Contents

1.0 Introduction........................................................................................................................................5

2.0 First-time User....................................................................................................................................8
  2.1 Prepare the Build Machine ..............................................................................................................8
  2.1.1 Check the Prerequisites .............................................................................................................8
  2.1.2 Configure the Build Machine ....................................................................................................9
  2.2 Build the Base BSP Image ..............................................................................................................12
  2.3 Boot the Target Machine ...............................................................................................................13
  2.4 Install the Intel® Media SDK on the Target Machine ......................................................................14

3.0 Advanced User..................................................................................................................................16
  3.1 Prerequisites ....................................................................................................................................16
  3.2 Generating Bootable Image with Pre-installed Media SDK ............................................................16

4.0 Verifying Correct Installation...........................................................................................................21

5.0 Compiling and Running Intel® Media SDK Samples on the Target Machine.................................23

6.0 Installing for Non-supported Configurations ..................................................................................25

7.0 Legal Information..............................................................................................................................26

Figures

Figure 1. Installation Process ..................................................................................................................7
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 2017</td>
<td>003</td>
<td>MR3.1 Release</td>
</tr>
<tr>
<td>June 2017</td>
<td>002</td>
<td>MR3 Release</td>
</tr>
<tr>
<td>March 2017</td>
<td>001</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

§
1.0 Introduction

Intel® Media Software Development Kit for Embedded Linux (Intel® Media SDK) is a software development library that exposes the media acceleration capabilities of Intel® platforms for decoding, encoding and video preprocessing.

Unlike software-only packages that can be expected to work across a wide variety of platforms and environments, Intel® Media SDK is a combination of driver, library, and graphics stack components requiring specific hardware, Linux distributions, kernel levels, etc. as described here.

This document covers the basics of installing, compiling and validating the correct operation of Intel® Media SDK, using Intel® Media SDK sample applications.

For more information on the sample source code provided, see “/opt/intel/mediasdk/doc/Media Samples Guide.pdf”.

Another set of simplified examples can be found under the “tutorials” tab at https://software.intel.com/en-us/intel-media-server-studio-support.


1. Manually install the IoT Media stack (recommended for first-time users): Build the Yocto-based Linux bootable image with the base Board Support Package (BSP) and manually install IoT Media stack at Run time described in Section 2.0.

2. Pre-Install the IoT Media stack (recommended for advanced users): Build the Yocto-based Linux bootable image comprising the base BSP and IoT Media stack at Compile time described in Section 3.0.

The Operating System (OS) image can be built on multiple Linux distributions. However, only Ubuntu* is currently validated.
### 1.1 Terminology

#### Table 1. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOS</td>
<td>Basic Input/Output System</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package</td>
</tr>
<tr>
<td>CRB</td>
<td>Customer Reference Board</td>
</tr>
<tr>
<td>EFI</td>
<td>Extensible Firmware Interface</td>
</tr>
<tr>
<td>GNU GRUB *</td>
<td>GRand Unified Bootloader*</td>
</tr>
<tr>
<td>GST</td>
<td>GStreamer</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>Intel® Media SDK</td>
<td>Intel® Media Software Development Kit</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>MSDK</td>
<td>Media Software Development Kit</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Drive</td>
</tr>
<tr>
<td>UFO</td>
<td>Unified 3D library</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>
The following flowchart illustrates installation process:

**Figure 1. Installation Process**

![Installation Process Flowchart]

**Note:**

Base BSP – the Yocto Project BSP Base meta layer tarball for Intel® Atom™ E3900 SoC Family.

Unified 3D library (UFO) Package— Intel® Unified 3D Library includes iHD VA driver and LibVA binaries.

Media Software Development Kit (MSDK) Package – Intel® Media SDK package containing libraries, plugins, samples, tools etc.
2.0  **First-time User**

This section is meant for *first time users* who are completely new to the Yocto Project and Media SDK. This section describes the step-by-step procedure to build a Yocto Project-based Linux machine and run Intel® Media SDK on the “Intel® Atom™ E3900 SoC Family.

For the *advanced users* who are already familiar with Yocto Project and Intel® Media SDK, go to Section 3.0.

**Note:** Instructions are limited to specific platforms and packages tested for reference, but are intended to be general.

