USB Support On Non-Linux STBs

Contents

Chapter 1. Preface

Purpose of this Document

Content

This document provides technical information about the USB feature integration for non-Linux STBs

Audience

This document is written for

- software engineers who are integrating the USB stack on new platforms
- the product owners of any HDI project that needs to support USB mass storage and return path using USB.

Assumptions, Dependencies and Constraints

It is assumed that the reader is familiar with Linux and uCos operating systems, the C programming language, and the architecture and operation of an STB.

Within this document, the following words and phrases have the meanings stated here:

- "**shall**" is used to indicate a mandatory requirement or behaviour, which must be met;
- "**should**" is used to indicate a recommendation, which is desirable, but not mandatory;
- "**may**" is used to indicate a behaviour which is acceptable, but not required;
- "**will**" does not indicate a requirement, but can be used in a statement of fact supporting a requirement, or to indicate a future intention.

References

Table 0.1 lists documents and other reference sources containing information that may be essential to understanding topics in this document

Table 1. References

No.	Designation	Title
1	USB20CV x64-bit	USB 2.0 Specification
2	Document 124945r00JB	Jungo USBware Documents
3	HDI-ICD-171	HDI4 specification

Terminology

Table 0.2 provides a short glossary of any terms crucial to the understanding of this document, and lists the acronyms and abbreviations used in the document

Table 2. Terminology

Term	Definition
USB	Universal Serial Bus
STB	Set-Top Box
HDI	Hardware Device Interface
CDI	Common Device Interface
PnP	Plug and Play
EHCI	Enhanced Host Controller Interface
OHCI	Open Host Controller Interface
UHCI	Universal Host Controller Interface
XTVFS	XTV File System

Chapter 2. The USB host stack

To connect and use a USB device on an STB (or any other device), a USB host stack is needed to interact with the device that is connected via a USB port. The USB host stack does the following:

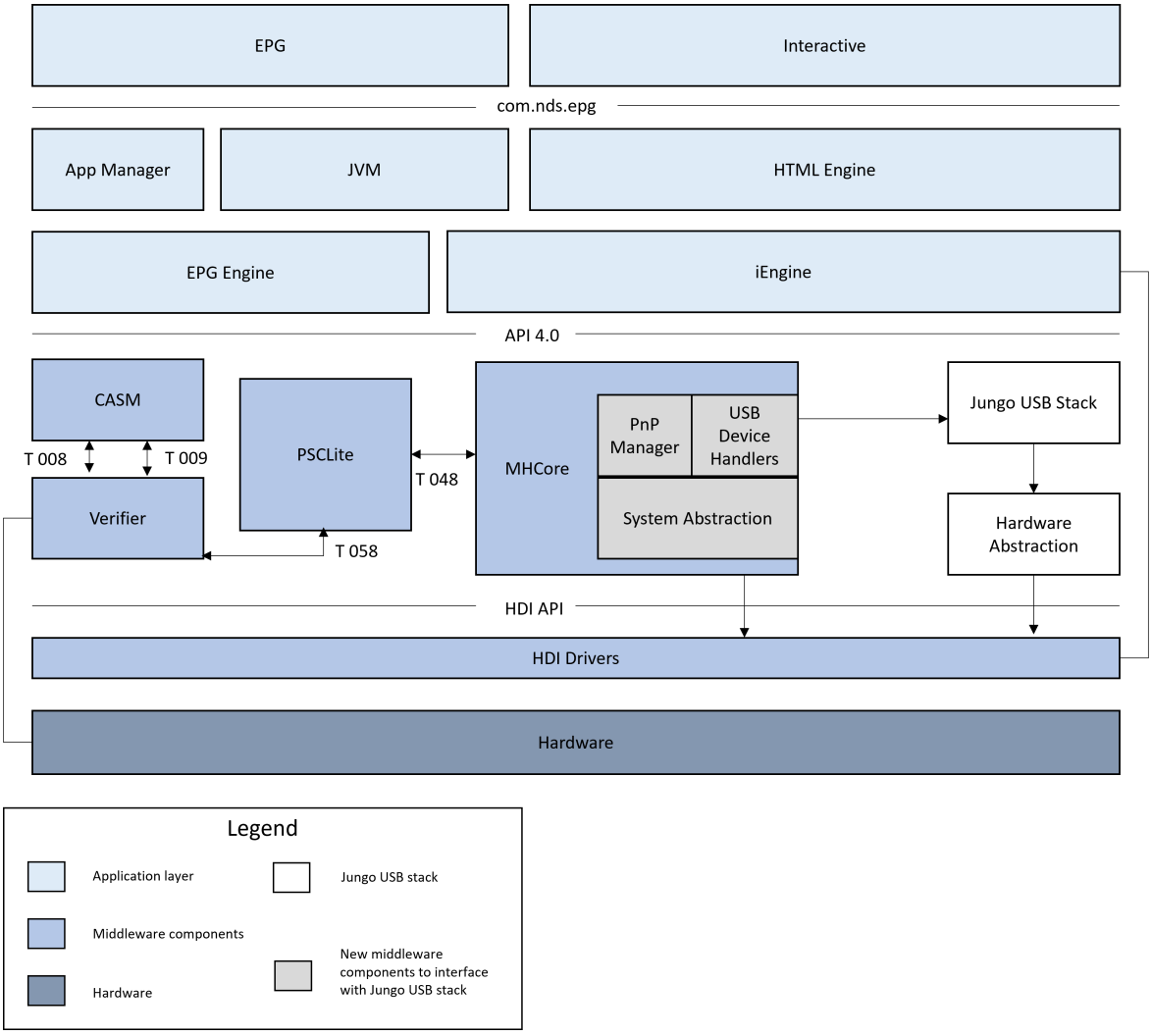
1. communicates with the hardware drivers of the box to interact with the USB device that is connected.
2. collects the information about the type of the device that is connected.
3. initializes respective callbacks for communicating with the USB device that is connected.

