

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

## BP30 GDD Technical Specification

Edition 2006

Published by Neonseven s.r.l.,  
Viale Stazione di Prosecco, 15  
34010 Sgonico (Trieste) Italy

© Neonseven.  
All Rights Reserved.

For questions on technology, delivery and prices please contact the Neonseven Offices in Italy Sgonico and Gorizia

Attention Please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

### Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact Neonseven.

Neonseven technologies may only be used in life-support devices or systems with the express written approval of Neonseven, if a failure of such technologies can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Author	Maurizio Davanzo	Department:	S2	Page: 1/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –		Confidential	

## Table of Contents

<b>1</b>	<b>Document Mission/Scope .....</b>	<b>4</b>
1.1	Mission .....	4
1.2	Scope .....	4
<b>2</b>	<b>List of Acronyms .....</b>	<b>4</b>
<b>3</b>	<b>Introduction .....</b>	<b>4</b>
3.1	Overview .....	4
3.1.1	Configuration using LCD .....	5
3.2	Features .....	6
3.2.1	Function interface for LCD .....	6
3.2.2	High speed serial LCD interface .....	6
3.2.3	EGOLDradio HW interface .....	6
3.2.4	Multitasking .....	6
3.3	System requirements .....	7
3.3.1	Hardware requirements .....	7
3.3.2	Software requirements .....	7
3.4	Limitations and HW requirements .....	8
3.5	Module contents .....	9
<b>4</b>	<b>Architecture .....</b>	<b>9</b>
4.1	Context diagram .....	9
4.2	GDD Driver Function Interface .....	11
4.2.1	(a) Result messages through APOXI and result semaphores .....	11
4.2.2	(b) Generic return messages and callback functions (BP30) .....	12
4.3	GDD structure .....	12
4.3.1	GDD command structure .....	14
4.3.2	GDD Internal result handling .....	15
4.4	GDD process Layer .....	15
4.4.1	Sequence Diagram of GDR and GDX .....	16
4.5	GDD Middle layer .....	17
4.5.1	LCD .....	17
4.6	GDD Low Level Layer (GTL) .....	18
<b>4.6.1</b>	<b>GTL architecture .....</b>	<b>19</b>
4.6.1.1	Static view .....	19
<b>GTL driver API (Physical Driver API) .....</b>		<b>21</b>
<b>4.6.3</b>	<b>GTL driver (Physical Driver) .....</b>	<b>22</b>
4.6.3.1	Interaction with the upper layers .....	23
4.6.3.2	Internal handling of Profiles .....	25
4.6.3.3	Internal Handling of Hardware Resources .....	28
4.6.3.4	System Drivers .....	29
4.6.3.5	Hardware Resources supported by GTL .....	29

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

4.7	GDD start-up and shutdown .....	30
4.7.1	Start-up sequence .....	30
4.7.2	Shutdown sequence .....	30
4.8	Function interface flow example .....	31
<b>5</b>	<b>References .....</b>	<b>33</b>
5.1	External .....	33
5.2	Internal .....	33
<b>6</b>	<b>Document change report .....</b>	<b>33</b>
<b>7</b>	<b>Approval .....</b>	<b>33</b>
<b>8</b>	<b>Annex 1 .....</b>	<b>33</b>

Author	Maurizio Davanzo	Department:	S2	Page:	3/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

# 1 Document Mission/Scope

## 1.1 Mission

This document contains the specification of the Graphics Device Driver (GDD) module

## 1.2 Scope

This document is addressed to SW developers who want understand and adapt the GDD module for their HW platform.

# 2 List of Acronyms

Abbreviation / Term	Explanation / Definition
APOXI	Application Programming Oriented X Interface. Object oriented application
PEC	Peripheral Event Controller
SDL	Specification Description Language
GDD	Graphics Device Driver
GTL	Graphics Transport Abstraction Layer
MMI	Man Machine Interface

# 3 Introduction

This document describes the Graphics Device Driver (GDD) specialized for the BP30 platform. This GDD driver is designed for the E-GOLDradio GSM baseband

The document is split into two main chapters:

- 1. Introduction**, in this chapter there will be given an overview of the supported HW configurations, and the main features of the GDD driver will be presented.
- 2. Architecture**, in this chapter the architecture of the GDD driver will be presented. The architecture is split into three layers and a Driver Function Interface; each of the layers consists of one or more sub-modules which all will be described thoroughly.

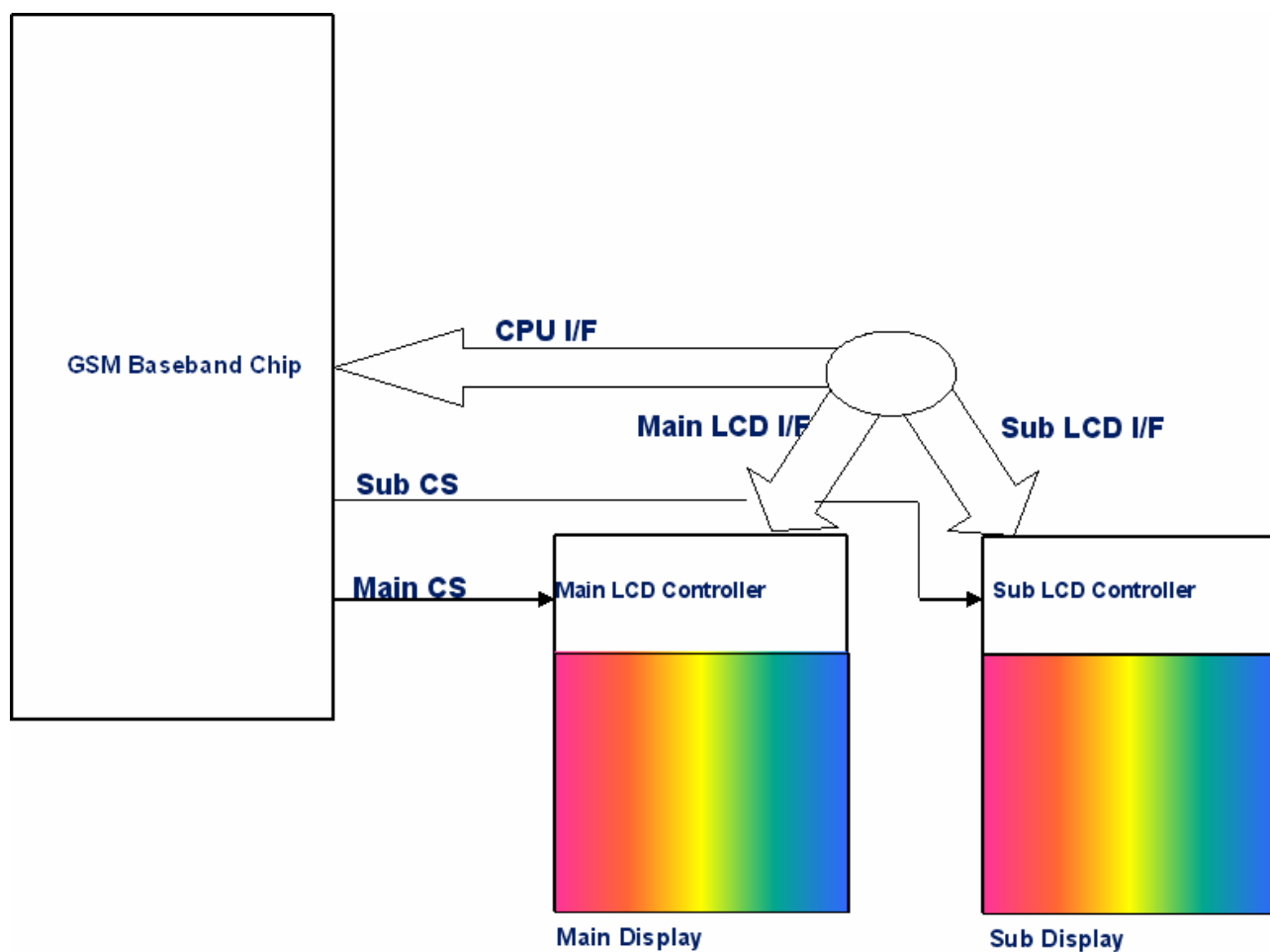
## 3.1 Overview

The N7 GDD driver is build on the generic GDD driver, which is prepared for supporting different kinds of graphics HW and is easily extended. Some of the main features of the N7 GDD driver are:

- Flexible HW interface to LCDs: Serial/Parallel, 8bit/16bit.
- High speed data transmission (PEC).
- LCD configuration.
- OSE operating system.

### 3.1.1 Configuration using LCD

N7 GDD driver is configured for applications with two display.



**Figure 3-1** Configuration LCD

Author	Maurizio Davanzo	Department:	S2	Page:	5/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

## 3.2 Features

### 3.2.1 Function interface for LCD

The N7 GDD driver is providing a function interface between the application and the graphics hardware for controlling the LCD. The main features are:

- Updating the LCD displays
- Setting the LCD contrast
- Power saving LCD
- 

### 3.2.2 High speed serial LCD interface

The GDD driver is supporting dedicated hardware channels of the GSM baseband chip for fast LCD updates. In the following section, the solutions for the E-GOLDRadio platform will be presented.