2.1  **Prepare the Build Machine**

The generic procedure to prepare the build machine can be found at [https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master](https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master).

For easy reference we have provided the procedure used during internal testing in the following sections.

Customers are advised only to use the information in the following sections for reference because the configurations, settings, Package names, links, etc. may change in the future.

2.1.1  **Check the Prerequisites**

- **Host/Build Machine Hardware:**
  - Processor: Core i7-4770R @3.20GHz
  - Platform: Gigabyte M4HM87P-00
  - Memory: 16GB DDR3 @1600MHz
  - Disk: > 200GB (50GB if the INHERIT variable is used to remove the unused workspace)

- **Operating System(OS):** Ubuntu 14.04.5 was tested with this release

- **Download the following software packages:**
  - Media SDK binary tarball i.e [IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz](https://software.intel.com/en-us/media-sdk)
  - The LibVA driver (iHD driver binary) is packaged in Intel® Unified 3D Library tarball (referred to as UFO binary tarball). Refer to this link-[http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html](http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html) to obtain the latest UFO binary tarball i.e [intel-}
2.1.2 Configure the Build Machine

The build script assumes it will be run connected to a VPN. This means some proxy server information must be set. This adds additional steps to prepare the git environment since the proxy for the git client is different from the host environment.

**Note:** If the proxy settings are not setup correctly you could experience build issues. So to save time, it is recommended to avoid the proxy settings and directly use the public network.

Following are the steps to set up the proxy server and it assumes the user Id is "aplbuild"

1. Open “Network setting” and set the proxy server.
2. Click the “apply to system wise” button to apply the new settings to the whole system.
3. Set the environment variables.

```
$ sudo vi /etc/sudoers
add: aplbuild        ALL=(ALL:ALL) ALL

$ vi ~/.bashrc
add:
export SOCKS_SERVER=$socks_proxy
export HTTP_PROXY=$http_proxy
export HTTPS_PROXY=$https_proxy
export FTP_PROXY=$ftp_proxy
export SOCKS_DIRECT=$HTTP_DIRECT
export NO_PROXY=$HTTP_DIRECT
export ALL_PROXY=$HTTP_DIRECT
export http_direct=$HTTP_DIRECT
export socks_direct=$SOCKS_SERVER
export no_proxy=$NO_PROXY

$ vi /etc/apt/apt.conf
confirm:
Acquire::http::proxy "http://proxy.aplbuild.com:911/";
Acquire::https::proxy "https://proxy.aplbuild.com:911/";
Acquire::ftp::proxy "ftp://proxy.aplbuild.com:911/";
Acquire::socks::proxy "socks://proxy.aplbuild.com:911/";
```

4. Install the build tools.

```
$ sudo apt-get update
```
5. Customize the Git environment.

$ vi ~/.gitconfig

```bash
add:
   [user]
   mail = aplbuild.user@aplbuild.com
   name = Aplbuild User

   [sendemail]
   smtpserver = smtp.aplbuild.com
   signedoffcc = false
   suppresscc = all
   chainreplyto = false
   assume8bitEncoding = utf-8
   from = < Aplbuild User> <aplbuild.user@aplbuild.com>
   confirm = always

   [color "grep"]
   match = red

   [color]
   diff = auto
   ui = auto
   interactive = auto
   grep = always

   [alias]
   co = checkout
   br = branch
   ci = commit
   st = status
   ol = log -oneline

   [core]
   editor = gedit OR vi
   gitproxy = /home/aplbuild/bin/gitproxy
```

$ mkdir ~/.bin
$ vi ~/.bin/gitproxy

```bash
add:
```
#!/bin/bash
exec socat stdio SOCKS:proxy.aplbuild.com:$1:$2
$ chmod +x /home/aplbuild/bin/gitproxy

6. Register your SSH key at GitHub.
   a. Confirm a GitHub account exists.
      
      ```bash
      $ ssh-keygen -t rsa -b 4096 -C "aplbuild.user@xyz.com"
      $ eval "$(ssh-agent -s)"
      $ ssh-add ~/.ssh/id_rsa
      $ sudo apt-get install xclip
      $ xclip -sel clip < ~/.ssh/id_rsa.pub
      ```
   b. Generate a local SSH key.
   c. Open the browser to website [https://github.com](https://github.com).
   d. Login to github account, find the icon on the right upper corner.
   e. Click the icon for the pull down menu, then Settings>SSH and GPG keys>New SSH key.
   f. Input the name of the new key and paste the key string from xclip.
   g. Click "Add SSH Key" button.

