HVPLUS02

Software Architecture Design

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Page(s)** | **Contents** |
| A |  | All | First official release |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 11

1.1 Purpose 11

1.2 Scope 11

1.3 Definitions and Acronyms 11

1.4 Applicable Documents 11

2. SOUP Items 12

3. Architecture overview 13

4. Block diagram 14

4.1 Apalis Imx 6 description 16

4.2 LPC description 16

4.3 Buzzer description 16

4.4 LED description 16

4.5 Sensor description 17

4.6 Motor description 17

4.7 Valve description 17

4.8 Power description 17

4.9 USB description 17

5. LPC Task Architecture 18

5.1 Update Task 20

5.2 IPC Task 21

5.2.1 Priority: TBD 21

5.2.2 Delay time: TBD 21

5.2.3 Size of task: TBD 21

5.2.4 Describe: 21

5.2.5 Initialization 21

5.2.6 Interfaces 21

5.3 Alarm Task 21

5.3.1 Priority: TBD 21

5.3.2 Delay time: 100 ms 21

5.3.3 Size of task: TBD 21

5.3.4 Describe: 21

5.3.5 Initialization 22

5.3.6 Interfaces 22

5.4 PD Task 23

5.4.1 Priority: TBD 23

5.4.2 Delay time: 0 ms 23

5.4.3 Size of task: TBD 23

5.4.4 Describe: 23

5.4.5 Initialization 25

5.4.6 Interface 25

5.5 BD Task 25

5.5.1 Priority: TBD 25

5.5.2 Delay time: 5ms 25

5.5.3 Size of task: TBD 25

5.5.4 Describe: 25

5.5.5 Initialization: 26

5.5.6 Interfaces 26

5.6 CBIT Task 27

5.6.1 Priority: TBD 27

5.6.2 Delay time: TBD 27

5.6.3 Size of task: TBD 27

5.6.4 Describe: 27

5.6.5 Initialization 28

5.6.6 Interface 28

5.7 Diagnostic Module 28

5.7.1 Priority: TBD 28

5.7.2 Delay time: TBD 28

5.7.3 Size of task: TBD 28

5.7.4 Describe: 28

5.7.5 Initialization 29

5.7.6 Interfaces 29

5.8 Monitor Watchdog Module 29

5.8.1 Priority: TBD 29

5.8.2 Delay time: 10ms 29

5.8.3 Size of task: TBD 29

5.8.4 Describe: 29

5.8.5 Initialization 29

5.8.6 Interface: TBD 30

6. Implementation Constraints 31

6.1 Programming Language 31

6.2 Memory Utilization 31

6.3 Task Starvation and Deadlock Protection 31

6.4 Handling of Unanticipated Conditions 31

List of Figures

Figure 1 System block Diagram 11

Figure 2 – LPC system diagram 13

Figure 3 – LPC update progress Diagram 15

Figure 4 – IPC Task Diagram 16

Figure 5 – Alarm Task Diagram 17

Figure 6 – Patient data task diagram 19

Figure 7 – Cbit Task Diagram 22

Figure 8 – Diagnostic task diagram 24

List of Tables

Table 1 – Soup Items List 9

# Introduction

## Purpose

This document specifies the LPC system architectural design that will be used to implement and perform verification.

This document is intended to satisfy the requirement for a Software Architectural Design Document identified in the *BSEN 62304: 2006, Medical Device Software- Software life-cycle Processes*, section 5.3.

## Scope

The subsystem decomposition and software architecture described in this document will be used to write Software Design Documents for individual subsystems in LPC system.

## Definitions and Acronyms

*See Humming Vue Plus 02 Software Definitions and Acronyms* document.

## Applicable Documents

*BS EN 62304: 2006, Medical Device Software - Software life - cycle Processes*, section 5.3

# SOUP Items

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Title** | **Manufacturer** | **The unique SOUP designator** |
| Operating system | The FreeRTOS™ Kernel | Open source | X86 Release Ver. 9.0 |
| LPC Library | Lpc\_board\_ea\_oem\_4357  Lpc\_chip\_43xx | NXP | Nxp Ver 1.0 |

Table – Soup Items List

Notes: Update Library each 3 month.

