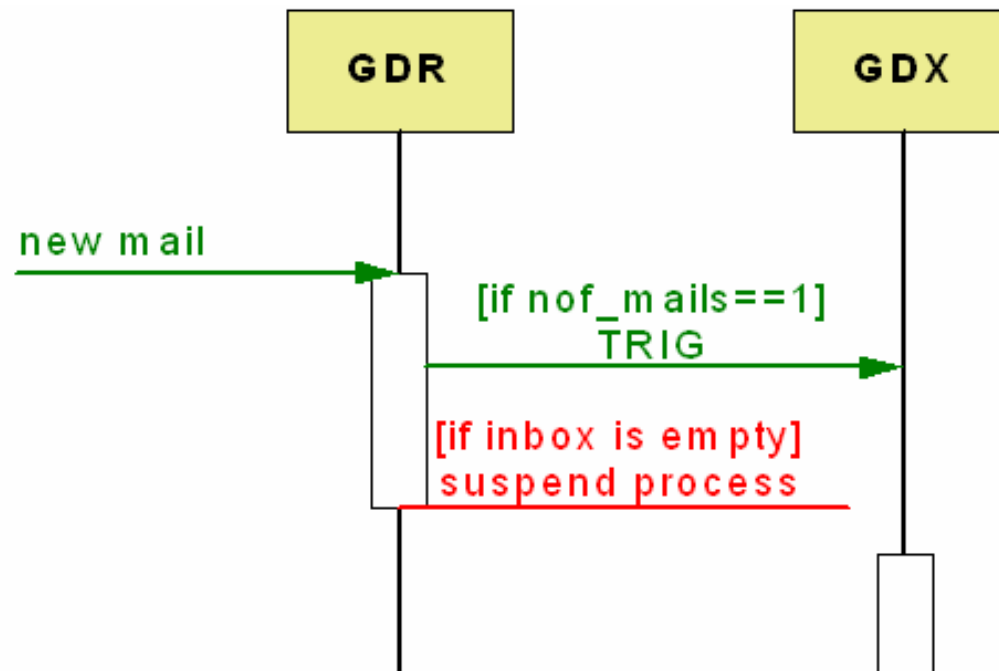
It triggers GDX when the circular mail buffer contains exactly one mail

- **GDX:** The GDX process is responsible for executing all the messages in the circular mail buffer. It continues to run until the buffer is empty

NOTE

the circular mail buffer is protected by the semaphore *gdd_api_semaphore*

Sequence Diagram of GDR and GDX

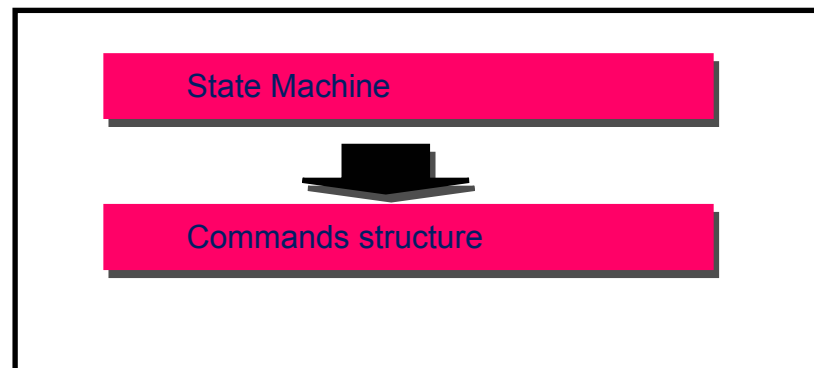


Sequence Diagram of GDR and GDX

- If there are no messages in mailbox both processes will be suspended.
- When a new message is sent, GDR process will resume, and as long as there are pending messages, it will continue to store them in the circular mail buffer.
- The GDX process is triggered by GDR when there is exactly one message in the circular mail buffer, but it will not run until the GDR process suspends itself.
- The GDR is suspended when there are no more incoming messages and then the GDX will start executing the messages.

Generic GDD middle Layer

- The generic GDD contains 3 different middle layer module
 1. GDD_lcd (lcd module)
 2. GDD_cam (camera module)
 3. GDD_GDC (may be used for a companion chip)
- Each module contains a state machine and a command structure, which contains all the possible commands for the specific module.