### 3.2.3 EGOLDRadio HW interface

The Peripheral Event Controller (PEC) works through the Synchronous Serial Interface (SSC) for serial connection (up to 26Mbaud).

### 3.2.4 Multitasking

The N7 GDD driver is designed to make use of OSE real-time operating. This allows the GDD driver to store the requested command in a local mailbox, thus minimize the time suspending the calling process. Further busy waiting is avoided by suspending the calling process while accessing the graphics hardware.

Author	Maurizio Davanzo	Department:	S2	Page: 6/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

### 3.3 System requirements

#### 3.3.1 Hardware requirements

##### E-GOLDradio

- Chip select logic (LCD controllers).
- Core timer

##### E-GOLDradio serial transfer

- PEC channel
- SSC

##### LCD controller

- On-chip display data RAM
- Auto-increment of address
- Possibility to read from display data RAM

#### 3.3.2 Software requirements

##### Supported operating systems

- OSE

##### Operating system resources

- Two processes with equal priority
- Interrupt routines for core timer.

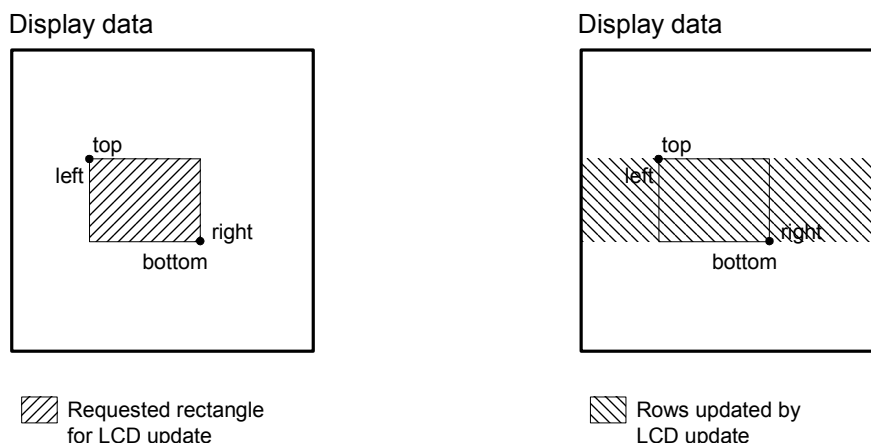
Author	Maurizio Davanzo	Department:	S2	Page: 7/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

### 3.4 Limitations and HW requirements

The GDD driver is implemented to support write register to the LCD controller.

As described earlier the GDD driver is using the PEC to gain high speed LCD updates. This means that only display controllers, which is capable of auto-increment the source and destination address, is supported.

The usage of the auto-increment of the source address combined with APOXI's LCD leads to the following restriction. When the application is requesting a LCD update of a rectangular area of the image mirror (`void GDD_lcd_update(.)`), not only the rectangular area is updated, but also the complete rows containing the rectangular area, see figure below.



**Figure 3-2** Display data writing

If this should be avoided the source address must be reprogrammed for each new line, which again would lead to a lot more PEC transfers and thus a more time consuming update.

After the application has requested an update of the LCD image, the application should not modify the data area of the affected rows of the image until the LCD update has finished, since there is no other protection of the APOXI mirror.



### 3.5 Module contents

The GDD driver is using the prefix “GDD” for any interface function name.

The source files of the N7 GDD driver are listed below.

File	Description
<b>gdd.c</b>	Main control source file, including process control and mailbox handling
<b>gdd.h</b>	GDD header file with declarations of interface functions
<b>gdd_lcd.c</b>	LCD controller source file
<b>Gdd_tfs6040.h</b>	Specific header file for the main LCD
<b>Gdd_tfs6040_cstn.h</b>	Specific header file for the sub LCD
<b>gdd_ptest.c</b>	Production and module test interface source file.
<b>gtl.c</b>	functions used in GTL initialization and profiles handling
<b>gtl.h</b>	GTL functions interface prototypes
<b>Gtl_ssc.c</b>	GTL functions interface for SSC
<b>Gtl_ssc.h</b>	types definition for SSC
<b>Gtl_timer.c</b>	timers handling

**Table 3-1** GDD source files

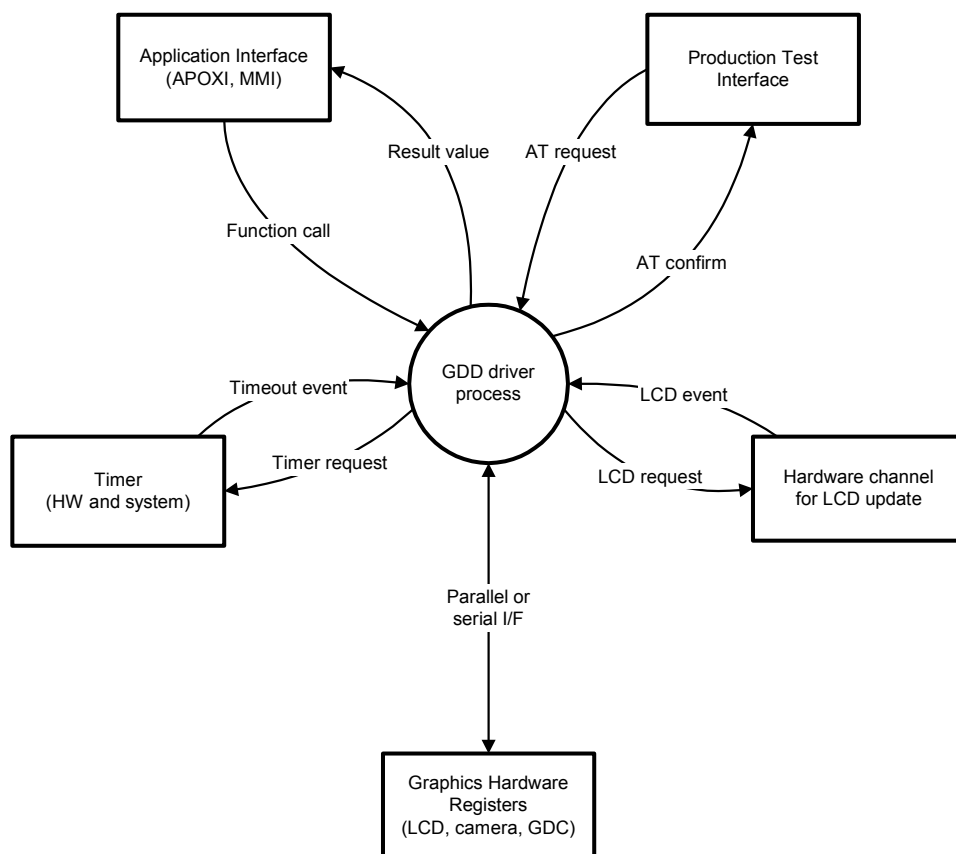
## 4 Architecture

### 4.1 Context diagram

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

The terminators of the GDD driver are shown in the figure below.

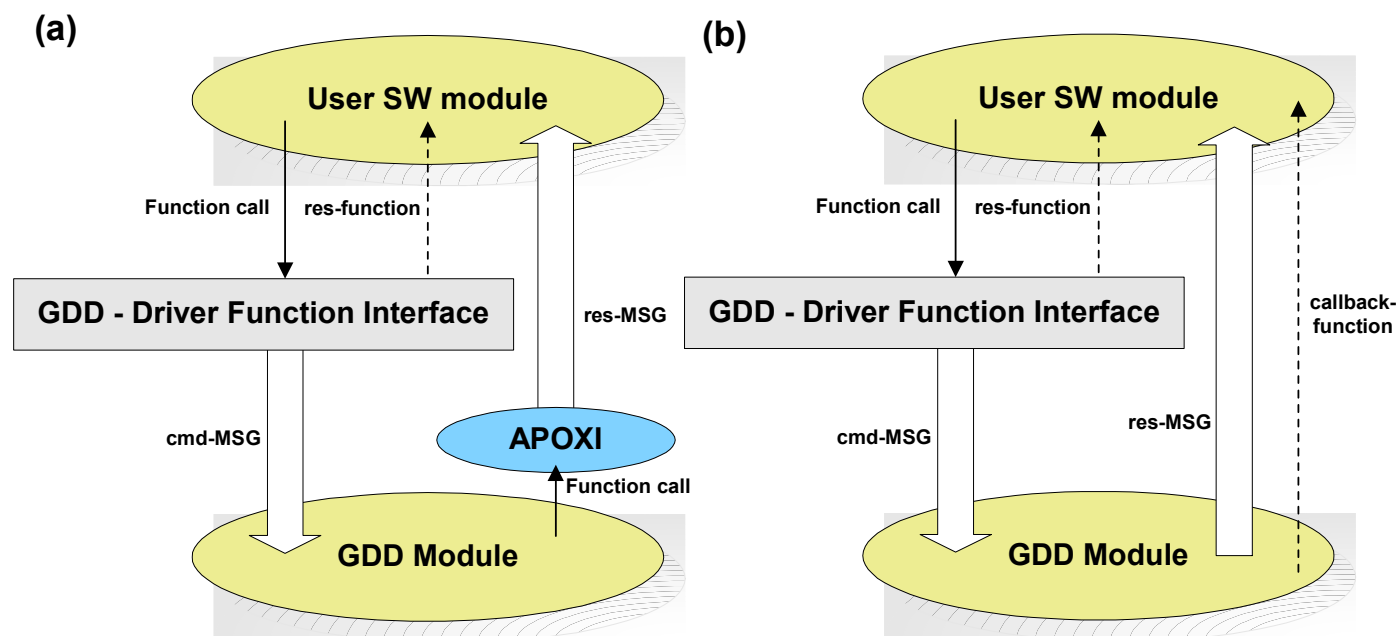


**Figure 4-1** GDD terminators

Author	Maurizio Davanzo	Department:	S2	Page:	10/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

## 4.2 GDD Driver Function Interface

The interaction of the GDD module, the Driver Function Interface and the User SW module is shown in the figure below.



