See all versions
of this document

Porting Guide for embeddedsw Components

System Device Tree Based Build Flow

UG1647 (v2025.2) December 3, 2025

Table of Contents

Introduction

The legacy methodology of building AMD embeddedsw components takes .xsa as a handoff file from the hardware persona and uses mdd, mld, and mss files for different software configurations. This makes the legacy approach dependent on AMD proprietary tools such as the software command-line tool (XSCT) and hardware software interface (HSI). Without these proprietary tools, the embeddedsw components cannot be compiled using the legacy approach.

The new system device tree based flow aims to avoid such dependencies by embracing open-source industry standards. It uses the system device tree, CMAKE, and a Python-based open source tool named Lopper to organize the required build system.

The system device tree is a superset of a regular Linux device tree. Unlike a regular Linux device tree which represents hardware information required for Linux/APU only, a system device tree represents the complete hardware information in device tree format, thus creating opportunities to work with more use cases. It contains information about all the processors such as the platform management controller (PMC), PS management controller (PSM), real-time processing unit (RPU), application processing unit (APU), and all peripherals in the system. To generate a system device tree, see Appendix A: Generating a System Device Tree Using SDTGen.

The system device tree based flow uses Lopper to extract the required hardware metadata from the system device tree. Software configurations that were earlier done using mld/mss files in the legacy approach are now replaced with the CMAKE-based infrastructure where the CMAKE variables are set to generate the required headers. In the legacy flow, component specific information were stored in mdd/mld/mss files but in the new flow, they are stored in a YAML file. Python scripts read these YAML files, configure the component's sources using Lopper and CMAKE commands, and populate the standalone BSP specific data, accordingly. Thus, the whole build system replaces the usage of proprietary AMD files and tools with the available open-source infrastructure.

Porting existing embeddedsw standalone and RTOS components is necessary to ensure legacy software components work seamlessly in this system device tree based flow. The following sections explain the steps and changes required to port a legacy embeddedsw driver/library/application to the new system device tree based flow.

Assumptions and Prerequisites

This document makes the following assumptions:

- You are familiar with AMD embeddedsw drivers and libraries.
- You are familiar with CMAKE and yaml.
- You have an introductory knowledge of Python.

What's New

The following changes are new to this flow:

- The open source system device tree specifications borrows heavily from Linux device tree specifications. These specifications do not have the concept of Device IDs to identify between similar peripherals. As an example, in the legacy flow, if there are two Ethernet MACs in a system from the same vendor, each MAC would be assigned a separate Device ID. The Device IDs are positive integers starting from 0. The driver config structure always has a field `u16 DeviceId` that represents the assigned Device ID.

  IMPORTANT! *The system device tree based flow does not use Device ID and hence driver Config structures do not have an entry for Device ID.*

  In the new flow, driver Config structures have an entry for `char *Name` that stores the Compatible String(s) for the corresponding device. This compatible property along with the base address of the device are used the uniquely identify a device in a system. For more details refer to Porting a Driver.

- The system device tree based flow abstracts interrupt handling through a wrapper. In the legacy flow, driver users had the responsibility to figure out the interrupt controller to be initialized (GIC or Soft AXI INTC), which complicated the interrupt controller initialization and handling processes.

  The new flow provides generic APIs that are to be called with the generated interrupt ID available in the driver config structures. The new flow introduces a new entry in the driver Config structure with the name `IntrParent` that identifies whether the peripheral device is connected to GIC or AXI INTC for interrupt management.

- The system device tree based flow makes the XilTimer library inclusion mandatory. The XilTimer library abstracts out the timer and sleep handling in bare-metal environments. A typical system can have multiple timer devices which can be used as timers or to implement sleep functionality. The XilTimer provides uniform implementation, therefore you do not have to worry about the underlying hardware timer. For more details, refer to the XilTimer section in *BSP and Libraries Document Collection* (UG643).

Chapter 2

Porting to the System Device Tree Based Flow

The following sections provide instructions to port the existing embeddedsw drivers, libraries, and applications to work with the new flow.

Porting a Driver

This section describes the instructions to port an existing driver to the new flow. Each driver contains the following folders:

- `data`
- `src`
- `examples`
- `doc`

The new flow requires some changes to be made to these folders. These changes are described in the following sections.