BP30 GDD middle Layer

- In BP30 project GDD driver contains only one module:
GDD_lcd (lcd module)

The state machine and the command structure are respectively implemented in functions:

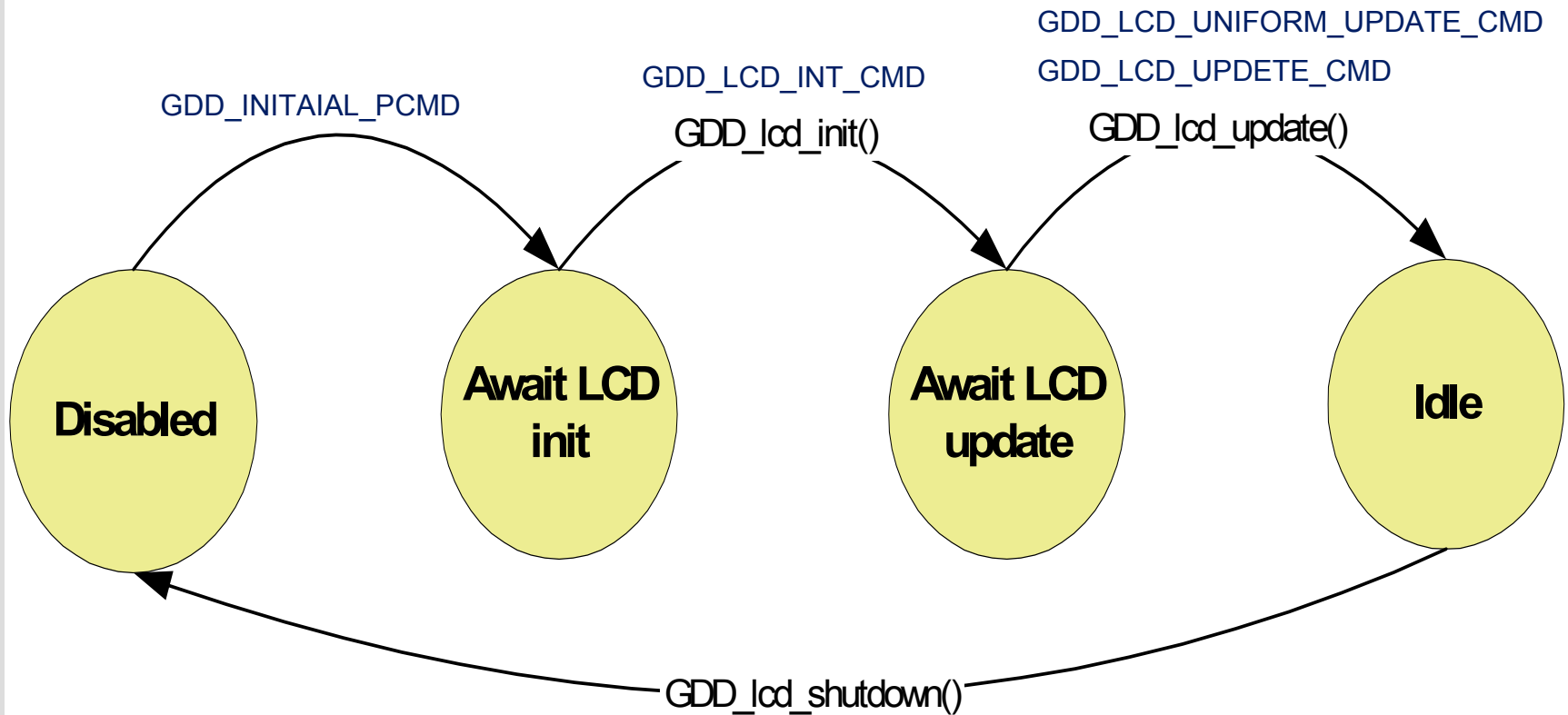
- GDD_cam_control();
- gdd_lcd_cmd_control();

LCD sub-module

The LCD sub-module is supporting the following features for the LCD:

- LCD controller initialization.
- LCD update.
- Contrast control.
- Power save.