**Figure 4-2** GDD module interaction

As shown in the figure there is two different solutions, which one is used depends on the project. The two solutions will be described separately in the following sections. In the Interface Specification it will be noted which return type is used for the actual project.

### 4.2.1 (a) Result messages through APOXI and result semaphores

The User SW module is using the Driver Function Interface by calling one of the interface routines as indicated by a solid black line (Function call).

The Driver Function Interface will generate a message with the requested command and parameters and send this to the GDD module, where the command will be executed; this is indicated by a solid white arrow (cmd-MSG). Some interface functions has a pointer as input where a result must be stored. In such cases the Driver Function Interface will hold back the User SW module until the result has been stored by the GDD module, this is indicated as a dotted black line (res-function). The hold back mechanism is implemented by using a semaphore, the *gdd\_api\_result\_semaphore*. For other functions, where the timing is critical for the User SW module, a return signal is used instead; this is indicated as a solid white arrow (res-MSG). This is for instance used for LCD updates, where the MMI only is hold back until the cmd-MSG has been send by the Driver Function Interface. When the update is completed, a message is send to the User SW module; this has been implemented by GDD calling an APOXI callback function, which will send a message to the right application

The GDD driver is using two return signals, which are defined in the APOXI by the callback functions:

- `ApoxiSendDisplayUpdateFinished()`
- `ApoxiSendGddViewfinderError()`

Author	Maurizio Davanzo	Department:	S2	Page:	11/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

ApoxiSendDisplayUpdateFinished() is called by the GDD driver whenever a requested LCD update has executed in order to send a signal indicating that the User SW module may go on modifying the affected rows in the display image in preparation for the next LCD update

#### 4.2.2 (b) Generic return messages and callback functions (BP30)

In this solution, there is three different ways of returning a result from GDD to the User SW Module:

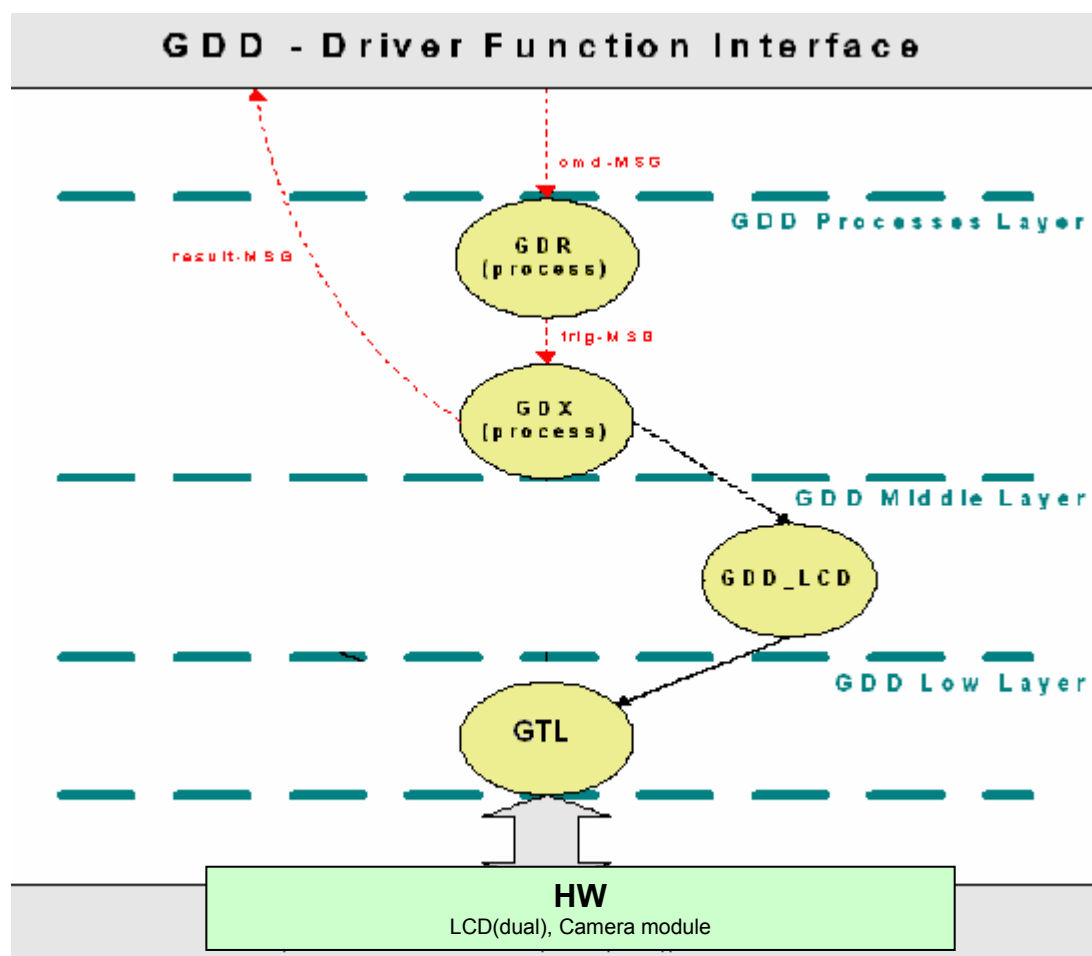
1. The function will return immediately if no interaction with GDD is needed, this is for instance used for GDD\_lcd\_get\_contrast\_limits().
2. A generic return message is send when the result is ready, or the command has been executed. This is used for functions, which require interaction with GDD and is not time critical.
3. A callback function is called in the User SW module, when the result is ready, or the command has been executed. The callback function will be specified by an input parameter in the GDD function. This is used for functions, which require interaction with GDD and is time critical.

In BP30 project the interface has been configurated as explained in b) case and only the first and the third interaction has been implemented.

### 4.3 GDD structure

In the figure below the interaction between the GDD Driver Function Interface, GDD sub-modules and HW is illustrated.

Author	Maurizio Davanzo	Department:	S2	Page: 12/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential



**Figure 4-3** Interaction between GDD Driver Function, GDD sub module and HW

The GDD driver consists of three layers, indicated by dashed green lines, and a Driver Function Interface, illustrated as a gray box in the top:

- **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. These two processes are the same for all projects.
- **GDD Middle Layer**, each one of the middle layer modules abstracts a Graphical HW block (which in BP30 project is only LCD). The module contains a state machine and all possible commands for that particular HW block. Depending on the chosen HW configuration for a specific project these modules are replaced with the one matching the chosen HW. It should be noted that the middle layer modules also contains the Driver Interface functions provided by the GDD driver. In BP30 project all commands sent by GDX go directly to the LCD state machine.
- **GDD Low Layer (GTL)**, the low level module is responsible for interfacing with the HW on the BP30 platform. (the chapter 4.6 is completely dedicated to explain the GTL architecture )

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)” as indicated by a dashed red line. The GDR process will trigger the GDX module when a message has been received. The executer sub module “GDX (process)” then executes the message by calling the LCD state machine.