Data Folder in Drivers

A YAML file named `<driver_name>.yaml` must be added inside the `data` folder. This YAML file is a replacement of the older AMD proprietary metadata files (*.mdd, *.tcl). For example, the `data` folder in the CSUDMA driver must contain a file named `csudma.yaml`.

The adopted YAML specification borrows heavily from open source Linux YAML device tree bindings with some customizations to work with AMD embeddedsw stack. The YAML file contains the following sections:

- Headers, copyrights, and generic information such as maintainer and type.

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: <Description of the yaml file>
maintainers:
    - Name(s) of the maintainer(s)
type: <Describes the type of baremetal module the yaml file describes,
e.g, driver>

device_type: <Describes the type of devices that the driver supports>
# Helps in giving meaningful error when there is a dependency of one kind
of IP over a template application, a library or a driver.
# Available values are ethernet, serial, interrupt-controller, ipi and
timer.
```

- Specification(s) that help(s) in selecting the corresponding driver for an IP. The corresponding driver would get picked up into BSP if the IP is present in the system device tree and its compatible property in the device tree node matches with this "compatible" string specified in the driver YAML. It replaces the .mdd file-based approach and instead relies on the Linux-like compatible property. Detailed usage of compatible property can be found in Appendix B: Usage Information for "compatible". Refer to the sample `csudma.yaml` at the end of this section.

```
properties:
  compatible:
     items:
       - const: <Typically the name of the HW IP as seen in xsa file,
e.g. xlnx,zynqmp-csudma-1.0>
  reg:
     description: <Physical base address and size of the controller
register map>
  other_properties_as_needed:
    description: <Description of the property>
```

- Specifications that help in generating the driver configuration file (*_g.c file) and driver config structure alongside the required `xparameters.h`. More data on the usage of below section can be found in below snippet and in Appendix C: Options Available in the "required" Section and Appendix D: Sample YAML for a Driver.

```
config:
    - <The name of the driver config structure, e.g. XCsuDma_Config>

xparam_prefix: xcsu
# Specifies a custom name different than that of ip-name in canonical
macros
# General canonical macros follow the
XPAR_X<driver_name>_<instance_number>_<PROPERTY> syntax
# X<driver_name> will be replaced with the value of xparam_prefix in the
above defined syntax

required:
# Describes the elements that will show up in the driver config structure
# As an example, csudma driver *_g.c file will contain the XCsuDma_Config
structure with following elements.
    #- compatible
    #- reg
    #- xlnx,dma-type
    #- interrupts
    #- interrupt-parent

# The order mentioned under the "required" section must be same as that
the driver config structure.

optional:
# Any information provided in the optional field for these properties
```

```
meta-data/values will be generated in the <driver name>Config.cmake file,
which will be then used by the drivers (e.g., for generating headers, and
so on).
# Same keys and rules as that under the required section.

additionalProperties:
# Entries that are needed only in xparameters.h and not in the config
structure will be generated using this option.
# Same keys and rules as that under required section.
```

- Specifications that describe driver example dependencies with hardware IP properties. Examples can depend on other supporting .h or .c files. Examples can have dependency on hardware IP properties (like interrupts). Few examples can be valid only for few platforms. Refer to the sample `csudma.yaml` at the end of this section and to the Appendix D: Sample YAML for a Driver section for more information. Examples listed here are also listed in the import example section in the GUI of the AMD Vitis™ Unified Software Platform 2023.2.

```
examples:
    <Example name>:
        - <supported_platforms>: <has to be mentioned only when example
is applicable to specific platform>
        - <dependency_files>: <List of supported files that need to be
pulled while creating the example>
        - <Which HW parameters, it is dependent on. As an example
xcsudma_intr_example.c needs interrupts to be enabled through HW.>
```

- Specifications that help in defining driver dependencies. A driver can be dependent on other drivers or a library, for example, libmetal.