LCD sub-module (LCD state machine)



LCD sub-module (LCD state machine)

GDD_INITAIAL_PCMD: (at boot up)

- set the LCD profile in GTL
- LCD reset HW

GDD_INITAIAL_CMD:

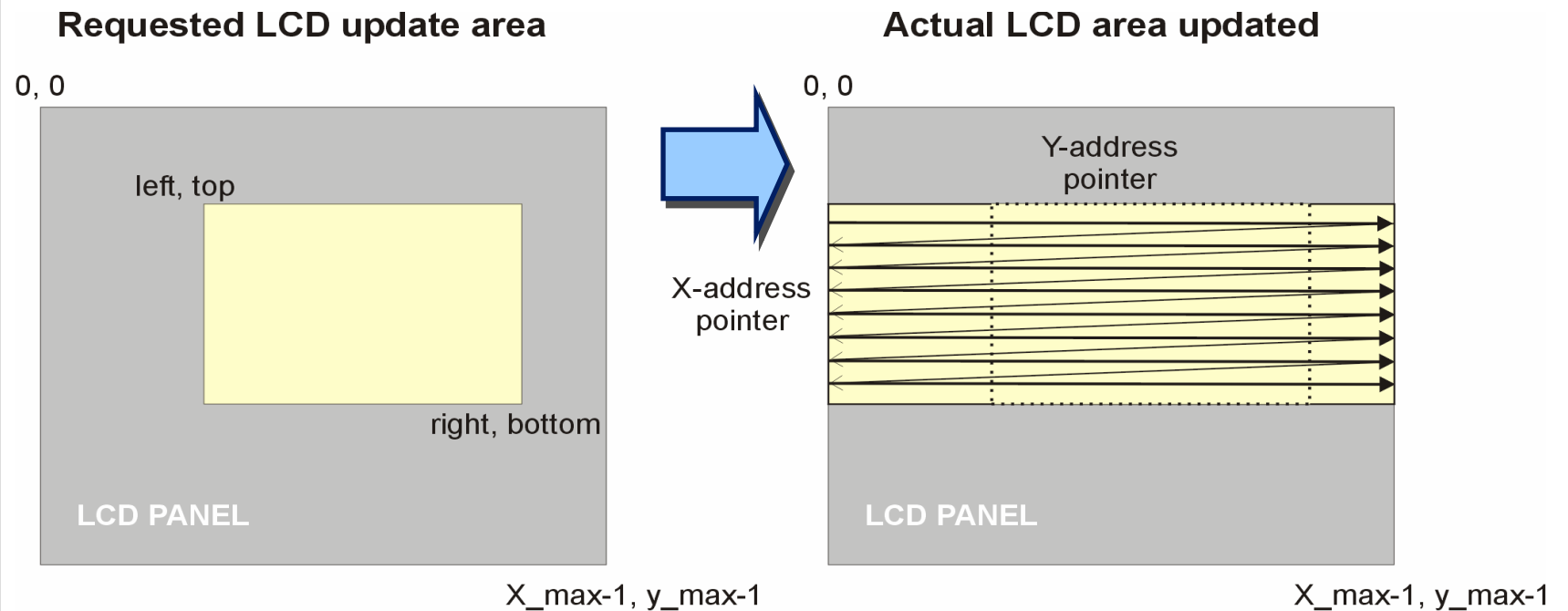
- Send the init_table_commands
- Send the cmd GDD_LCD_UNIFORM_UPDATE_CMD that paint the display in black

GDD_LCD_UPDATE_CMD:

- Set window address area on LCD
- Paint the image on display

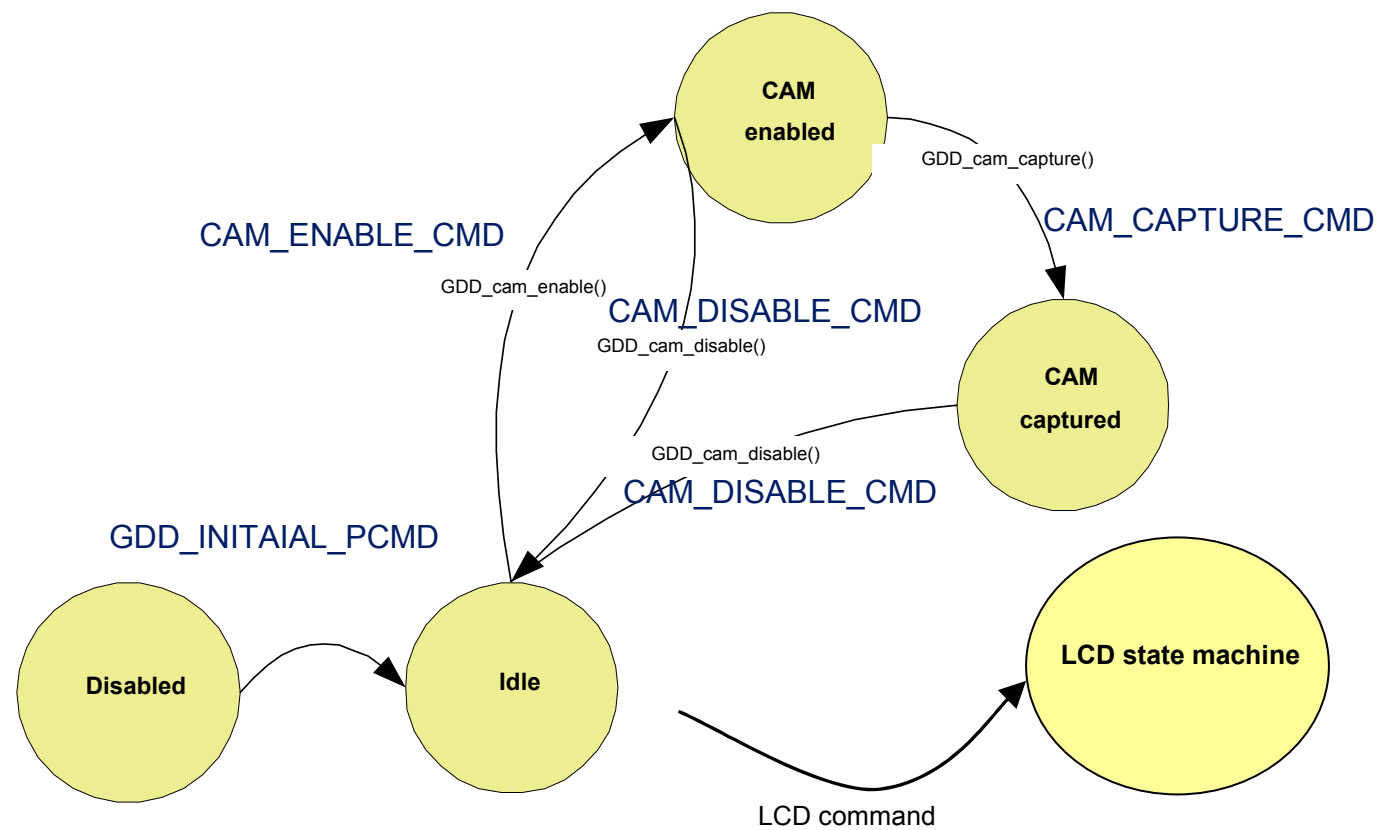
LCD sub-module (LCD_UPDATE)

- When a LCD update cmd is sent to the display the windows coordinates are automatically modified in this way:



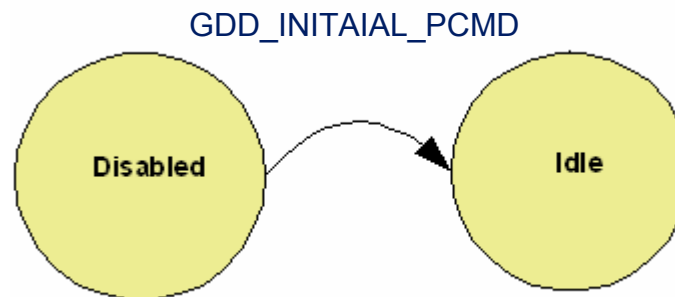
GDC sub-module (GDC state machine)