7. Add following settings to the .ssh/config file:
   ```bash
   $ vi .ssh/config
   add:
      host github.com
      user git
      hostname ssh.github.com
      identityfile ~/.ssh/id_rsa
      port 443
      tcpkeepalive yes
      compression yes
      connectionattempts 3
   ```

8. Reboot the machine.

9. In a temporary folder, do the following to register your SSH key. Otherwise your install could be blocked by Yes/No questions during SSH key registration.
   ```bash
   $ cd ~/tmp
   $ git clone git@github.com:01org/iotg-yocto-bsp-public (this step makes sure the '[[ssh.github.com]:443' RSA key to the host)
2.2 Build the Base BSP Image

1. On the host machine, create a working directory. For example, let this directory be `/home/user/development`.

2. Download the BSP for Yocto Project* from GitHub to your host machine
   - HTTPS directly from [https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master](https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master) by selecting the appropriate tag version, for example, E3900-MR3.1, from the top left menu or
   - SSH using the following commands:

   ```bash
   $ cd /home/user/development
   $ git clone https://github.com/01org/iotg-yocto-bsp-public.git -b e3900/master
   ```

   This git tree is maintained as single product branch. To get code base for a specific release, check out to its specific tag. For eg:
   - For PV release: `$ git checkout E3900-PV`
   - For Maintenance Release Version 1: `$ git checkout E3900-MR1`
   - For Maintenance Release Version 2: `$ git checkout E3900-MR2`
   - For Maintenance Release Version 3: `$ git checkout E3900-MR3`
   - For Maintenance Release Version 3.1: `$ git checkout E3900-MR3.1`

3. Run the `setup.sh` script in `iotg-yocto-bsp-public` to prepare the combo-layer build environment.

   **Note:** Driver Option: Select 1. Build Kernel image with CAVS HD Audio driver (Default).
   ```bash
   $ cd /home/user/development/iotg-yocto-bsp-public/
   ```
$ ./setup.sh
Select an option:
1. Build kernel image with CAVS HD Audio driver (Default)
2. Build kernel image with CAVS SSP Audio driver
3. Build kernel image with legacy HD Audio driver
Default option is build kernel image with CAVS HD Audio driver. If no input is received within 20 secs, default will be used.

Select an option:
1. core-image-sato-sdk (Default)
2. core-image-sato
3. linux-kernel
4. custom
Default build target is core-image-sato-sdk. If no input is received within 20 secs, default target will be built.
......

Note: Image Option: Select 1. core-image-sato-sdk (Default).
4. The build is successful if the message "BSP Build: PASSED!!" appears, and a bootable image i.e. core-image-sato-sdk-intel-corei7-64-*.hddimg, is available under yocto_build/build/tmp/deploy/images/intel-corei7-64/

2.3 Boot the Target Machine
1. Insert the Universal Serial Bus (USB) thumb drive to the USB port of the host machine and burn the live image using the “dd” command.

$ sudo dd if=tmp/deploy/images/intel-corei7-64-cavs-hda/core-image-sato-sdk-intel-corei7-64-cavs-hda.hddimg of=/dev/sdc && sync

2. Remove the USB thumb drive from the host machine.
3. Connect the USB thumb drive to the target device and connect the keyboard, the DisplayPort* to the monitor, Hard disk Drive (HDD)/Solid State Drive (SSD) and power cable.
4. Click the power button to start the target device.

Note: Steps 5 to 8 are applicable only to Intel Customer Reference Board (CRB).
5. Watch the monitor, when the Intel logo shows on the screen, click F2 key to enter the BIOS menu.
6. Select the Boot Manager. There should be two Extensible Firmware Interface (EFI) Hard Drive entries. Select the entry labeled “EFI USB Device”.
7. When the GRand Unified Bootloader (GNU* GRUB) menu appears, select “install” to start the installation process.
8. Follow the on-screen instructions to install the image to the target correct storage device

2.4 Install the Intel® Media SDK on the Target Machine

Note: The following steps do not include the installation of GStreamer-MediaSDK plugins. For the installation of GStreamer-MediaSDK plugins, refer to https://github.com/01org/gstreamer-media-SDK.

1. On the host machine, extract the UFO tarball (intel-linux-ufo-yocto_bxt-<version>-64bit.tar) to obtain iHD_drv_video.so and copy it to a USB thumb drive. Also copy MSDK tar ball (IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz) into the USB thumb drive.

2. On the host machine, clone the libva-staging source files to a local folder and copy it to the USB thumb drive (e.g: sdb1).