```
depends:
# Create dict keys with the driver name on which the driver depends
# Source files of the driver mentioned here will be pulled in libsrc
    <drv_name>: []

depends_libs:
# Create dict keys with the library name on which the driver depends
    libmetal: {}
```

- Specifications that help in generating peripheral tests. The following examples are selected for generating peripheral tests. For more information, see Appendix E: Including a Driver Example in the Peripheral Test Application.

```
tapp:
    <Example name>:
        - declaration: <Declaration to be used for the example>
        - hwproperties:
            - <HW property that should be available for the example to be
picked as peripheral
test>
# For csudma driver example, it would look like:
tapp:
    xcsudma_selftest_example.c:
        declaration: XCsuDma_SelfTestExample
    xcsudma_intr_example.c:
        declaration: XCsuDma_IntrExample
        hwproperties:
            - interrupts
```

- A sample YAML file for the existing CSUDMA driver looks as shown in the following example. The YAML name is `csudma.yaml`. More information on how the YAML takes the device tree data and convert them into xparameters and Config structure can be found in Appendix D: Sample YAML for a Driver.

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: Bindings for CSUDMA controller
maintainers:
    - Maintainer Name <abc.xyz@domain.com>
type: driver
properties:
    compatible:
        items:
            - const: xlnx,zynqmp-csudma-1.0
    reg:
        description: Physical base address and size of the controller
register map
    xlnx,dma-type:
        description: |
            Differentiates the dma controller that driver supports
            0 - CSUDMA controller
            1 - PMCDMA-0 controller
            2 - PMCDMA-1 controller
    config:
        - XCsuDma_Config
    required:
        - compatible
        - reg
        - xlnx,dma-type
        - interrupts
        - interrupt-parent
examples:
    xcsudma_intr_example.c:
        - interrupts
tapp:
    xcsudma_selftest_example.c:
        declaration: XCsuDma_SelfTestExample
    xcsudma_intr_example.c:
        declaration: XCsuDma_IntrExample
        hwproperties:
            - interrupts
...
```

Src Folder in Drivers

A new file called `CMakeLists.txt` is introduced in the new flow. Instead of Makefile, every driver should have a `CMakeLists.txt` file. Almost all driver `CMakeLists.txt` files look similar. A typical driver `CMakeLists.txt` file is shown below:

```
cmake_minimum_required(VERSION 3.15)
project(<driver name>)
find_package(common)
collector_create (PROJECT_LIB_HEADERS "${CMAKE_CURRENT_SOURCE_DIR}")
collector_create (PROJECT_LIB_SOURCES "${CMAKE_CURRENT_SOURCE_DIR}")
include_directories(${CMAKE_BINARY_DIR}/include)
collect (PROJECT_LIB_SOURCES <source file>)
collect (PROJECT_LIB_HEADERS <header file>)
collector_list (_sources PROJECT_LIB_SOURCES)
collector_list (_headers PROJECT_LIB_HEADERS)
file(COPY ${_headers} DESTINATION ${CMAKE_BINARY_DIR}/include)
add_library(<driver name> STATIC ${_sources})
set_target_properties(<driver name> PROPERTIES LINKER_LANGUAGE C)
install(TARGETS <driver name> LIBRARY DESTINATION ${CMAKE_SOURCE_DIR}/build
ARCHIVE DESTINATION
${CMAKE_INSTALL_LIBDIR})
```

A typical `CMakeLists.txt` file for the CSUDMA driver looks as shown in the following snippet:

```

# SPDX-License-Identifier: MIT
cmake_minimum_required(VERSION 3.15)
project(csudma)
find_package(common)
collector_create (PROJECT_LIB_HEADERS "${CMAKE_CURRENT_SOURCE_DIR}")
collector_create (PROJECT_LIB_SOURCES "${CMAKE_CURRENT_SOURCE_DIR}")
include_directories(${CMAKE_BINARY_DIR}/include)
collect (PROJECT_LIB_SOURCES xcsudma.c)
collect (PROJECT_LIB_HEADERS xcsudma_hw.h)
collect (PROJECT_LIB_HEADERS xpmcdma.h)
collect (PROJECT_LIB_SOURCES xcsudma_sinit.c)
collect (PROJECT_LIB_SOURCES xcsudma_selftest.c)
collect (PROJECT_LIB_HEADERS xcsudma.h)
collect (PROJECT_LIB_SOURCES xcsudma_g.c)
collect (PROJECT_LIB_SOURCES xcsudma_intr.c)
collector_list (_sources PROJECT_LIB_SOURCES)
collector_list (_headers PROJECT_LIB_HEADERS)
file(COPY ${_headers} DESTINATION ${CMAKE_BINARY_DIR}/include)
add_library(csudma STATIC ${_sources})
set_target_properties(csudma PROPERTIES LINKER_LANGUAGE C)
```