- Not used in BP30



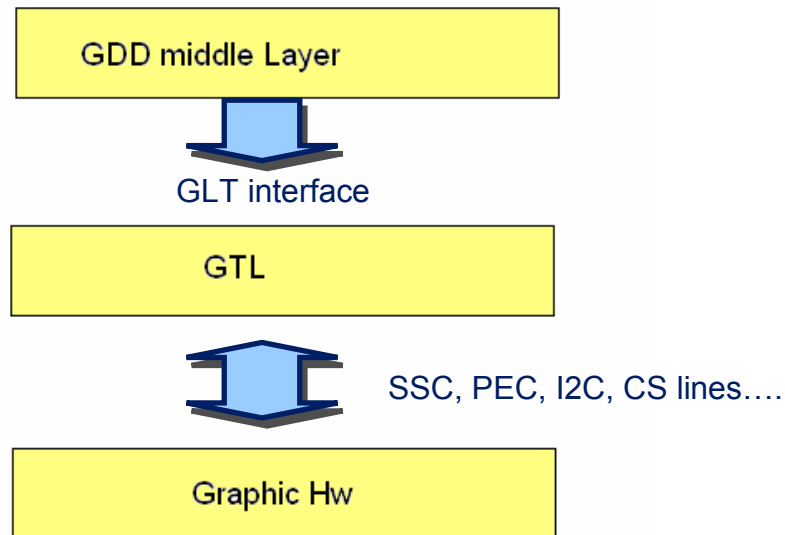
Camera sub-module (Camera state machine)

- Not used in BP30



GDD Low Level Layer GTL

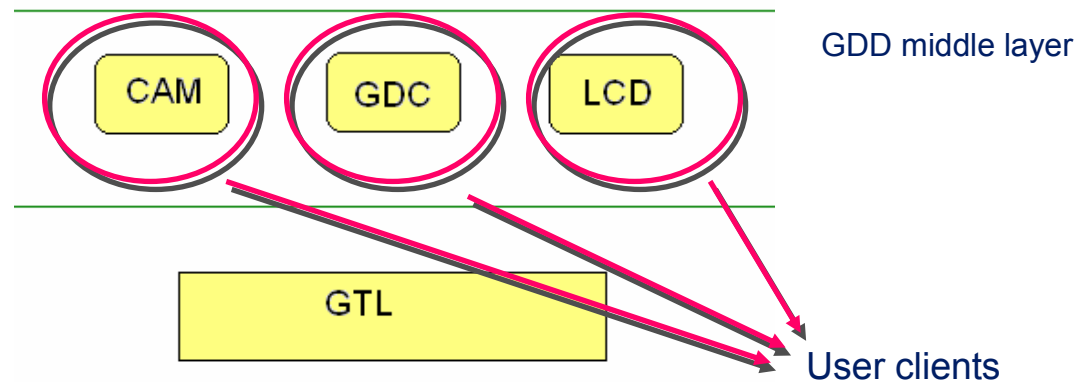
- GTL handles all the graphic hw resources and offers to the upper layer (GDD middle layer) a simple and useful interface in order to use these resources.



- Other HW resources could be: LCD, Camera, Companion chip...

GDD Low Level Layer GTL

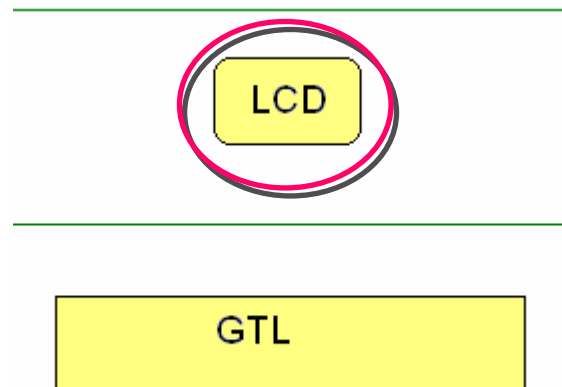
- In the GDD Middle Layer every Hw resources are represented with a module (state machines + command structure).
- In GTL these modules are named **user clients** and they represent the clients who need to access to the Hw resources.