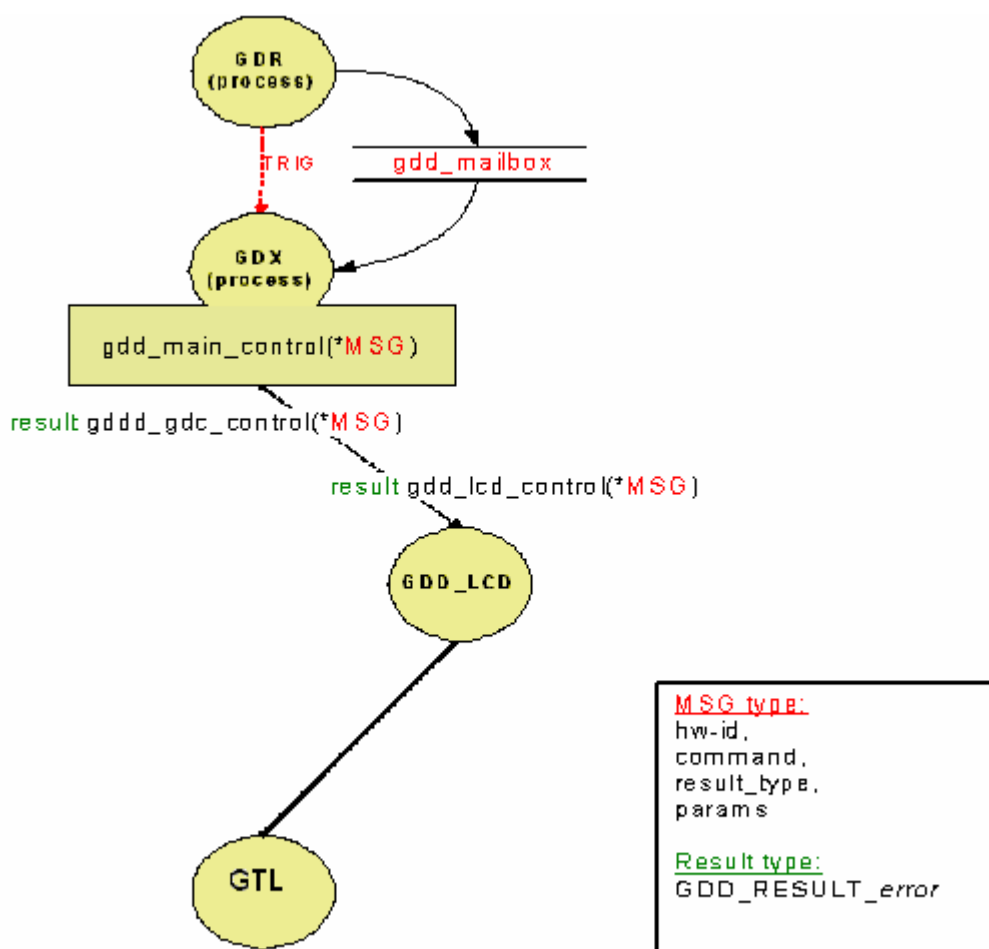
Author	Maurizio Davanzo	Department:	S2	Page:	13/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

### 4.3.1 GDD command structure

The entire GDD driver is based on a message interface, where the messages send from the Driver Function Interface contains one single command and some parameters. The interaction between GDR and GDX (and the *gdd\_mailbox*) will be described in the next chapter.

The GDX contains the `gdd_main_control()` function. `gdd_main_control()` will forward the incoming message to the LCD sub-module.

This is illustrating in the next figure, the function inputs are marked with red:



**Figure 4-4** Messages interface

Beside the state machine, the middle layer module contains a command control structure, which contains all the possible commands for that specific module.

An important feature of the chosen structure is that all incoming requests are handled sequential, where one command is executed at a time. I.e. it is, for instance, not possible to send a new LCD update while the previous one is ongoing, the last command will be queued until the first has finished. This is realized by using a so-called *gdd\_access\_semaphore*, which will be more toughly described.

Author	Maurizio Davanzo	Department:	S2	Page:	14/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

The interface function only sends a single message containing a single command, if a command requires different commands this is controlled by the LCD state machine.

### 4.3.2 GDD Internal result handling

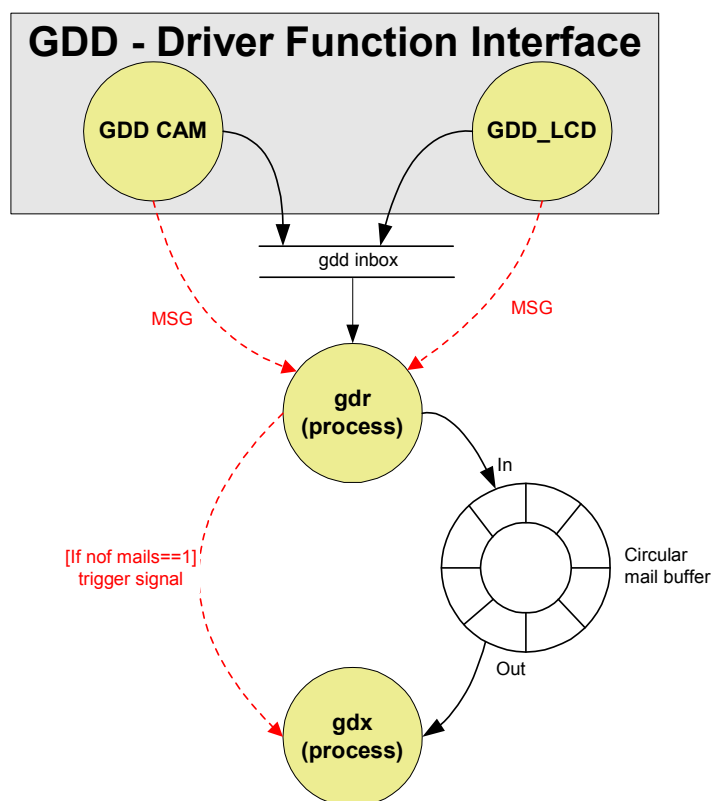
As described in section 4.2 the GDD driver handles results in several different ways:

- (a) The application is hold back until the result is ready. This is typically functions, which do not require actual activation of GDD, such as LCD\_get\_contrast\_limits().  
These functions will all return a *gdd\_result\_type*.
- (b) GDD will release the application process and send a result message when finished.  
These functions will all return *void*.
- (c) GDD will call a callback function directly in the application.

All internal GDD functions are returning *gdd\_result\_type*, where GDD\_RESULT\_OK indicates that the requested action was successful; all other values are indicating an error (GDD\_RESULT\_error). The *gdd\_result\_type* is used and checked through all the sub layers from the GTL module and returned to the GDX. In case (a), GDX will store the result in the *gdd\_api\_result* buffer from where GDR will fetch it and return it to the User SW module. In case (b) GDX will put the result in the return message as parameter 2.

## 4.4 GDD process Layer

The interaction between the Driver Function Interface, the GDR process and the GDX process is illustrated in the figure below:



Author	Maurizio Davanzo	Department:	S2	Page:	15/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

**Figure 4-5** Interaction between Driver Function, GDR process and GDX process

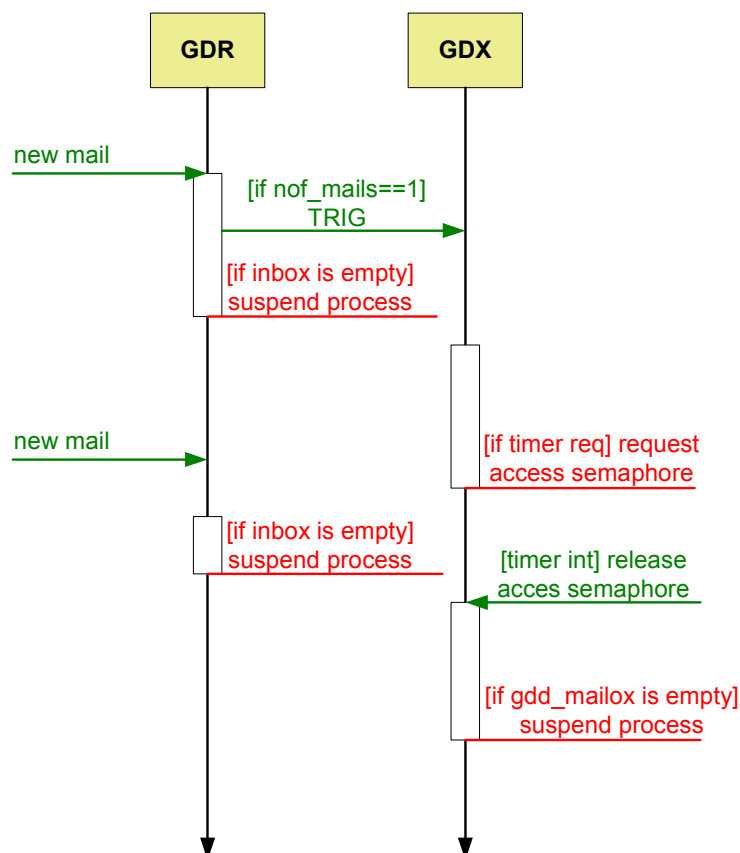
The GDD driver consists of two processes GDR and GDX. GDR is responsible for receiving messages from the Driver Function Interface through a common inbox called *gdd\_mailbox\_params*. When a message is received, it is copied to the circular mail buffer (*gdd\_mail\_box*) and the calling application 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.

The GDX process is responsible for executing all the messages in the circular mail buffer. It is triggered from the GDR process when the circular mail buffer contains exactly one mail and will continue to run until the buffer is empty, i.e. until all requested commands has been executed.

As GDR and GDX share the circular mail buffer, they are set to run at the same priority in order to protect this buffer. An example of when which process runs is given the section 4.4.1.

#### 4.4.1 Sequence Diagram of GDR and GDX

A possible sequence diagram of GDR and GDX is shown in the figure below; it should be noted that the other actors such as the User SW Module, the RTOS and the interrupt service routine is not shown in this figure.





**Figure 4-6** Sequence diagram of GDR and GDX

If there are no messages in either mailbox both processes will be suspended. When a new message is sent from the Driver Function Interface the 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.