Changes in Existing Source Files (.c and .h)

AMD drivers have to be generic and the driver sources should not include platform specific configuration details through `xparameters.h`. The existing embeddedsw driver uses an AMD proprietary parameter called `DeviceId` in the driver config structure. Because the system device is designed to be fully open source, it no longer uses `DeviceId` and instead uses `BaseAddress`.

Note: If you are creating a new driver, use `BaseAddress`. If you are porting an old driver to the new system device tree based flow, changes are needed to support both the legacy (`DeviceId`) and new (`BaseAddress`) flows.

A char * (for name) in the driver `*_Config` structure has to be used to support the system device tree based flow. This can also be used later for differentiating between different versions of IP in the driver.

This leads to the following changes in source files:

```
typedef struct {
+#ifndef SDT
        u16 DeviceId;           /**< DeviceId is the unique ID of the
                                  *  device */
+#else
+       char *Name;             /**< Unique name of the device */
+#endif
        UINTPTR BaseAddress;    /**< BaseAddress is the physical base address
                                  *  of the device's registers */
```

```
################################################################################
# Changes in *_sinit.c file
################################################################################

+#ifndef SDT
 X<Driver>_Config *X<Driver>_LookupConfig(u16 DeviceId);
+#else
+X<Driver>_Config *X<Driver>_LookupConfig(UINTPTR BaseAddress);
+#endif

+#ifndef SDT
 #include "xparameters.h"
+#endif

 /************************** Constant Definitions *****************************/

@@ -65,6 +67,7 @@
 *
 * @note                None.
 ******************************************************************************/
+#ifndef SDT
X<Driver>_Config *X<Driver>_LookupConfig(u16 DeviceId)
 {
      /* Legacy code */
 }
+#else
+X<Driver>_Config *X<Driver>_LookupConfig(UINTPTR BaseAddress)
+{
+       X<Driver>_Config *CfgPtr = NULL;
+       u32 Index;
+
+       /* Checks all the instances */
+       for (Index = (u32)0x0; X<Driver>_ConfigTable[Index].Name != NULL; Index++) {
+               if ((X<Driver>_ConfigTable[Index].BaseAddress == BaseAddress) ||
+                    !BaseAddress) {
+                       CfgPtr = &X<Driver>_ConfigTable[Index];
+                       break;
+               }
+       }
+
+       return CfgPtr;
+}
+#endif
```

Example Changes in the CSUDMA Driver

The following code is provided as a sample:

```
+#ifndef SDT
 extern XCsuDma_Config XCsuDma_ConfigTable[XPAR_XCSUDMA_NUM_INSTANCES];
+#else
+extern XCsuDma_Config XCsuDma_ConfigTable[];
+#endif

+#ifndef SDT
 XCsuDma_Config *XCsuDma_LookupConfig(u16 DeviceId);
```

```
+#else
+XCsuDma_Config *XCsuDma_LookupConfig(UINTPTR BaseAddress);
+#endif


+#ifndef SDT
 #include "xparameters.h"
+#endif

 /************************* Constant Definitions *****************************/

@@ -65,6 +67,7 @@
 *
 * @note                None.
 ******************************************************************************/
+#ifndef SDT
 XCsuDma_Config *XCsuDma_LookupConfig(u16 DeviceId)
 {
        XCsuDma_Config *CfgPtr = NULL;
@@ -81,4 +84,22 @@ XCsuDma_Config *XCsuDma_LookupConfig(u16 DeviceId)

        return (XCsuDma_Config *)CfgPtr;
 }
+#else
+XCsuDma_Config *XCsuDma_LookupConfig(UINTPTR BaseAddress)
+{
+       XCsuDma_Config *CfgPtr = NULL;
+       u32 Index;
+
+       /* Checks all the instances */
+       for (Index = (u32)0x0; XCsuDma_ConfigTable[Index].Name != NULL; Index++) {
+               if ((XCsuDma_ConfigTable[Index].BaseAddress == BaseAddress) ||
+                    !BaseAddress) {
+                       CfgPtr = &XCsuDma_ConfigTable[Index];
+                       break;
+               }
+       }
+
+       return (XCsuDma_Config *)CfgPtr;
+}
+#endif
```