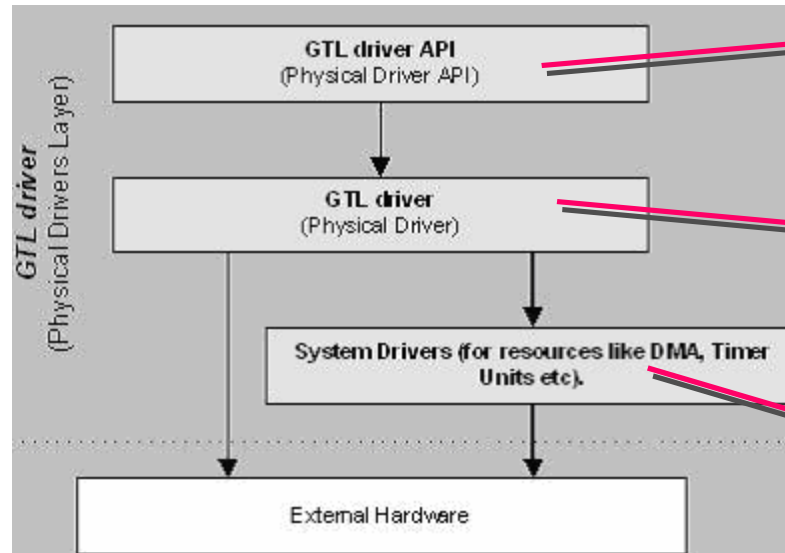
- Other HW resources could be: LCD, Camera, Companion chip...

GDD Low Level Layer GTL

- GTL has been made with a flexible structure in order to handle many hw resources and many user client without conflicts.
- In BP30 there is only one user client that represents the LCD sub-module:



GTL architecture (static view)



GTL Driver Api Layer: GTL interface layer

GTL Driver Layer: This layer implements the GTL driver

GTL platform depended device control (System driver): These are the drivers for the system resources, example: SSC, PEC, I2C...

GTL Driver API

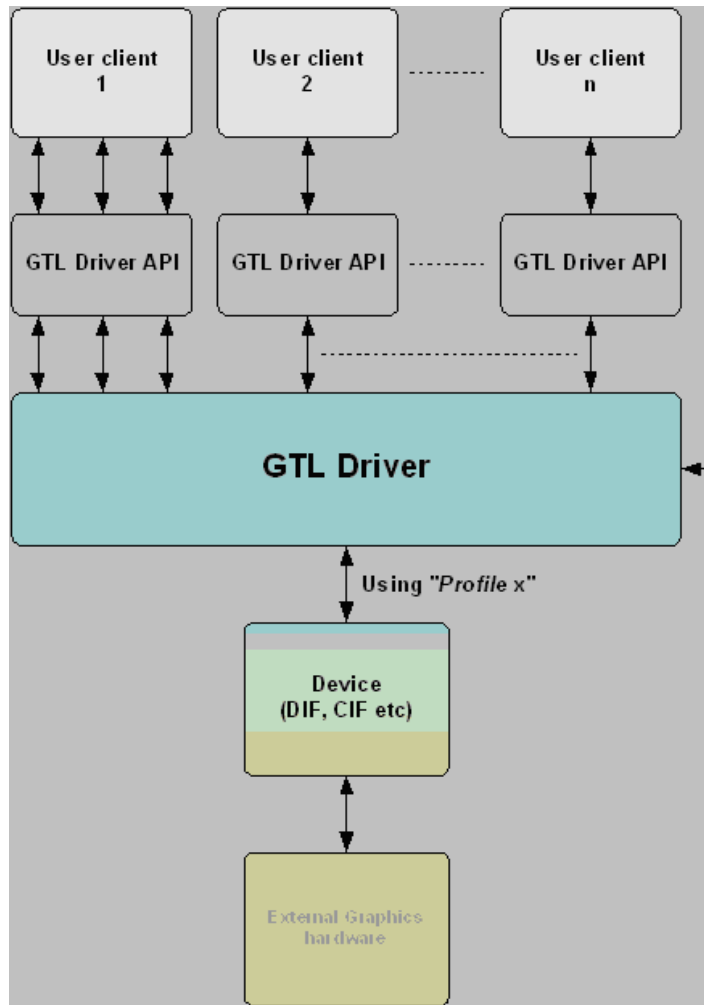
- GTL supplies towards every user client a set of function interface that allows the sub-moudule to access to the Hw resource:

- **OPEN**:The user clients use this interface to open an active connection with the GTL driver
- **CLOSE**: The user clients use this interface to close an active connection with the GTL driver
- **write_register**: The user clients use this interface to write a single <register index, data> to the external hardware.
- **write_register_tab**: The user clients use this interface to write a sequence of <register index, data> to be written to the external hardware.
- **read_register**:The user clients use this interface to read from a particular register of the external hardware.
- **write_data**: The user clients use this interface to write data to the external hardware.
- **read_data**: The user clients use this interface to read data from the external hardware.