```
$ git clone git://github.com/01org/iotg-lin-gfx-libva.git
$ cd iotg-lin-gfx-libva
$ git checkout 9430287e9e1563777e3d51ba730636e78ed10796 -b mr2_staging
$ cd ..
$ cp -R iotg-lin-gfx-libva /dev/sdb1/.
```

3. Insert the USB thumb drive into the target machine. Transfer the Media SDK tarball and libVA-staging source files from the USB thumb drive to a local folder (e.g: downloads) on the target machine. Also, transfer the iHD VA driver to /usr/lib/dri as shown below:

```
$ mkdir downloads
$ cd downloads
$ cp /run/media/sda1/IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz
$ cp -R /run/media/sda1/iotg-lin-gfx-libva/
$ cp /run/media/ufo/user/lib/iHD_drv_video.so /usr/lib/dri/
```

4. Add environment variables.

```
$ vi /etc/environment
Insert env variables below:

export LIBVA_DRIVER_NAME=iHD
export LIBVA_DRIVERS_PATH=/usr/lib/dri
export LD_LIBRARY_PATH=/usr/lib/media-libva:/opt/intel/mediasdk/samples/_bin/x64:/opt/intel/mediasdk/lib64/
export XDG_RUNTIME_DIR=/run/wayland
```
5. **On the target machine, build and install the libva library.**

   ```bash
   $ cd ~/downloads/iotg-lin gfx libva/
   $ chmod +x autogen.sh
   $ ./autogen.sh
     --prefix=/usr
     --libdir=/usr/lib
     --enable-wayland
     --enable-x11
   $ make -j 4
   $ make install
   ```

6. **Unzip the tar ball and install the MSDK package.**

   ```bash
   $ tar -xvzf IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz
   $ cd IntelMediaSDK2017forEmbeddedLinux-<version>
   $ rpm -ivh --nolinktos intel-linux-mediasdk-16.6-541.yocto.x86_64.rpm
   $ rpm -ivh --nolinktos intel-linux-mediasdk-devel-16.6-541.yocto.x86_64.rpm
   ```

7. **Ensure that the Intel® Media SDK library can be found.** By default, the dispatcher searches in `/opt/intel/mediasdk/lib64/`. The `libmfxhw64-p.so.<version>` and `libmfxhw64.so` files can be located in library `/opt/intel/mediasdk/lib64/`.

8. **Reboot and make sure $MFX_HOME is set.**

9. **Go to Chapter 4.0 to verify the correct installation.**
3.0 Advanced User

This section is for advanced users who are already familiar with setting up Yocto Project-based Linux build machines (detailed steps to setup a Yocto build machine are not covered here) and describes the procedure to add all the necessary recipes during the build process so that the generated BSP image will contain the pre-installed IoT Media stack. Therefore, there is no need to install any of the components of the SW stack manually at run time. Upon booting the target machine with this BSP image containing the pre-installed IoT Media stack, users can directly run Intel® Media SDK samples.

3.1 Prerequisites

- Setup a build machine in order to build the Yocto Project BSP image for “Intel® Atom™ E3900 SoC Family”. Refer to https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master for the instructions to setup the build machine.

  Note: The IoT Media stack comprises Media SDK binaries, User Mode VAAPI driver (referred to as iHD-va driver in this document), libva-staging and GStreamer (GST) Media SDK plugins.

- The Yocto Project BSP Base meta layer tarball i.e iotg-yocto-bsp-public is available at http://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master.


- The iHD driver binary is packaged in Intel® Unified 3D Library tarball (referred to as UFO binary tarball). Refer to http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html to obtain the UFO binary tarball i.e intel-linux-ufo-yocto_bxt-<version>-64bit.tar.gz. This tarball may be packaged in unified_3dlib_<version>.zip.

3.2 Generating Bootable Image with Pre-installed Media SDK

1. On the build machine create a working directory. For instance, let this directory be /home/user/development.

2. Download the BSP for Yocto Project* from GitHub to your host machine.
• HTTPS directly from https://github.com/01org/iotg-yocto-bsp-public/tree/e3900/master by selecting the appropriate tag version, for example, E3900-MR3.1, from the top left menu or