## 4.5 GDD Middle layer

In BP30 project GDD driver contains only one module:

- GDD\_LCD

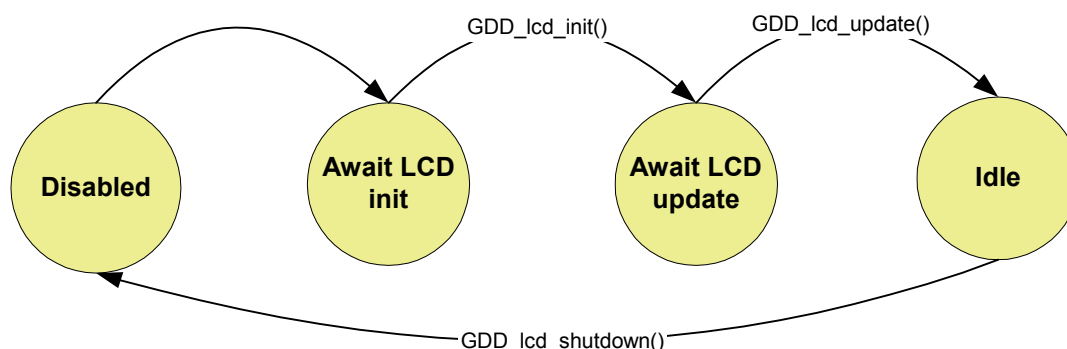
The LCD module contains a state machine and a command structure, which contains all the possible commands for the specific module. The state machines is directly called from `gdd_main_control()`, which is executed by GDX.

### 4.5.1 LCD

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

- LCD controller initialization.
- Contrast control.
- Power save handling of LCD controller.
- Shutdown control.
- LCD update of rectangular areas from APOXI virtual RAM.
- 

The state machine of the LCD sub-module is shown in the figure below.

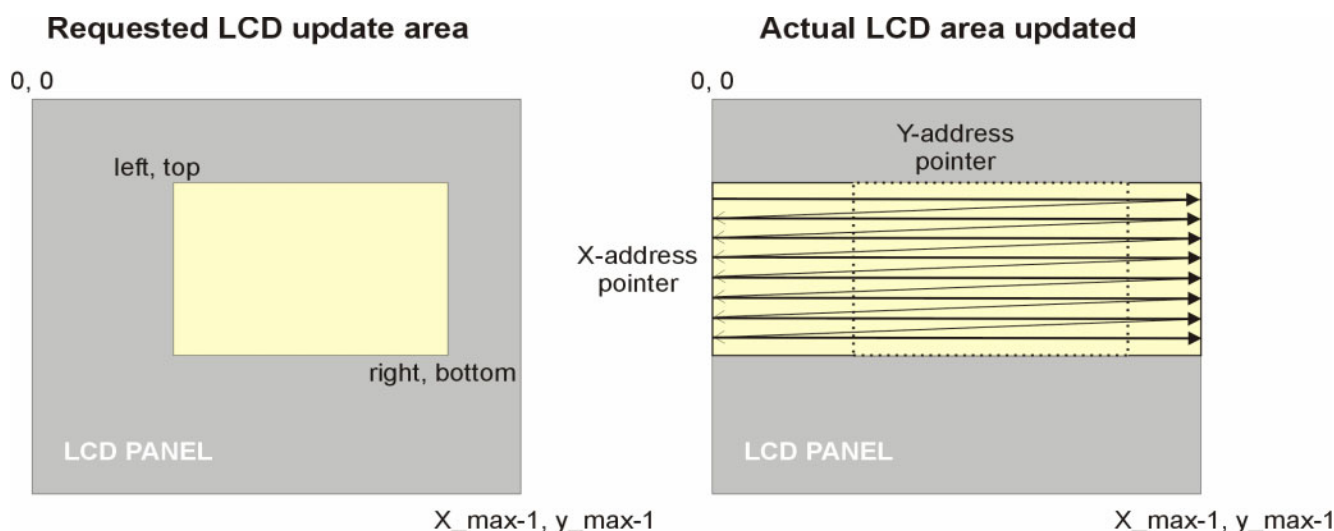


**Figure 4-8** LCD state machine

The LCD will be turned on when switching state from “Await LCD update” to “Idle”; this enables the User SW Module to set the contrast before anything is shown in LCD.

#### 4.5.1.1 LCD update

The LCD panel is operating by internal controller X- and Y address pointers, which controls the area of the panel to be updated. These pointers are set by the driver, and they are derived from the input parameters of the LCD update request. Prior to an update of the display area these pointers are programmed and another set of parameters are programmed to hold the four coordinates identifying the rectangle to be updated. Once those parameters are programmed within the LCD controller, the display data to follow will be copied to the display RAM area identified by the coordinates programmed. It's important to notice that when the controller has received one row of data, then the X- and Y address pointers will automatically be incremented by the LCD controller itself, i.e. they are set for the next row and column position (relative column "0").



**Figure 4-9** LCD update

Thus when the User SW module is requesting a LCD update of a rectangular area of the image, not only the rectangular area is updated, but also the complete rows containing the rectangular area, see figure above.

### 4.6 GDD Low Level Layer (GTL)

In N7 GDD the Low Level Layer is implemented by GTL driver.

GTL can handle all the multimedia hw resources and offers to the upper layer (GDD middle layer) a simple and usefull interface.

Each one of Graghich HW block which ,in BP30 project is only LCD, is rappresented in GDD Middle Layer with a module (and with the corresponding state machine).

In GTL these modules are named *user cliet* and they rappresent the clients who need to acces to the Hw resources.

GTL has been made with a flexible structure in order to handle many hw resources and many user client without conflicts.

In BP30, only one user client is defined (the LCD module) and it must communicate with the LCD Hw resource The Gtl architecture will be explained in the following chapters.

Author	Maurizio Davanzo	Department:	S2	Page:	18/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

## 4.6.1 GTL architecture

### 4.6.1.1 Static view

Below in Figure -- the static view of the GTL driver is given.

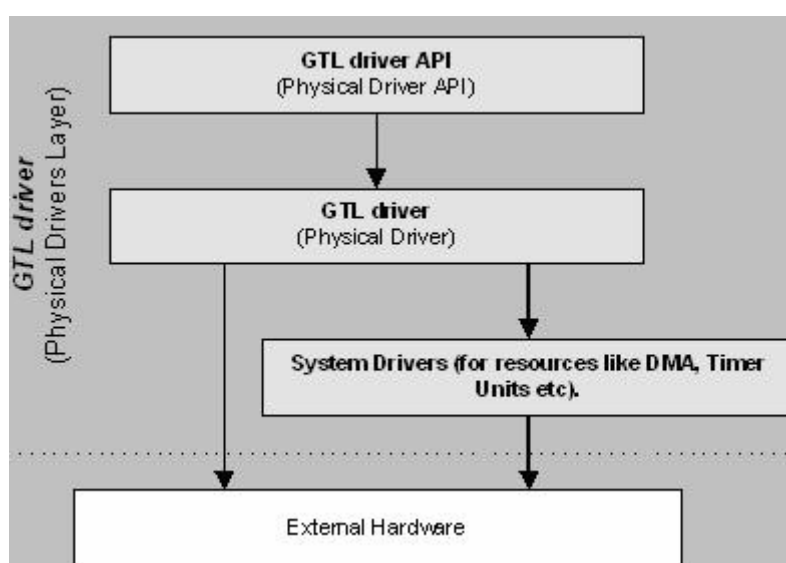


Figure 1: Static view of GTL driver

- GTL Driver API Layer (Physical Driver API)**  
 It is the GTL interface layer.  
 The interfaces provided by this layer are intended to be robust to future hardware changes. All the hardware specific parts are encapsulated in the layers below it.
- GTL Driver Layer (Physical Driver)**  
 This layer implements the GTL driver.
- GTL platform dependent device control (System Drivers)**  
 These are the drivers for the system resources that are required by other hardware blocks. Examples of these drivers would be the DMA driver, timer unit driver, I2C driver etc.

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

Author	Maurizio Davanzo	Department:	S2	Page: 20/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

## GTL driver API (Physical Driver API)

The GTL driver offers a generic API towards the *user clients* (GDD middle layer), abstracting the hardware dependent platform devices being used.

The platform device interface functions are exported through the standard device driver interface functions:

- ***open***  
The user clients use this interface to open an active connection with the GTL driver layer below.
- ***close***  
The user clients use this interface to close an active connection with the GTL driver layer below.
- ***write\_register***  
The user clients use this interface to write to a particular register of the external hardware. This interface writes a single <register index, data> to the external hardware.
- ***write\_register\_tab***  
The user clients use this interface to write to a table of registers of the external hardware. This interface is to be used when there is 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.