# Architecture overview

LPC is a subsystems of of Humming Vue Plus\_02 Ventilator system. It has the main function below:

* Manage and control hardware(motor, sensor, switch, …)
* Handle button, encoder signal.
* Control LED, buzzer.
* Communication with Apalix IMX6

# Block diagram



Figure System block Diagram

## Apalis Imx 6 description

The Apalis IMX6 is a computer module based on the NXP®/Freescale i.MX 6 embedded System-onChip (SoC). The SoC features a scalable multicore ARM Cortex™ A9 processor with one to four cores, depending on the version.

Some information of Apalis IMX6 is described as follow:

* The CPU frequency peaks at 1.2GHz
* 1G memory.
* The module delivers high CPU and graphical performance with minimum power consumption.

The main function of this system refers to **3 Architecture overview**. The software is used on this component is embedded Linux apalis-imx6 version 2.7

## LPC description

The LPC435X\_3X\_2X\_1X are ARM Cortex-M4 based microcontrollers with Floating Point Unit (FPU) for embedded applications which include

* ARM Cortex-M0 coprocessor, up to 1 MB of flash and
* 136 kB of on-chip SRAM
* 16 kB of EEPROM memory
* Two high-speed USB controllers, Ethernet, LCD,
* An external memory controller, operate at CPU frequencies of up to 204 MHz.

LPC component is subsystem of Humming Vue Plus\_02 ventilator. Apalis receives display data, alarm data from this component. Apalis IMX6 also sends settings, mode information to LPC component.

## Buzzer description

Buzzer notifies the warning signal in case alarm occurs and speaker can’t play sound. Buzzer is controlled by signal PWM 2,4 KHz and duty cycle 50%.

## LED description

LED notifies the warning light in case alarm occurs. There are two colors: yellow and red led.

## Sensor description

Get temperature value, O2 concentration, flow value and pressure value from hardware, analyze, process and send this information to manage.

## Motor description

Base on each mode that user set, motor runs to respond with mode and system.

## Valve description

Base on requirement of system, each valve is on or off state to follow each mode.

## Power description

Manage power of system.

## USB description

This component to communication with Apalis IMX6.

# LPC Task Architecture

General system is shown in following diagram. This diagram shows the processor and device layout of the system.



Figure – LPC system diagram

LPC systems communicates with IMX6 system by IPC module using USB bus.

Another tasks want to communicate with IMX6 systems, it must communicate with IPC module through a Queue and vice versus.

In LPC system, if one task want to pop up a event, they use public function that providing by PD module. In PD module, when event occurs, it automatically catch and process follow event request.

Alarm module provides public function to set alarm, reset alarm when some problem happens with system. Alarm can directly access with some devices that it manages.

In BD module, it has a global data that another module can access through some public function. It is main responsibility to manage Device. If one task want to access to device, It must access through BD module.

CBIT module access BD module through a queue also.

Diagnostic module can access directly to Device. But DB and Diagnostic can not run at the same time.

## Update Task



Figure – LPC update progress Diagram

This section describes the LPC Architecture when systems updates.

This progress is starting when IMX6 request updating process. It sends signal update to LPC.

After receiving this signal, LPC begin to update and IMX6 waits until get successful signal from LPC.

The process is successful.

## IPC Task

### Priority: TBD

### Delay time: TBD

### Size of task: TBD

### Describe:

it is responsibility for communication between 2 subsystems (IMX6 – LPC)



Figure – IPC Task Diagram

### Initialization

Initialize USBVCom driver in hardware and Queue to communicate

### Interfaces

Provide Queue for another task can access.

## Alarm Task

### Priority: TBD

### Delay time: 100 ms

### Size of task: TBD

### Describe:

Annunciates alarm conditions based on alarm information provided by Breath Delivery, Patient Data, and CBIT. The Alarms task is also responsible for managing alarm priority elevation.

The Alarms task interfaces with the Device to activate the alarm LED indicator lights, sound the appropriate alarm tone, and react to alarm silence and alarm reset commands.

Figure – Alarm Task Diagram

### Initialization

MgrAlarmStat class initialize a list alarm follow request of system

### Interfaces

Alarm module provide public function for another module can access

## PD Task

### Priority: TBD

### Delay time: 0 ms

### Size of task: TBD

### Describe:

Calculates monitored data and sends it to the GUI Application task; checks patient data related alarms such as high respiratory rate and low exhaled minute volume; interfaces with the Alarms task to annunciate the alarms.