• SSH using the following commands:
  
  $ cd /home/user/development
  $ git clone https://github.com/01org/iotg-yocto-bsp-public.git -b e3900/master

This git tree is maintained as single product branch. To get code base for a specific release, check out to its specific tag. For eg:

• For PV release: $ git checkout E3900-PV
  
• For Maintenance Release Version 1: $ git checkout E3900-MR1
  
• For Maintenance Release Version 2: $ git checkout E3900-MR2
  
• For Maintenance Release Version 3: $ git checkout E3900-MR3
  
• For Maintenance Release Version 3.1: $ git checkout E3900-MR3.1

3. Run setup.sh script in iotg-yocto-bsp-public to prepare the combo-layer build environment.
  
  $ cd /home/user/development/iotg-yocto-bsp-public
  
  $ ./setup.sh

4. First, you will be prompted to select the Audio driver:

   Select an option:
   1. Build kernel image with CAVS HD Audio driver (Default)
   2. Build kernel image with CAVS SSP Audio driver
   3. Build kernel image with legacy HD Audio driver

   Default option is build kernel image with CAVS HD Audio driver. If no input is received within 20 secs, default will be used.

Select Option 1. Build Kernel image with CAVS HD Audio driver (Default)

5. Next you will be prompted to choose the image type. Select Option 4. custom.

   Select an option:
   1. core-image-sato-sdk (Default)
   2. core-image-sato
   3. linux-kernel
   4. custom

   Default build target is core-image-sato-sdk. If no input is received within 20 secs, default target will be built.

6. Upon selecting option 4, configure the appropriate meta layers needed to pre-install the IOTG media stack during bitbake.
7. Once `setup.sh` script has run, there will be a new folder named `yocto_build` created under `/home/user/development/`.

8. Create another directory in the build machine to place all the binary tarballs. For instance, let this directory be `/home/user/rpm-binary`.

9. Place Media SDK binary tarball i.e `IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz` & UFO binary tarball i.e `intel-linux-ufo-yocto_bxt-<version>-64bit.tar.gz` in this directory.

10. Extract the packages.
    
    ```bash
    $ cd /home/user/rpm-binary/
    $ tar xf IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz --strip 1
    $ tar xf intel-linux-ufo-yocto_bxt-<version>-64bit.tar.gz
    ```

11. Place the Media SDK meta layer tarball i.e `meta-intel-msdk.tar.bz2` in `/home/user/development`.

12. Extract the package.
    
    ```bash
    $ cd /home/user/development
    $ tar xvjf meta-intel-msdk.tar.bz2
    ```

13. Edit `bblayer.conf` under `/home/user/development/yocto_build/build/conf` to include `meta-intel-msdk` layer
    
    ```bash
    $ vim bblayers.conf
    ```

    Add the paths of `meta-intel-msdk` as shown in the screen capture below.
14. **Edit the local.conf**

    $ vim local.conf

Export the path to all the binary tarballs i.e /home/user/rpm-binary, and also include recipes for Media SDK components (binaries, samples, documentation etc), iHD va-driver, and GStreamer-MediaSDK plugins as shown below.

```
    #export RPM_PATH="/home/user/rpm-binary"
    IMAGE_INSTALL_append = "msdk msdk-media gstreamer-msdk"
    IMAGE_INSTALL_append = "msdk-samples msdk-doc msdk-plugins"
```

**Note:** There is a <space> before msdk as shown in screen capture above.

15. **Edit the local.conf to exclude the default open source libva and i965 va-driver from installation. Instead, include libva-staging.**

    $ vim local.conf
16. Edit `local.conf` to uncomment the line as shown in the picture below.

```bash
$ vim local.conf
```

17. You are now ready to build the image containing the pre-installed IoT media stack.

18. Go up one level above to build/.

```bash
$ cd ..
```

Prepare the environment to run the bitbake command.