Author	Maurizio Davanzo	Department:	S2	Page: 21/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

### 4.6.3 GTL driver (Physical Driver)

The GTL driver maintains a central database (*ssc\_profiles\_db*) that contains information about each active user client connection. The size of this database imposes a restriction on the number of active connections with the GTL driver (i.e. share the hardware). This database is where the information for each *user client* that has an active connection is stored. Each such information per active connection is henceforth referred to as the *profile*. The information in each *profile* can be classified as:

- **User client specific information**

The user clients supply this information during an *open* call. This information is referred to as the *Device Object* information henceforth. The *Device Object* information relates to:

1. Identify the hardware resource to be used by the user client. The GTL driver could be handling multiple hardware resources in which case the *user client* needs to specify the hardware resource to be used.
2. Information required by the driver for interacting with the underlying hardware as well as the external hardware. The underlying hardware information depends on the point (1) above. The external hardware information that is required is limited to the information that needs to be supplied for proper communication like timing, register width, endian etc.
3. Callback functions to be called for specific events. The intention of providing callback functions is to enable *user clients* to perform operations on an event and also to abstract certain external hardware interconnection information from the GTL.
4. Whether the *user client* wants to use the hardware in exclusive mode or in shared mode. If a *user client* opens an active *profile* in exclusive mode, subsequent open requests to the GTL would fail. Also, an open in exclusive mode will fail if there exist active *profiles* in shared mode. In addition to the above two modes, a *blocking* mode is provided to an active *profile* in shared mode to carry out read/write operations without any interference from other active *profiles*.
5. An identifier, which uniquely recognises the active connection of the *user client* with the GTL driver.

- **Internal information**

This information is specific to the driver.

Author	Maurizio Davanzo	Department:	S2	Page: 22/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential

The state machine below shows the different state transitions of a *profile*:

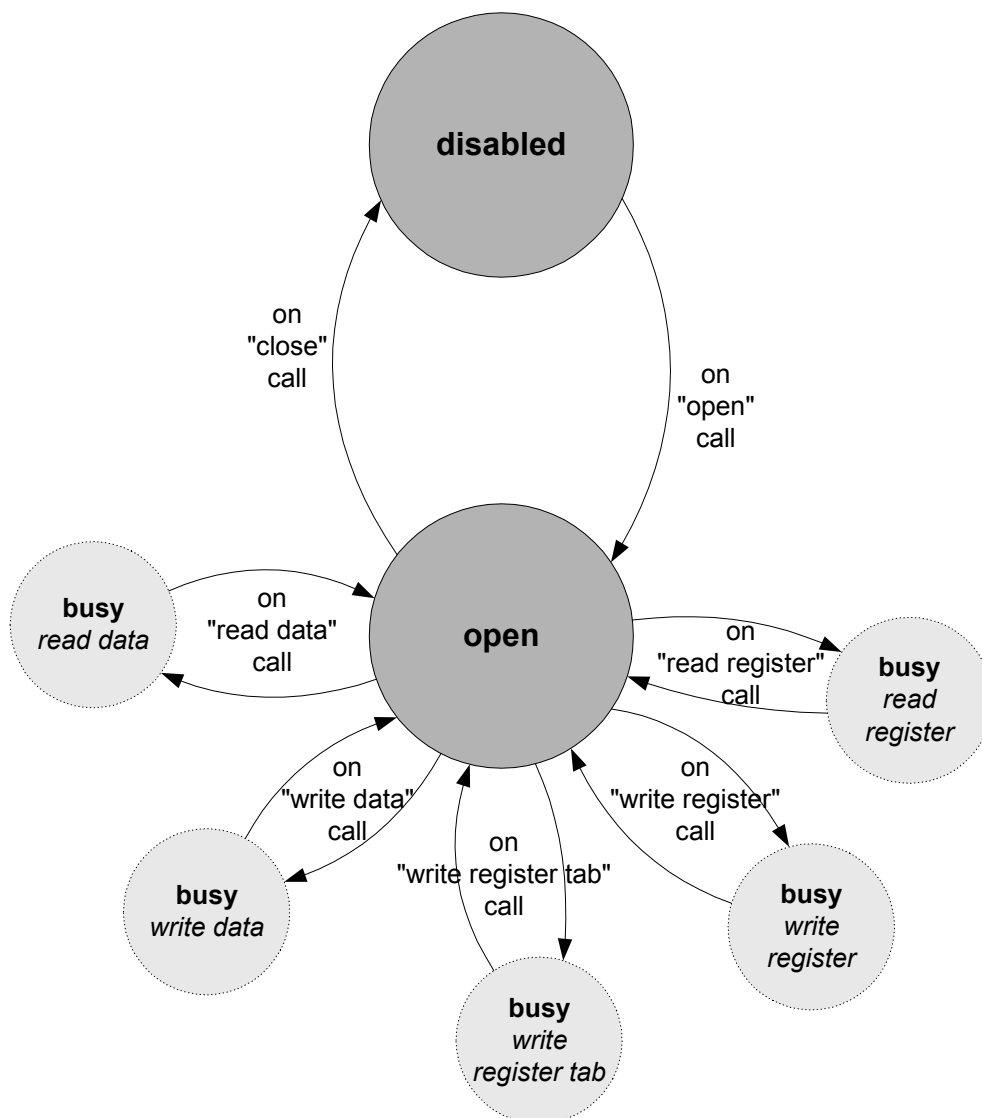


Figure 2: GTL driver profile state machine

The states in dotted circles are only transient states. The driver returns to the OPEN state as soon as it is done performing the required actions.

#### 4.6.3.1 Interaction with the upper layers

The GTL driver interacts only with the GTL driver API layer above. All the *user client* requests interface with the GTL driver API layer. The diagram below shows this interaction scenario.

Author	Maurizio Davanzo	Department:	S2	Page:	23/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

Multiple *user clients* can have an active *profile* with the GTL driver (In BP30 only one profile is active). The GTL driver allows a maximum number of active *profiles* at any point of time. Once this maximum number of active *profiles* is reached, subsequent requests for an active *profile* by a *user client* (using the *open* interface) will be returned as a failure. Further active *profiles* can be opened only after any of the currently active *profile* is closed. In the figure below, this maximum number is indicated as “n”.

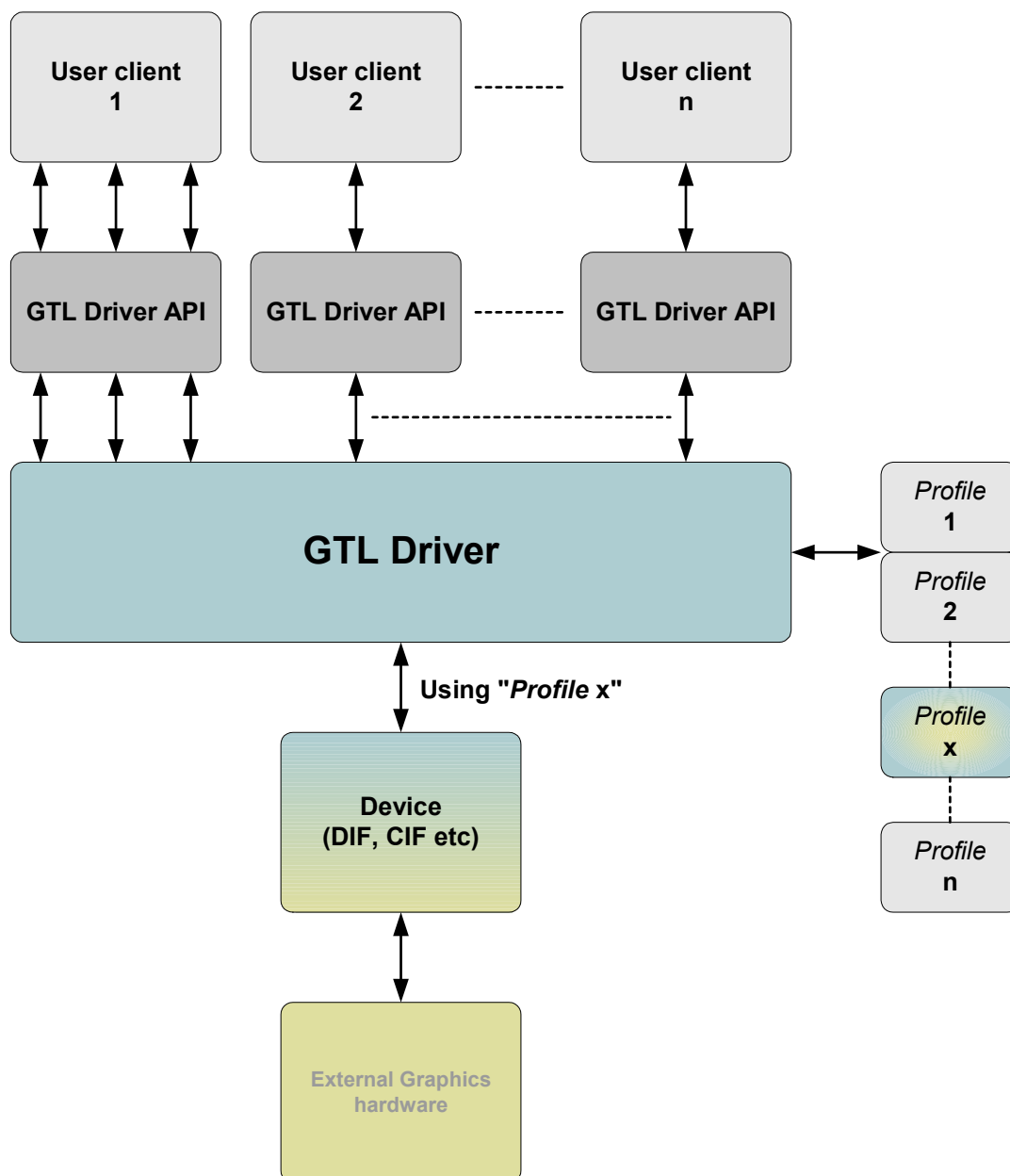


Figure 3: GTL driver profile management

The user client creates an active *profile* using the *open* interface.