Examples Folder in Drivers

The following changes are required in the `examples` folder:

- Update the `LookUp_Config` API in all the driver examples to use `BaseAddress` instead of `DeviceID`.
- In the system device tree based flow, interrupt registration has been simplified and simpler APIs are provided to offload interrupt registration process. Therefore, all references to interrupt handling in the example must be removed.
    - Use `XSetupInterruptSystem()` for registering the driver/application handler and `XDisconnectInterruptCntrl` for disabling interrupts.
    - Add `interrupts` and `interrupt-parent` fields to the driver `*_Config` structure.

  Refer to the changes in the following `xcsudma_intr_example.c` file for more information.

Sample changes for porting CSUDMA driver examples:

```
--- a/XilinxProcessorIPLib/drivers/csudma/data/csudma.yaml
+++ b/XilinxProcessorIPLib/drivers/csudma/data/csudma.yaml
@@ -31,4 +31,12 @@ required:
     - xlnx,dma-type
     - interrupts
     - interrupt-parent
+
+examples:
+    xcsudma_intr_example.c:
+        - interrupts
+    xcsudma_polled_example.c:
+        - reg
+    xcsudma_selftest_example.c:
+        - reg
```

```
--- a/XilinxProcessorIPLib/drivers/csudma/examples/
+++ b/XilinxProcessorIPLib/drivers/csudma/examples/xcsudma_intr_example.c
@@ -35,6 +35,7 @@
 * 1.9  sk     12/23/20 Add the documentation for XCsuDma_IntrExample() function
 *           parameters to fix the doxygen warning.
 * 1.11 sk     12/20/21 Add interrupt device id support for A78 and R52 processors.
+* 1.14  adk    05/04/23 Added support for system device-tree flow.
 * </pre>
 *
 ******************************************************************************/
@@ -44,11 +45,15 @@
 #include "xcsudma.h"
 #include "xparameters.h"
 #include "xil_exception.h"
+#ifndef SDT
 #ifdef XPAR_INTC_0_DEVICE_ID
 #include "xintc.h"
 #else
 #include "xscugic.h"
 #endif
+#else
+#include "xinterrupt_wrap.h"
+#endif

 /************************** Function Prototypes ******************************/

@@ -57,6 +62,7 @@
  * xparameters.h file. They are defined here such that a user can easily
  * change all the needed parameters in one place.
  */
+#ifndef SDT
 #define CSUDMA_DEVICE_ID  XPAR_XCSUDMA_0_DEVICE_ID /* CSU DMA device Id */
 #ifdef XPAR_INTC_0_DEVICE_ID
 #define INTC        XIntc
@@ -81,6 +87,9 @@
                         *  of CSU DMA device ID */
 #endif
 #endif
+#else
+#define CSUDMA_BASEADDR            XPAR_XCSUDMA_0_BASEADDR /* CSU DMA base address */
+#endif

 #define CSU_SSS_CONFIG_OFFSET  0x008       /**< CSU SSS_CFG Offset */
 #define CSUDMA_LOOPBACK_CFG    0x00000050  /**< LOOP BACK configuration
@@ -110,11 +119,15 @@ u32 SrcBuf[SIZE] __attribute__ ((aligned (64)));  /**< Source buffer
*/

 /************************** Function Prototypes ******************************/

+#ifndef SDT
 int XCsuDma_IntrExample(INTC *IntcInstancePtr, XCsuDma *CsuDmaInstance,
            u16 DeviceId, u16 IntrId);
 static int SetupInterruptSystem(INTC *IntcInstancePtr,
                XCsuDma *CsuDmaInstance,
                u16 CsuDmaIntrId);
+#else
+int XCsuDma_IntrExample(XCsuDma *CsuDmaInstance, UINTPTR BaseAddress);
+#endif
 void IntrHandler(void *CallBackRef);

 static void SrcHandler(void *CallBackRef, u32 Event);
@@ -124,8 +137,10 @@ static void DstHandler(void *CallBackRef, u32 Event);

 #ifndef TESTAPP_GEN
 XCsuDma CsuDma;        /**<Instance of the Csu_Dma Device */