```bash
$ source ../oe-init-build-env
```

19. Start the image compilation.

```bash
$ bitbake core-image-sato-sdk
```

**Note:** It is recommended to start with a brand new image build. However, if you are reusing the previous build, execute the following steps before image compilation.

```bash
$ bitbake libva -c cleanall && bb virtual/libva -c cleanall
$ bitbake libva -c cleansstate && bb virtual/libva -c cleansstate
$ bitbake linux-firmware -c cleanall
```

20. Once the build is successfully complete, the bootable image will be available at tmp/deploy/images/intel-corei7-64-cavs-hda under the build directory.

21. Boot the Target Machine. Refer to Section 2.3 for booting up the image.
4.0 Verifying Correct Installation

1. Once the system has booted, confirm that an Intel VGA adapter can be found.

   ```
   $ lspci -nn
   ...
   00:02.0 VGA compatible controller [0300]: Intel Corporation Device [8086:5a84] (rev 0a)
   ...
   
   $ lspci -nn
   ...
   00:02.0 VGA compatible controller [0300]: Intel Corporation Device [8086:5a84] (rev 0a)
   ...
   ```

2. Add the user(s) who will run Media SDK applications to the video group.

   ```
   $ sudo usermod --a --G video $USER
   ```

3. Confirm that the i915 module is loaded correctly.

   ```
   $ lsmod | grep 'i915'
   i915        1138688 2
   drm_kms_helper 126976 1  i915
   drm         352256 2  i915,drm_kms_helper
   i2c_algo_bit 13564  1  i915
   video        24576 1  i915
   ```

4. Communication with the DRM library occurs via /dev/dri/cardx handler (usually /dev/dri/card0, though there can be more entries if there are more graphics adapters). Quite often permissions are set so that root access is required for users not in the video group. Add Intel® Media SDK application users to the video group and/or ensure users have permissions to work as a regular users.

   ```
   $ sudo chmod 666 /dev/dri/card0
   ```

5. Ensure that the Intel® Media SDK library can be found. By default, the dispatcher searches in `<sdk-install-dir>/lib64/`. The `libmfxhw64-p.so.<version>` and `libmfxhw64.so` files can be located in library `<sdk-install-dir>/lib64/`. For example, the library search path can be adjusted by the `LD_LIBRARY_PATH` variable.

   ```
   $ export LD_LIBRARY_PATH=$MEDIASDK_INSTALL_FOLDER/lib64
   ```

6. Make sure the following environment variables are set.

   ```
   $ export LIBVA_DRIVER_NAME=iHD
   $ export LIBVA_DRIVERS_PATH=/usr/lib
   ```

   Or

   ```
   $ export LIBVA_DRIVERS_PATH=/usr/lib64 (if multilib is turned on in the user environment)
   ```

7. Run “vainfo” to verify the correct installation of the iHD-va-driver.
$ vainfo
[sudo] password for mediasdk:
error: can't connect to X server!
libva info: VA-API version 0.35.0
libva info: va_getDriverName() returns 0
libva info: Trying to open
/opt/intel/mediasdk/lib64/iHD_drv_video.so
libva info: Found init function __vaDriverInit_0_32
libva info: va_openDriver() returns 0
vainfo: VA-API version: 0.99 (libva 1.67.0.pre1)
vainfo: Driver version: 16.y-xxxx
vainfo: Supported profile and entrypoints
VAProfileNone : VAEntrypointVideoProc
VAProfileNone : <unknown entrypoint>
VAProfileMPEG2Simple : VAEntrypointEncSlice
VAProfileMPEG2Simple : VAEntrypointVLD
VAProfileMPEG2Main : VAEntrypointEncSlice
VAProfileMPEG2Main : VAEntrypointVLD
VAProfileH264Baseline : VAEntrypointEncSlice
VAProfileH264Baseline : <unknown entrypoint>
VAProfileH264Baseline : <unknown entrypoint>
VAProfileH264Main : VAEntrypointVLD
VAProfileH264Main : VAEntrypointEncSlice
VAProfileH264Main : <unknown entrypoint>
VAProfileH264Main : <unknown entrypoint>
VAProfileH264High : VAEntrypointVLD
VAProfileH264High : VAEntrypointEncSlice
VAProfileH264High : <unknown entrypoint>
VAProfileH264High : <unknown entrypoint>
VAProfileVC1Simple : VAEntrypointVLD
VAProfileVC1Main : VAEntrypointVLD
VAProfileVC1Advanced : VAEntrypointVLD
VAProfileJPEGBaseline : VAEntrypointVLD
VAProfileJPEGBaseline : VAEntrypointEncPicture

Pre-built samples are included in /opt/intel/mediasdk/samples/_bin/x64.