The GTL driver API passes this request to the GTL driver. If the maximum number of active *profile* is not already open, the GTL driver successfully acknowledges this request. The information of this user client is stored in a unique index in the database. This unique index is henceforth used by the GTL driver to identify further requests from this user client. The user client closes this active *profile* using the *close* interface.

Author	Maurizio Davanzo	Department:	S2	Page:	24/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	



	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

Note:

- In the figure above, *user client-1* has three active *profiles* open to the GTL driver.

The GTL driver handles the resource sharing internally. Hence, at any point of time only one *user client* can gain access to the underlying hardware (DIF, CIF etc). In the figure above, *user client "X"* has the access to the underlying hardware. The GTL driver uses the active *profile* corresponding to this *user client* for the communications (*Profile X*).

#### 4.6.3.2 Internal handling of Profiles

In BP30 project there is only one active profile so this paragraph has been inserted only for future developed of GDD driver.

The GTL driver handles the hardware resource contention and sharing internally. The *user clients* specify the mode in which the hardware resource needs to be used. These modes, specified in the *Device Object*, can be either *exclusive* or *shared*. If any *user client* were using the hardware resource in exclusive mode, it would prevent any other active profiles from being used. Hence, resource contention and sharing needs to be handled only when there are no exclusive mode requests, i.e. all active *profiles* are using the hardware in shared mode. The exclusive mode will minimize the *profile* configuration overhead required for each operation performed using the hardware resource.

For any active *profile* in shared mode, the GTL driver provides a *blocking* mode wherein the hardware resource is exclusively allocated to this active *profile*. This mode should be used by the *user clients* only for the duration of a critical read/write operation where it is important to ensure that the actions performed by this active profile are not un-done by other active *profiles*. In addition, it has all the benefits of the exclusive mode also.

The GTL driver implements this resource contention and sharing using a semaphore. For the duration a request is being processed by the GTL driver using the hardware resource, other requests are blocked by this semaphore protection. Which particular pending request would get the hardware next is determined by the priority of the task. The highest priority task would get the hardware resource next.

The diagrams below, illustrates the resource-sharing schemes described above.

Author	Maurizio Davanzo	Department:	S2	Page: 25/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential

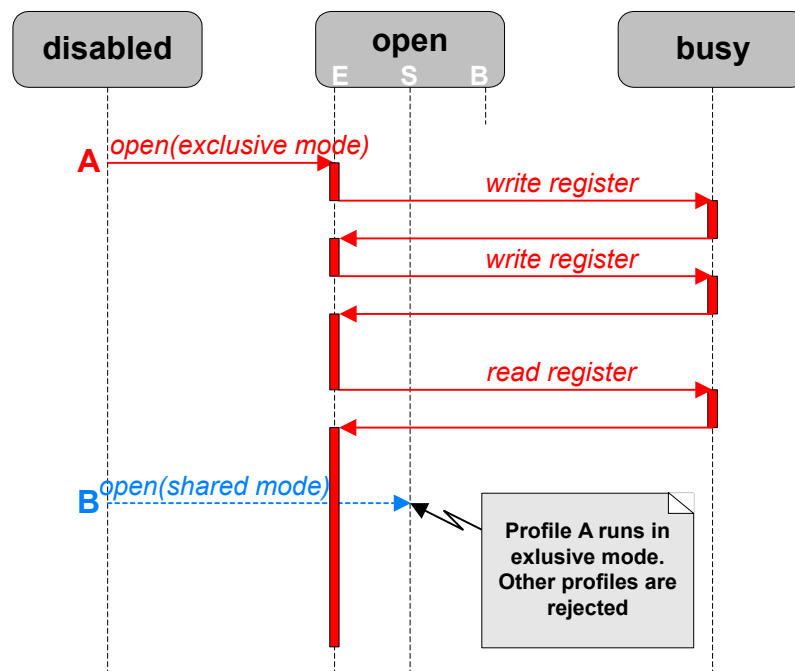


Figure 4: Resource contention and sharing – Exclusive mode



#### 4.6.3.3 Internal Handling of Hardware Resources

User clients access the hardware resources through the GTL driver APIs *write\_reg*, *write\_reg\_tab*, *read\_reg*, *write\_data* and *read\_data*. Each hardware resource internally has its own set of above functions. When the user client opens an active profile with the GTL using the *open* interface, the GTL driver returns the appropriate driver functions to use in the *driver\_funcs* member of the *Device Object* structure (i.e. *gtl\_device\_object\_type*).

The internal architecture of the GTL driver is illustrated in **Figure 6**.

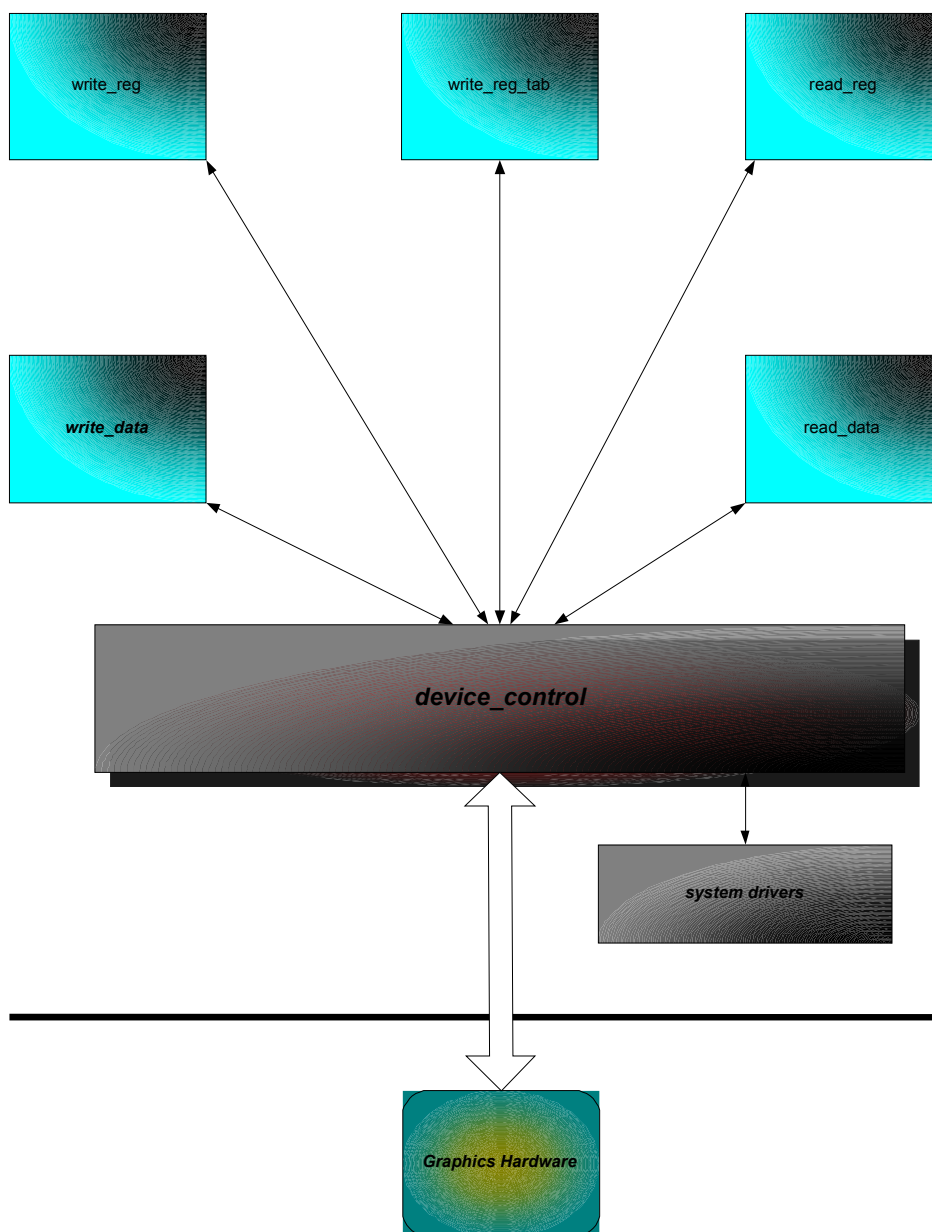


Figure 6: GTL, internal architecture

The GTL driver performs the following functions:

Author	Maurizio Davanzo	Department:	S2	Page:	28/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

- It configures the *hardware resource* as per the configuration passed by the *user client* through the *Device Object* structure.
- It performs the required action, i.e. write registers or read registers etc.,
- It handles resource sharing through the usage of
- It interacts with the appropriate system drivers (see section 4.6.3.4) to perform the requested action.