+#ifndef SDT
 static INTC Intc;  /* Instance of the Interrupt Controller */
 #endif
+#endif
 volatile u32 DstDone = 0;

 #ifndef TESTAPP_GEN
@@ -146,8 +161,12 @@ int main(void)
    int Status;

    /* Run the selftest example */
+#ifndef SDT
    Status = XCsuDma_IntrExample(&Intc, &CsuDma, (u16)CSUDMA_DEVICE_ID,
                     INTG_CSUDMA_INTR_DEVICE_ID);
+#else
+   Status = XCsuDma_IntrExample(&CsuDma, CSUDMA_BASEADDR);
+#endif
    if (Status != XST_SUCCESS) {
        xil_printf("CSU_DMA Interrupt Example Failed\r\n");
        return XST_FAILURE;
@@ -178,8 +197,12 @@ int main(void)
 * @note        None.
 *
 ******************************************************************************/
+#ifndef SDT
 int XCsuDma_IntrExample(INTC *IntcInstancePtr, XCsuDma *CsuDmaInstance,
           u16 DeviceId, u16 IntrId)
+#else
+int XCsuDma_IntrExample(XCsuDma *CsuDmaInstance, UINTPTR BaseAddress)
+#endif
 {
    int Status;
    XCsuDma_Config *Config;
@@ -194,7 +217,11 @@ int XCsuDma_IntrExample(INTC *IntcInstancePtr, XCsuDma *CsuDmaInstance,
     * look up the configuration in the config table,
     * then initialize it.
     */
+#ifndef SDT
    Config = XCsuDma_LookupConfig(DeviceId);
+#else
+   Config = XCsuDma_LookupConfig(BaseAddress);
+#endif
    if (NULL == Config) {
        return XST_FAILURE;
    }
@@ -221,8 +248,15 @@ int XCsuDma_IntrExample(INTC *IntcInstancePtr, XCsuDma *CsuDmaInstance,
    /*
     * Connect to the interrupt controller.
     */
+#ifndef SDT
    Status = SetupInterruptSystem(IntcInstancePtr, CsuDmaInstance,
                      IntrId);
+#else
+   Status = XSetupInterruptSystem(CsuDmaInstance, &IntrHandler,
+                       CsuDmaInstance->Config.IntrId,
+                       CsuDmaInstance->Config.IntrParent,
+                       XINTERRUPT_DEFAULT_PRIORITY);
+#endif
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
@@ -311,6 +345,7 @@ int XCsuDma_IntrExample(INTC *IntcInstancePtr, XCsuDma *CsuDmaInstance,

 *
 ******************************************************************************/
+#ifndef SDT
 static int SetupInterruptSystem(INTC *IntcInstancePtr,
                XCsuDma *InstancePtr,
                u16 IntrId)
@@ -415,7 +450,7 @@ static int SetupInterruptSystem(INTC *IntcInstancePtr,
    return XST_SUCCESS;
 }

-
+#endif
```

```
--- a/XilinxProcessorIPLib/drivers/csudma/examples/xcsudma_polled_example.c
+++ b/XilinxProcessorIPLib/drivers/csudma/examples/xcsudma_polled_example.c
@@ -22,6 +22,7 @@
 * ----- ------  -------- ----------------------------------------------------
 * 1.0   vnsld   22/10/14 First release
 * 1.4   adk     04/12/17 Added support for PMC DMA.
+* 1.14  adk     04/05/23 Added support for system device-tree flow.
 * </pre>
 *
 *******************************************************************************/
@@ -38,7 +39,11 @@
  * xparameters.h file. They are defined here such that a user can easily
  * change all the needed parameters in one place.
  */
+#ifndef SDT
 #define CSUDMA_DEVICE_ID   XPAR_XCSUDMA_0_DEVICE_ID /* CSU DMA device Id */
+#else
+#define CSUDMA_BASEADDR        XPAR_XCSUDMA_0_BASEADDR /* CSU DMA Baseaddress */
+#endif
 #define CSU_SSS_CONFIG_OFFSET  0x008       /**< CSU SSS_CFG Offset */
 #define CSUDMA_LOOPBACK_CFG    0x00000050  /**< LOOP BACK configuration
                         *  macro */
@@ -61,7 +66,11 @@
 /************************** Function Prototypes ******************************/