Note: For Intel® Media SDK 2017 for Embedded Linux*, there is no software implementation. For these samples to work “-hw” must be specified on the command line.

§
5.0 Compiling and Running Intel® Media SDK Samples on the Target Machine

1. Build the MSDK samples for testing; ensure $MFX_HOME is set to the directory corresponding to your build. The build.pl script will only build samples if the prerequisites can be found. It will include rendering to X11 compositor by default.

   $ cd /opt/intel/mediasdk/samples/
   $ export |grep MFX_HOME
   $ perl build.pl --cmake=intel64.make.release -build

2. DRM frame buffer rendering is supported by default.

   $ cd /opt/intel/mediasdk/samples/__bin/x64/sample_decode h264 -hw -vaapi -i /path/to/h.264 -rgb4 -rdrm

3. (Optional) Build by enabling X11 DRI3/present support. For more details, refer to the whitepaper on DRI3 implementation at this location: http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html

   Note: MSDK sample binaries included into the package are built with Wayland* support.

   If graphic interface support is required, the following steps can be used:

   a. Use "-enable-x11-dri3=yes" for build.pl during compilation:

      $ perl build.pl --cmake=intel64.make.release -enable-x11-dri3=yes -build

   b. Modify xorg.conf as below:

      $ vi /etc/X11/xorg.conf
      Section "Device"
      Identifier   "Card0"
      Driver       "intel"
      BusID        "0:2:0"
      Screen 0
      +   Option   "AccelMethod" "sna"
      +   Option   "DRI" "3"

   c. Run the decode sample by specifying conversion to RGB4 ("-rgb4" since Present Pixmap could only take RGB) and "-r".

      $ /opt/intel/mediasdk/samples/__cmake/intel64.make.release/__bin/release
      $ ./sample_decode h264 -vaapi -hw -i
4. To enable Wayland* support:
   a. Use 

   ```bash
   $ perl build.pl --cmake=intel64.make.release -enable-wayland=yes -build
   ```

   b. Run the following command to load Weston.
   ```bash
   $ export XDG_RUNTIME_DIR=/tmp
   $ weston --tty=1 --idle-time=0 &
   ```

   c. Run the decode sample (execute commands below from Weston terminal).
   ```bash
   $ cd
   /opt/intel/mediasdk/samples/__cmake/intel64.make.release/__bin/release
   $ ./sample_decode h264 -vaapi -hw -i
   /path/to/input/elementary/AVC/stream -rwld
   ```
6.0 Installing for Non-supported Configurations

There is a good chance Intel® Media SDK works in other configurations. For full support, using one of the supported configurations is required. However, you are free to use the Intel Media SDK in other settings if you are willing to go through an extra step in reporting issues. If an issue can be reproduced in one of the supported configurations, it can be addressed, otherwise, you are on your own.

Since Intel® Media SDK is based on hardware access via the video driver, the main concern with alternative installations is making sure that all device IDs and other changes to the kernel are available. Usually patches required to enable working with the hardware are submitted to the kernel repository tip relatively quickly. If you use an advanced kernel or compile from close to the tip you are likely to get most, if not all, changes required for Intel® Media SDK to work.

Unfortunately there are no guarantees at this point with this approach. Enabling Intel® Media SDK to work on a wider set of configurations is a work in progress.

Important Note: When using your own kernel configuration, make sure CONFIG_MMU_NOTIFIERS=y is enabled. It can be disabled implicitly by disabling virtualization support.

§
THIS DOCUMENT CONTAINS INFORMATION ON PRODUCTS IN THE DESIGN PHASE OF DEVELOPMENT.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

VP8 video codec is a high-quality royalty free, open source codec deployed on millions of computers and devices worldwide. Implementations of VP8 CODECs, or VP8 enabled platforms may require licenses from various entities, including Intel Corporation.
Intel, the Intel logo, Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

§