Figure – Patient data task diagram

### Initialization

At initialization, all patient data averages are reset and patient data displays are blanked. The BreathPhase data object must exist and contain valid data before the Patient Data task runs

### Interface

When the Breath Delivery Application Task detects a change in breath phase, it notifies the Patient Data via the Event Flag. The Breath Delivery Application Task samples pressure and flow data each time it runs and stores it in the BreathPhaseData object. Data in the BreathPhaseData object is double buffered so that the Breath Delivery Application Task can fill up data for the current breath while the Patient Data Task uses the previous breath’s data to compute monitored data. When the Breath Delivery Application Task starts a new breath, it switches the buffer pointers in the BreathPhaseData object.

If the Patient Data Task detects a change in a patient data alarm status, it notifies the Alarm Task by placing an event on the Alarm Status object.

The Patient Task also interfaces with the GUI Task to display updated patient data.

Additionally, the Patient Task uses an OS Timer to control the timed execution portion of its loop. The timer allows the Patient Data Task to calculate patient data such as the delivered %O2 which is time based instead of breath phase based.

## BD Task

### Priority: TBD

### Delay time: 5ms

### Size of task: TBD

### Describe:

Highest priority task during multi-tasking operation. Responsible for determining breath phase transitions, breath triggering, and detecting apnea and other breathe delivery alarms. This task interfaces with the GUI Applications task to provide real-time pressure readings for display. The Breath Delivery task also interfaces with the Alarms task to provide breath delivery alarm status and with the Patient Data task to provide raw patient data.

The Breath Delivery task interfaces with the Flow Controllers, the Inhalation Pressure Controller and Exhalation Pressure Controller to perform the physical closed loop control of the air and oxygen valves.

The Breath Delivery Application Task is a cyclic task based on a 2 ms Real Time Operating System (RTAI) timer. When the timer expires, the timer expiration routine signals the BD Timer Event Flag in the BD Event Flag group.

The Breath Delivery Application Task also has an associated event queue which allows other tasks to communicate with it. Whenever the Breath Delivery Application Task wakes up because of its timer, it pulls an entry off of its queue if one is available. Entries which can be placed on the queue include requests for settings changes from the GUI task and requests to perform operator requests (Manual Breath/HFO SI, 100% O2 and Exhalation Hold) from the Device. In order to keep the servicing of the controllers deterministic, the BD Application task does not pull entries off of its queue until the bottom of its loop, and it processes a maximum of one request each BD Task cycle. In other words, the check for alarms, triggering and the command of the controllers is performed at the top of the loop. Then, BD checks to see if any requests have been placed on its queue. These requests, therefore, are not serviced until the start of the next loop of the BD task.

The Breath Delivery Application Task has the following primary objectives

1. Collect current flow and pressure readings.
2. Determine which types of breaths are delivered based on operator settings for ventilation mode and other ventilator settings.
3. Detect patient triggering.
4. Interface with the HFO, Flow, Exhalation and Pressure Controllers to control breath delivery.
5. Handle operator requests such as Manual Breath/HFO SI and 100% O2.
6. Provide alarm detection and response to conditions
7. Interface with the Alarms Task to report these alarm conditions.
8. Provide flow, pressure and breath phase data to the Patient Data task.
9. Report pressure and breath phase data to the Graphical User Interface (GUI) task to be displayed on the real-time displays.

### Initialization:

Before the Breath Delivery Application Task begins, all of the mode, trigger and phase objects must be created and initialized. Additionally, the Breath Delivery Application Task cannot enter its main loop before it receives initial settings from the GUI task. It accomplishes the settings initialization by blocking on its queue until it receives the settings. Once the settings are received the BD cycle timer is started and the task enters its main loop.

The controllers must also be initialized and ready to receive commands from the Breath Delivery Application Task.

### Interfaces

The Breath Delivery Application Task calls Sensor public access methods to cause the flow and pressure sensors to be sampled. The Breath Delivery Application Task controls the execution of the controllers by interfacing directly with the Flow, Pressure and Exhalation Valve Controllers via public access methods.

The Flow Controllers receive desired flow commands from the BD Task. Using the desired flow and currently measured flow, step positions for the Air and Oxygen Valves are determined and transmitted to the corresponding stepper motor controller.

The Exhalation Pressure Controller is provided with the Positive End Expiratory Pressure (PEEP) value by the Breath Delivery task. The Exhalation Pressure Controller uses the measured circuit pressure and PEEP to compute an exhalation valve step position. This exhalation valve step position is then transmitted to the exhalation valve stepper motor controller.