+#ifndef SDT
 int XCsuDma_PolledExample(u16 DeviceId);
+#else
+int XCsuDma_PolledExample(UINTPTR BaseAddress);
+#endif

 /************************** Variable Definitions *****************************/

@@ -84,7 +93,11 @@ int main(void)
    int Status;

    /* Run the selftest example */
+#ifndef SDT
    Status = XCsuDma_PolledExample((u16)CSUDMA_DEVICE_ID);
+#else
+   Status = XCsuDma_PolledExample(CSUDMA_BASEADDR);
+#endif
    if (Status != XST_SUCCESS) {
        xil_printf("CSU_DMA Polled Example Failed\r\n");
        return XST_FAILURE;
@@ -110,7 +123,11 @@ int main(void)
 * @note        None.
 *
 ******************************************************************************/
+#ifndef SDT
 int XCsuDma_PolledExample(u16 DeviceId)
+#else
```

```
+int XCsuDma_PolledExample(UINTPTR BaseAddress)
+#endif
 {
    int Status;
    XCsuDma_Config *Config;
@@ -125,7 +142,11 @@ int XCsuDma_PolledExample(u16 DeviceId)
     * look up the configuration in the config table,
     * then initialize it.
     */
+#ifndef SDT
    Config = XCsuDma_LookupConfig(DeviceId);
+#else
+   Config = XCsuDma_LookupConfig(BaseAddress);
+#endif
```

```
--- a/XilinxProcessorIPLib/drivers/csudma/examples/xcsudma_selftest_example.c
+++ b/XilinxProcessorIPLib/drivers/csudma/examples/xcsudma_selftest_example.c
@@ -22,6 +22,7 @@
 *       ms     04/10/17 Modified filename tag to include the file in doxygen
 *                       examples.
 * 1.2   adk    11/22/17 Added peripheral test app support.
+* 1.14  adk    05/04/23 Added support for system device-tree flow.
 * </pre>
 *
 ******************************************************************************/
@@ -38,7 +39,11 @@
  * xparameters.h file. They are defined here such that a user can easily
  * change all the needed parameters in one place.
  */
+#ifndef SDT
 #define CSUDMA_DEVICE_ID  XPAR_XCSUDMA_0_DEVICE_ID /* CSU DMA device Id */
+#else
+#define CSUDMA_BASEADDR    XPAR_XCSUDMA_0_BASEADDR /* CSU DMA base address */
+#endif

 /**************************** Type Definitions *******************************/

@@ -48,7 +53,11 @@
 /************************** Function Prototypes ******************************/


+#ifndef SDT
 int XCsuDma_SelfTestExample(u16 DeviceId);
+#else
+int XCsuDma_SelfTestExample(UINTPTR BaseAddress);
+#endif

 /************************** Variable Definitions *****************************/

@@ -72,7 +81,11 @@ int main(void)
    int Status;

    /* Run the selftest example */
+#ifndef SDT
    Status = XCsuDma_SelfTestExample((u16)CSUDMA_DEVICE_ID);
+#else
+   Status = XCsuDma_SelfTestExample(CSUDMA_BASEADDR);
+#endif
    if (Status != XST_SUCCESS) {
        xil_printf("CSU_DMA Selftest Example Failed\r\n");
        return XST_FAILURE;
@@ -99,7 +112,11 @@ int main(void)
 * @note        None.
 *
 ******************************************************************************/
+#ifndef SDT
 int XCsuDma_SelfTestExample(u16 DeviceId)
+#else
+int XCsuDma_SelfTestExample(UINTPTR BaseAddress)
+#endif
```

```
 {
    int Status;
    XCsuDma_Config *Config;
@@ -109,7 +126,11 @@ int XCsuDma_SelfTestExample(u16 DeviceId)
     * look up the configuration in the config table,
     * then initialize it.
     */
+#ifndef SDT
    Config = XCsuDma_LookupConfig(DeviceId);
+#else
+   Config = XCsuDma_LookupConfig(BaseAddress);
+#endif
    if (NULL == Config) {
        return XST_FAILURE;
    }
```