Chapter 3. Linux and non-Linux systems

The USB feature implementation depends on the architecture of the STB. This is because the operating systems (OS) used by STBs can be different. There are two categories:

- STBs based on the CDI (Common Device Interface) architecture use Linux OS. Linux already has an internal USB host stack. Therefore, additional support is not required.
- STBs based on the HDI (Hardware Device Interface) architecture use UCOS OS which does not support any USB stack. Therefore the Jungo USB stack is installed for such systems to interact with the USB devices

  The below diagram dipicts the architecture of Jungo USB stach on HDI based systems



The **Hardware Abstraction layer** is present to make the Jungo USB stack independent of the hardware being used. After the implementation of the HDI calls, this layer will not be required as the HDI calls will be directly made and hardware details will be handled by the HDI.

The **System Abstaction layer** is present in the Core component to achieve abstraction between the USB stack used, the file system used and the IP stack used

The **PnP Manager** has implements functionalities to communicate with the USB device that is connected. These functionalities will be called by the Application depending on the device and user preference. The module is discussed in detail here *(on page 11)*

Chapter 4. Jungo USB Stack

Overview

Jungo provides a complete, small footprint USB Host Stack. This includes Application Programming Interfaces (APIs) and services that enable software and device manufacturers to incorporate standard USB Host connectivity in their embedded device.

The stack supports the following USB versions:

- USB 1.1: Full Speed rate of 12 Mbit/s (1.5 MB/s)
- USB 1.0: Low Speed rate of 1.5 Mbit/s (187.5 kB/s)

USB 2.0: High-Speed rate of 480 Mbit/s (60 MB/s)

Porting Layer

Driver Level

The HDI driver has to provide the following functionalities to the Jungo USB stack

USB Controllers

The USB stack is generic to all platforms and the implementation is done in accordance with the USB specification (EHCI, OHCI and UHCI). The USB stack should know the controllers available in the platform and register an offset to initialize the stack.

The HDI driver provides the below interface to pass the controller and register information to USB stack. It is done during the HDIInit() call.

```
HDI_STATUS CORESetUSBHostController( DEVICE_NO device_no, HDI_USBHC_TYPE hc_type, ULONG *reg_address_p)
```

Note:

- If driver does not invoke this call, the USB stack will not be initialized.
- The HDI can invoke this function more than once to support more than one controller.

Memory

The USB stack requires both cache and Non-cache memory.