#### 4.6.3.4 System Drivers

The GTL driver interacts with the system drivers for using the various system resources required to perform the operations. The system resources used by the GTL driver are:

- PEC resource:
- Timer Units like GPTU.
- Capture Compare Unit
- Interrupt Controller Unit.
- I2C driver.

#### 4.6.3.5 Hardware Resources supported by GTL

##### SSC

- Support for PEC transfer.
- Data transfer is supported only in the transmit direction. The data transfer is done using the PEC.
- Register read and write supported.

##### Timer

- Using CAPCOM.

Author	Maurizio Davanzo	Department:	S2	Page: 29/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

## 4.7 GDD start-up and shutdown

### 4.7.1 Start-up sequence

The GDD driver is initialized as follows.

- `GDD_init_rtos_resources()` is called in order to initialize the semaphores used to protect critical code sections from concurrent execution (so called mutual exclusion).
- 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. As a result of the `GDD_INITIALIZE_PCMD` command, the graphics hardware is initialized and the GDD driver is put in IDLE state, waiting for the application to call `GDD_lcd_init()`.
- `GDD_lcd_init()` is called by the application. As a result the LCD panel is reset, the LCD controller is initialized and the `GDD_LCD` sub-module is put in the `AWAIT_LCD_UPDATE` state, waiting for the application to call `GDD_lcd_update()`. This allows the application to set the contrast before any image is displayed on the LCD panel.
- `GDD_lcd_update()` is called by the application for the first time. As a result, the LCD panel is turned on and the requested image is displayed. The GDD driver is put in IDLE state.

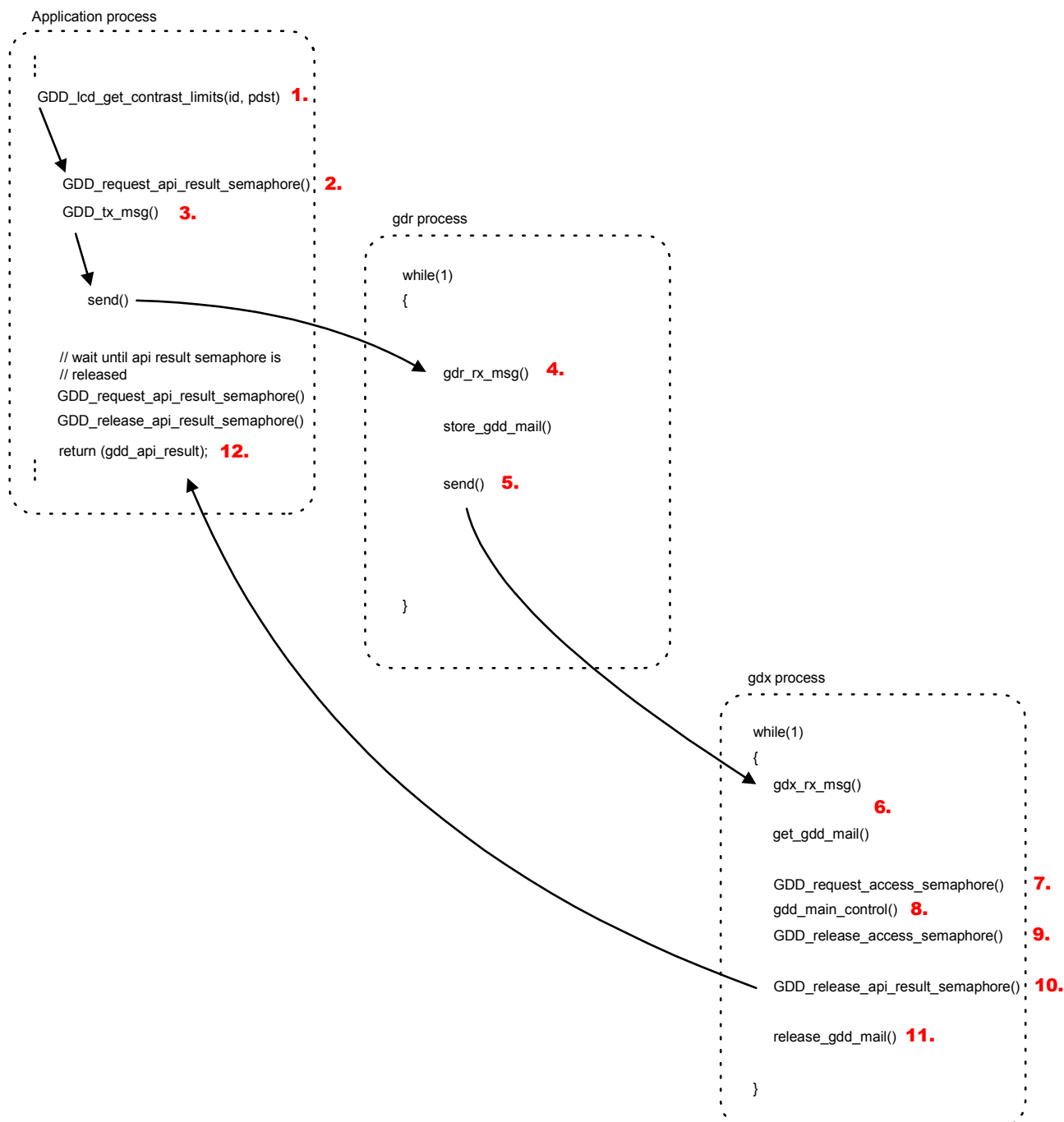
### 4.7.2 Shutdown sequence

The GDD driver is shutdown as follows.

- The shutdown sequence is triggered by the `GDD_lcd_shutdown()` function.
- The `GDD_LCD_SHUTDOWN` command is sent to the LCD sub module, which will execute the LCD shutdown procedure. The state LCD state is switched to DISABLED.

## 4.8 Function interface flow example

This section demonstrates the internal program flow of the GDD driver when the application is calling an interface function. This program flow is generic for all interface functions, and is illustrated below by `GDD_lcd_get_contrast_limits()` as an example. The GDD driver is assumed to be in IDLE state.



**Figure 4-11** Function interface

Author	Maurizio Davanzo	Department:	S2	Page:	31/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	<h1>Technical Specification</h1>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	----------------------------------	---

1. The application is calling an interface function, e.g. `GDD_lcd_get_contrast_limits()`.
2. The API result semaphore is requested, as the application process must wait for the result of the interface function. In this example, `GDD_lcd_get_contrast_limits()` returns `GDD_RESULT_OK` if the GDD driver is initialized and the LCD contrast limits are copied using the destination pointer (`pdst`).
3. The parameters (`id` and `pdst`) are copied to the global variable `gdd_mailbox_params` and the mail is transmitted to the GDD mailbox by calling `GDD_tx_msg()` with the appropriate command, e.g. `GDD_LCD_GET_CONTRAST_LIMITS_CMD`.
4. The GDD mailbox receiver process `gdr()` is awoken from `gdr_rx_msg()`, and the mail is stored in the GDD mailbox by calling `store_gdd_mail()`.
5. The `gdr()` process then sends a trigger signal the GDD mailbox executer process `gdx()` and starts waiting for the next mail.
6. The `gdx()` process is awoken from `gdx_rx_msg()`, and the mail is fetched by calling `get_gdd_mail()`.
7. The access semaphore is requested to ensure that only one process is accessing the graphics hardware at the same time.
8. The mail is executed by calling `gdd_main_control()` which will forward it to `GDD_lcd_control()` which fills in the minimum, maximum and default contrast settings of the LCD.
9. The access semaphore is released.
10. The result of the mail is copied to the global variable `gdd_api_result` and the API result semaphore is released.
11. The mail is released by calling `release_gdd_mail()`.
12. Finally, the result is returned to the application.

Author	Maurizio Davanzo	Department:	S2	Page: 32/33
Filename	BP30_GDD_Technical_Specification.doc			
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential



	<b>Technical Specification</b>	Doc. ID: AH01.SW.TS.000015 Rev.:1.1 Date:13/01/2006
---	--------------------------------	---

## 5 References

### 5.1 External

E-GOLDradio GSM/GPRS Baseband System – PMB 7870 – Design Specification – 2005-08-02 Rev. 1.05 – Infineon Technologies

### 5.2 Internal

Title	Doc ID
BP30 GDD Interface Specification	AH01.SW.TS.000016

## 6 Document change report

Change Reference			Record of changes made to previous released version	
Rev	Date	CR	Section	Comment
1.0	25/06/2004	N.A	Document Created	
1.1	13/01/2006	N.A.	Document updated to BP30 Platform	

## 7 Approval

Revision	Approver(s)	Date	Source/signature
1.0	Stefano Godeas	25/06/2004	Document stored on server
1.1	Stefano Godeas	13/01/2006	Document stored on server

## 8 Annex 1

None

Author	Maurizio Davanzo	Department:	S2	Page:	33/33
Filename	BP30_GDD_Technical_Specification.doc				
M06-N7 Rev. 2	Copyright (C) 2006NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	