GTL Driver

- GTL maintains a database (*ssc_profiles_db*) that contains information about each active user client.
- Every elements of such database is named user client's profile
- Every profile contains information about the client and how the client can access to the Hw resource (example: using SSC, I2C...)

GTL Driver (interaction between upper layer and GTL)

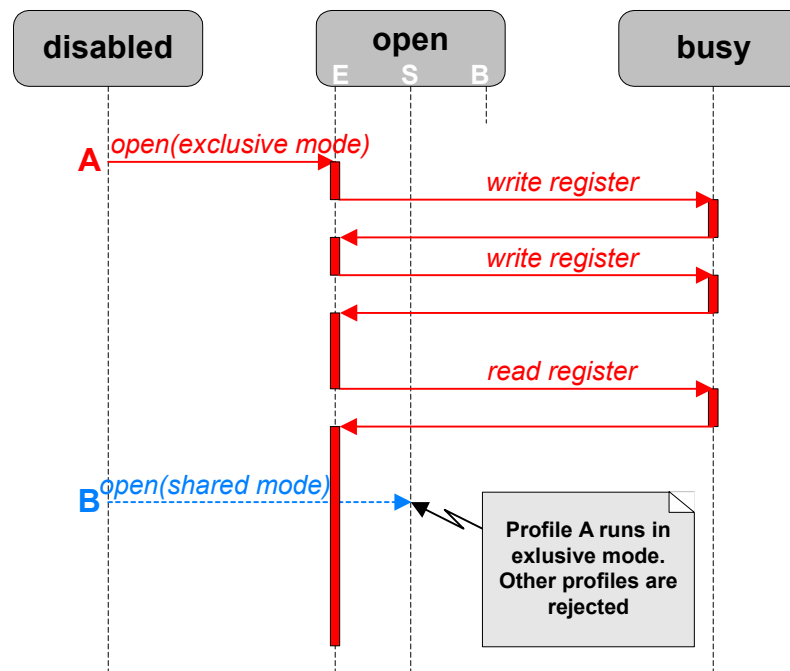


- The user client opens the profile by OPEN interface.
- All the information about client are stored in the user profile.
- Multiple clients can have an active profile at any points of time.
- GTL handles the Hw resources and, at any points of time, only one client gain the access to the resource
- In **BP30** there is only one user client (LCD) and the corresponding profile is opened at initialization.
- In **BP2** there are 2 user client (LCD and Camera) and the corresponding profiles are opened at initialization.

GTL Driver (internal handling of Profiles)

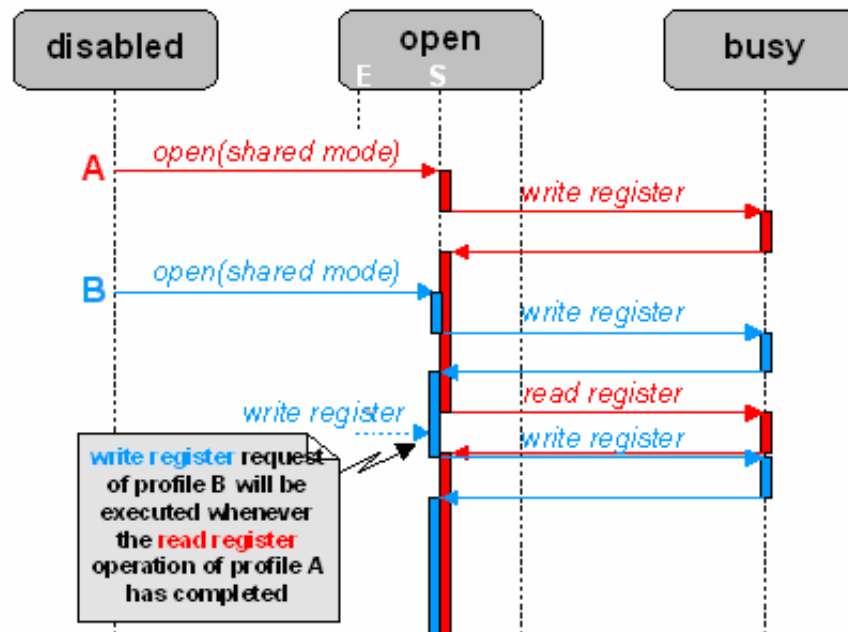
When a new profile is opened, it must specify the mode in which the hardware resource needs to be used:

- *exclusive* mode: Only the user client in exclusive mode can use the resource.



GTL Driver (internal handling of Profiles)

- *shared mode*: The resource is shared between more user clients



- While the profile is performing a critical task (like “Write Register”) the operation is protected with a semaphore (ex: SSC is protected by `sem_gtl_ssc_control`).

GTL Driver (System Driver)

The system driver layer is made of all the system resources that can be used by GTL in order to communicate with the Graphic Hw:

- PEC
- SSC
- I2C
- Timer units
- Interrupt controller

GDD Start-up sequence

- The GDD initialization is the following:.
 - GDD_init_rtos_resourcs(): initialize the semaphores used to protect critical code sections from concurrent execution.
 - The two processes gdr() and gdx() are started. The gdr() process receives all GDD mails and stores them in a circular mailbox buffer. The gdx() process executes the stored mails.
 - GDD_init() is called from gdr(), and the command GDD_INITIALIZE_PCMD is executed, by this cmd GDD driver is put in IDLE state.
 - GDD_lcd_init() is called by the application => the LCD panel is reset, LCD sub-module is put in the AWAIT_LCD_UPDATE state
 - GDD_lcd_update() is called by the application for the first time.

GDD Driver files (BP30)

- **gdd.c** Main control source file, including process control (GDR; GDX) and mailbox handling,
- **gdd.h** GDD header file with declarations of interface functions
- **gdd_lcd.c** LCD state machine, LCD command structure and all the LCD function
- **gdd_gdc.c** GDC state machine
- **gdd_gdctrl.c** GDC functions
- **gdd_ids183.h** Specific header file for Philips LCD
- **gdd_da8912a.h** Specific header file for Sharp LCD
- **gdd_ptest.c** Production and module test interfaces





GTL Driver files (BP30)

- **gtl.c** functions used in GTL initialization and profiles handling
- **gtl.h** GTL functions interface prototypes
- **gtl_i2c.c** GTL functions interface for I2C
- **gtl_i2c.h** types definition for I2C
- **gtl_ssc.c** GTL functions interface for SSC
- **gtl_ssc.h** types definition for SSC
- **gtl_timer.c** timers handling

Phone Tool GDD interface (BP30)

Phone Tool v21.01 (c) Infineon Technologies Denmark A/S - [Graphics device]

File Edit View Modes Trace Settings Help

BAR     AT on AT off

HW configuration LCD Interface

Get HW Type <- Click here to get information about supported hardware types.
!!! PLEASE MAKE SURE THE DUT IS STARTED UP IN PRODUCTION TEST MODE.

Hardware types & Capabilities **GDD API version: ?**

Hardware type

Right-Click on a hardware type to get the capability..

Capabilities	Value

Register control

☐ LCD (Main) ☐ Companion Chip
☐ LCD (Sub) ☐ Camera

	Register	Value
Read	(Hex) 0	0
Write	(Hex) 0	0

LCD Hardware ID

☒ Main LCD ID no:
☐ Sub LCD

Get ID

Phone Tool GDD interface (BP30)

Phone Tool v21.01 (c) Infineon Technologies Denmark

File Edit View Modes Trace Settings Help

HW configuration LCD Interface

LCD Type

☐ Main LCD ☐ Sub LCD

LCD setting

Contrast setting

	Main LCD	Sub LCD
?		
?		
Set value	0	0
Get Static NV	?	?
Store in EEP	0	0

Test image setting

	Main LCD	Sub LCD
Test Image	0	0

Pixel setting

	Main LCD	Sub LCD
X:	0	0
Y:	0	0
Data:	0	0

Set pixel