- **Non-cache memory:**

  The USB controller requires non-cache memory for USB read and write operation. The below interfaces must be implemented by the driver to bring up the USB stack. These calls must be mapped in the Jungo memory porting layer.

  ```
  HDI_STATUS HDIGetNonCachedRam(void **non_cached_virt_p, void **non_cached_phys_p,ULONG non_cached_size)
  ```

  ```
  HDI_STATUS HDIFreeNonCachedRam(void *non_cached_virt_p)
  ```

  Note:

  - The stack requires atleast 256K Non-cache memory.
  - The USB stack will not work correctly, if virtual address or physical address is wrong. The controller shall throw spurious interrupts when the physical address is not correct.

- **Cache Memory:**

  Cache memory is provided by the middleware. The USB stack internally maintains a pool and the USB allocation and de-allocation is done from this pool by the USB stack. USB stack shall not access any external memory.

  The middleware creates the separate partition for USB stack cache memory operation. It provides the partition allocation and de-allocation interfaces. These interfaces are then mapped to the Jungo memory porting layer.

  The stack requires atleast 256K cache memory. This memory shall increase when project wants to support more classes.

Interrupts

The stack handles interrupts for USB operations (attach, detach, I/O). To make the stack common across all platforms, HDI driver provides the below mentioned interface to associate the interrupt.

```
HDI_STATUS HDIInitUSBHostController( DEVICE_NO device_no, HDI_USB_FP function_p)
```

USB Stack Tasks

USB stack uses five tasks for USB operation. The task details are listed in the below table. Middleware has one interface task to init the USB stack and also for the future use.

Table 3. USB Task details

S.No	Task	UCOS Priority
1	Controller	67
2	DSR	68
3	Core	69
4	Driver	70
5	Other(debug thread)	71

Porting Interface

Threads and IPC interfaces

USB stack requires OS Interfaces for thread creation and IPC. To make the generic USB stack, Jungo JOS layer defines the interfaces to be implemented by porting layer and NDS defines another generic interface(X_NDS_OS_..) which should be used by porting layer

> **Note:**
> The NDS interfaces are implemented using UCOS calls, no further porting required when porting the USB stack to new platform which uses UCOS

Memory

The USB stack memory porting layer defines two set of interfaces for memory alloc/dealloc operation.

1. Alloc/Dealloc from cache memory
    a. os_malloc()
    b. os_free

> **Note:**
> A separate USB partition of size 256K is created to handle this

2. Alloc/Dealloc from non-cache memory
    a. os_dma_alloc()
    b. os_dma_free()

Note:

- HDI calls are remapped to get the alloc and dealloc non-cache memory
- No further porting required on this because HDI already abstracted this implementation.

Bus Access

Jungo USB stack porting layer provides the set of interfaces to be implemented by porting layer for USB bus operations but the bus implementation differs from one platform to other platform.

Platform specific HAL layer is added in porting layer for USB bus operation and Jungo BUS porting layer invokes the platform specific functions implemented in HAL layer. The following platforms are support by HAL layer

- ST-5202
- ST-5228
- BCM-7454

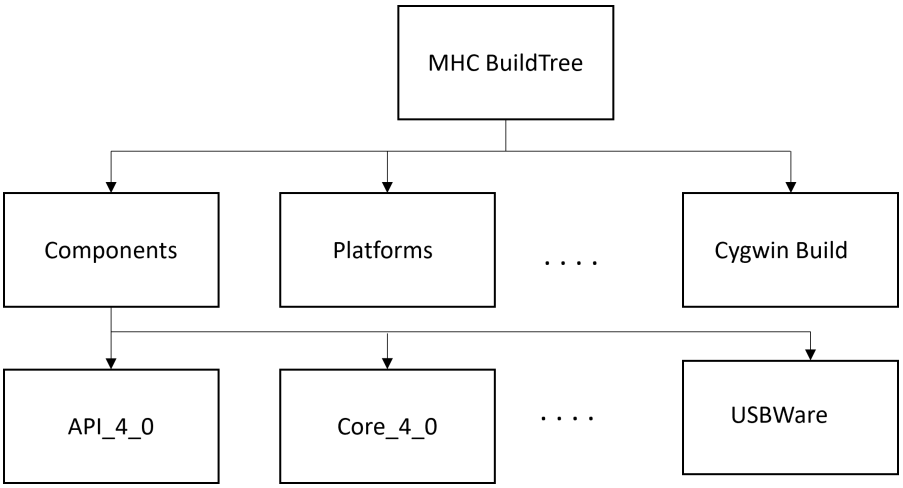
Chapter 5. Implementation details

The following components need to be implemented

Porting the USB Host Stack

The Jungo USB stack needs to be added on non-linux systems only. It will be added as a new separate component called UsbWare outside of the Core component folder.