The Breath Delivery Application Task interfaces with the Alarms Task via shared Alarm Status objects. The Breath Delivery Application Task also sets event flags in the Patient Data Event Flag Group that is used to control the Patient Data task, and it interfaces with the GUI task via a shared Real Time Data object to report real time pressure and breath phase data.

The BD Task is also responsible for performing some of the Valve Stuck tests based upon commanded position and measured flow. If any of the tests fail, the BD Task interfaces with the ErrTask, which handles the error. And, the BD Task is responsible for notifying the AzTask when a pressure transducer auto-zero is to be performed.

## CBIT Task

### Priority: TBD

### Delay time: TBD

### Size of task: TBD

### Describe:

Performs on-going background checks. This task also functions as the ‘slack’ task in the system and uses all remaining throughput to perform continuous testing



Figure – Cbit Task Diagram

The CBIT Task is a continuously run background task with the lowest priority in the system. The CBIT Task has the following primary objectives:

1. Perform Auto-Zero, Motor Stuck tests.
2. Perform validation CRC computations on critical ventilator data.
3. Perform Analog to Digital Converter (ADC)/Digital to Analog Converter (DAC) wraparound tests.

### Initialization

Before the CBIT Task begins, all of the device objects must be created and initialized. The CBIT task does not have any direct dependencies on the other tasks in the system. The CBIT task assumes that the system is up and operating.

### Interface

The CBIT task interfaces directly with the device objects that it checks. It also interfaces with the Error Handling objects to force a system reset or put the ventilator in an inoperative state in response to a failed test.

## Diagnostic Module

### Priority: TBD

### Delay time: TBD

### Size of task: TBD

### Describe:

Executes the diagnostics mode self-tests (i.e. SST and EST). The GUI task is utilized by Diagnostics to obtain operator input and display test results. The Device is utilized to command the various pneumatic systems during test.



Figure – Diagnostic task diagram

### Initialization

When the Diagnostic task is created, it is responsible for creating the other diagnostic subsystem classes.

### Interfaces

The Diagnostic subsystem interfaces with four other subsystems: Devices, RS-232, Diagnostic GUI, and Controllers. Devices provide the sensor control and query. RS-232 allows test results and sensor data to be sent to an external device. Diagnostic GUI provides the Intel® Core™ 2 duo CPU P8400@2.26gHz user interface routines required while in diagnostic mode. And lastly, the Controllers subsystem provides control of the flow valves and query of the pressure and flow sensors.

## Monitor Watchdog Module

### Priority: TBD

### Delay time: 10ms

### Size of task: TBD

### Describe:

it is responsibility for reset system when system crashes.

### Initialization

Initialize watchdog with 21ms

### Interface: TBD

# Implementation Constraints

## Programming Language

All software for the Humming Vue Plus 02 ventilator will be implemented in the C++ programming language.

All source code will be compiled/ assembled and linked using the LPCXpresso v8.2.2 [Build 650]

## Memory Utilization

The software is designed to reside in the 1M Byte Flash memory (eUSB). This memory space contains the operating system, NXP library, device drivers, and software application that performs the actual ventilator and diagnostic mode functions.

Subsequent changes and enhancements to the software can be made with an external USB containing new application. These changes and enhancements will affect the whole software.

## Task Starvation and Deadlock Protection

The LPC software design will require each task during normal operation to check in with a monitor routine at a preset time interval. If any task should fail to check in within the prescribed interval, the monitor routine will cease strobing of the watchdog timer. This will result in the ventilator being reset. The monitor routine will also be responsible for logging this type of failure including the name of the offending task, in the error log prior to the reset. This information will then be available for use in diagnosing software anomalies.

In Diagnostics Mode, the watchdog timer will be strobed automatically during each cycle of the BD Task.

## Handling of Unanticipated Conditions

The LPC software design will include specific checks for unanticipated conditions. These checks will include:

Pre-Conditions: Items assumed to contain specific data values or ranges of data values prior to an operation.

Assertions: Anomalous results detected during an operation.

Post-Conditions: Items assumed to contain specific data values or ranges of data values following an operation.

When the software detects that any unanticipated conditions have occurred, a error log entry will be made. This entry will contain the file name and line number where the anomaly occurred, and will then be available for diagnosing and correcting software anomalies. Once the information has been logged, the system will be reset by ceasing the strobing of the watchdog timer.