The code tree structure is as follows:



The UsbWare component will be compiled externally using the Cygwin tool and the object file will linked thereafter.

These are the steps to build the UsbWare component:

1. Run the cygwinbuild batch file available in MHC_BuildTree folder
2. Change directory to root directory of USB ware
3. If you have previously build USB ware for different configuration, in order to clean the previous build run the following command

   ```
   Make distclean
   ```
4. Modify the relevant config/target_cfg.cfg file in order to define the modules you wish to build.
5. The following command creates the config.h file for the configurations done in the target configuration file

   ```
   Make dist=target_cfgs
   ```
6. Build USB ware by running the following command from root directory

   ```
   Make
   ```

7. Copy the created main USB ware library from USB ware root directory to the target specific driver's folder in platforms folder
8. Change directory to root directory of MHC_BuildTree
9. Run the following command to build the executable image using all the available libraries

```
./make_hdi release <TARGET_BOX>
```

Hardware abstraction layer for the USB host stack

Currently the JUNGO USB host stack needs to call the USB related hardware calls. To make the JUNGO independent of the platform the hardware abstraction layer is written, which takes care of the hardware details. Once the HDI specification is given the hardware related task is taken care by the HDI. Then this layer can be removed, as we can directly call the HDI calls

PnP Manager Component

PnPDevMan is the core component of the USB feature. This component takes care of attach detach notification to the application, maintains the device structures that tell about the device type, number of device connected and status of the devices

The source files for the PnPDevMan are:

1. Apipnp.h

   Contains the data structures that are used by USB component and function declaration for application interface
2. Pnpman.c

   Contains implementation for the PnPManTask and device status information. PnPManTask gets the events from the USB host stack about the attach/detach state of a USB device. If the device is a mouse or joystick events about the x, y location of the pointer is also received.
3. Gfs.c

   Contains the implementation of generic file system interface for XTVFS / Linux File system.
   It has functions like APIGFSMount to mount the USB mass storage device. APIGFSUnMount, APIGFSOpen, APIGFSRead etc are the other similar functions

The PnP Manager will be added in the Core component structure as a separate folder. It will compiled as a separate Core component. These are the changes to be made to the Core makefiles:

1. In Core_libraries.mk: include the following lines

   LIBRARIES_MAKEDIR_pnpmanager := $(LIBRARIES_MAKEDIR_core_4_0)/pnpmanager/code

LIBRARIES_OUTDIR_usbware := $(LIBRARIES_MAKEDIR_core_4_0)/pnpmanager/code

2. In the makefile: include the following line

   COMPONENT_SOURCE_LIBRARIES+=pnpmanager

System Abstraction Layer

The system abstraction layer is present in core to bring about the following abstraction:

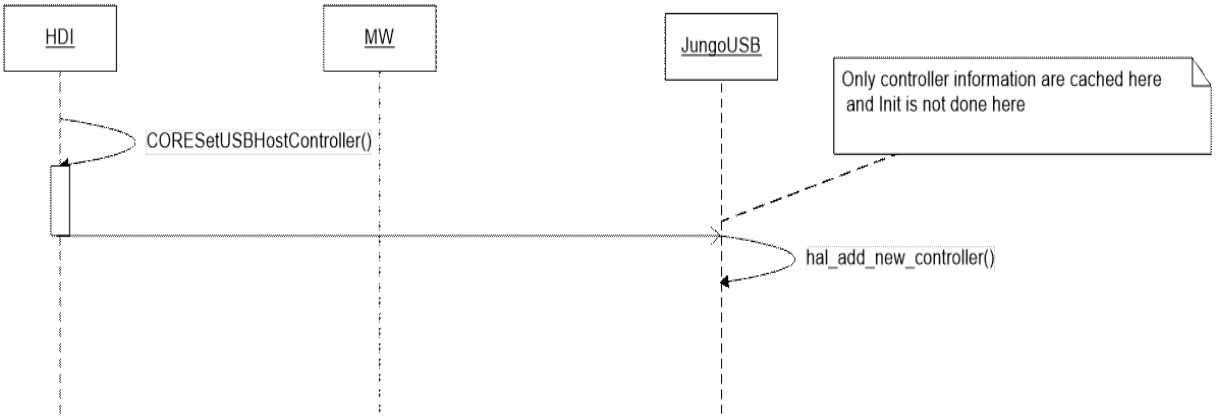
1. USB stack used – Linux/Jungo
2. File system to be used – Linux/XTVFS
3. IPStack to be used – Linux/Treck (IP Stack existing in CORE)

A compilation switch used for this purpose. For instance, LINUX_USED can be used for the purpose. For CDI architecture this flag is defined and Linux calls are compiled and used in this layer. For HDI architecture boxes this flag will not be defined and JUNGO USB host stack calls are made for host stack, for file system XTV File system (XTVFS) is used and for IP stack to process the IP packets in case when the USB device is a communication device, Treck is used for HDI boxes or else Linux IP stack is used. Treck is the IP stack existing in CORE already

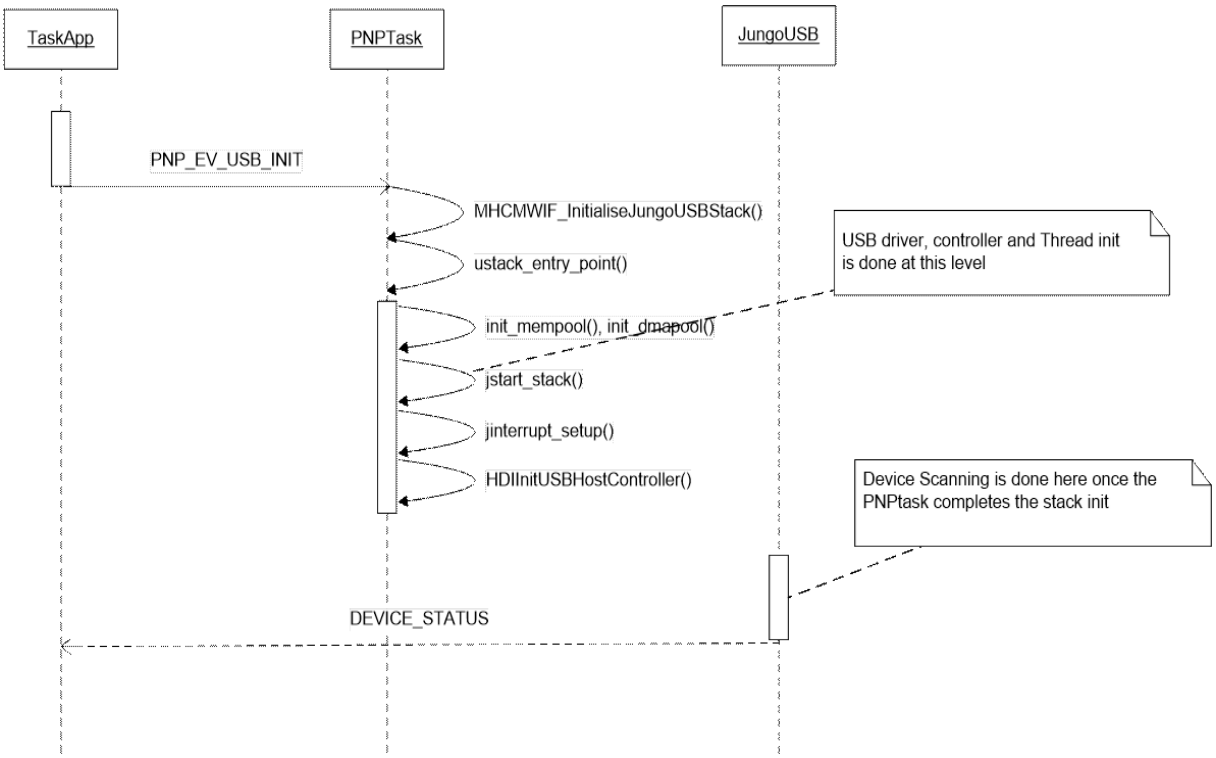
Chapter 6. Sequence Diagrams

USB Stack Initialization Sequence

The sequence diagram below shows the new controller notification sent by the driver as a trigger to initialize the Jungo USB stack



The sequence diagram below shows the complete high-level initalization sequence of Jungo USB stack



USB Interaction Sequences

The sequence diagram below shows the high-level communication between the Jungo USB and the middleware. It is not specific to any USB class.