Porting a Library

Each library contains the following folders:

- data
- src
- examples
- doc

The new flow requires some changes to be made to these folders. These changes are described in the following sections.

Data Folder in Libraries

In the new flow, the data folder contains a single YAML file that replaces the older proprietary metadata files (*.mld, *.tcl). As an example, XilFPGA library has a `xilfpga.yaml` file in the data folder.

- The YAML contains the following three sections:
  - Headers, copyrights, versions, and generic information such as maintainer and type.

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: <Description of the yaml file>
maintainers:
    - Name(s) of the maintainer(s)
type: <Describes the type of baremetal module the yaml file describes,
e.g, driver/library>
description: < A brief description of the library >
```

- Specifications that help in selecting the library for a given platform (processor/OS configuration).

  ```
  supported_processors:
      - <The list of CPUs the library supports>
  supported_os:
      - <The list of OSes the library supports>
  ```

- Specification(s) that describes the dependencies of the library. A library can depend on other driver/IP configurations which must be present in the device tree for the library to be used. In addition, a library can also depend on other libraries.

  ```
  depends:
  # Used in libraries like xilmailbox which depends on ipi or xiltimer
  which depends on timer related drivers
      <driver_name1>
          - <List of properties of driver/IP1 that library relies upon>
      <driver_name2>
          - <List of properties of driver/IP2 that library relies upon>

  depends_libs:
      <Library 1 that should be picked>: {}
      <Library 2 that should be picked>: {}
  ```

- Specifications that describe library example and their dependencies. Examples can depend on other supporting .h or .c files. There can be few examples only valid for few platforms. Examples listed here are also listed under the import example section in the GUI of the Vitis unified software platform 2023.2. Refer to the sample `xilfpga.yaml` below for more information.

  ```
  examples:
      <Example name>:
          - <supported_platforms>:
              - <has to be mentioned only when example is applicable to
  specific platform>
          - <dependency_files>:
              - <List of supported files that need to be pulled while
  creating the example>
  ```

- From 2025.2 onwards, ability to pick supported_processors, depends and the examples conditionally has been added. Refer Appendix H: Conditional Settings in Library YAMLs for the advanced usage of conditions for these three YAML sections.

- The sample XILFPGA YAML has the following contents in `xilfpga.yaml`.

  ```

  # SPDX-License-Identifier: MIT
  %YAML 1.2
  ---
  title: Bindings for xilfpga library.

  maintainers:
      - Zxcv Yuiop <zxcv.yuiop@domain.com>

  type: library

  description: |-
      XilFPGA library provides an interface to the Linux or bare-metal
  users for configuring the PL over PCAP from PS.
  ```

```
supported_processors:
    - psu_cortexa53
    - psu_cortexr5
    - psu_pmu
    - psv_cortexa72
    - psv_cortexr5

supported_os:
    - standalone
    - freertos10_xilinx

depends_libs:
    xilsecure: {}
    xilmailbox: {}

examples:
    xfpga_partialbitstream_load_example.c: []
    xfpga_readback_example.c:
        - supported_platforms:
            - ZynqMP
    xfpga_reg_readback_example.c:
        - supported_platforms:
            - ZynqMP
    xfpga_load_bitstream_example.c:
        - supported_platforms:
            - ZynqMP
            - Versal
```

Src Folder in Libraries

Replacing the Makefile

Instead of a Makefile, every library should have a `CMakeLists.txt` file that looks as shown in the following snippet.

```
# SPDX-License-Identifier: MIT
cmake_minimum_required(VERSION 3.15)
project(<library name>)

find_package(common)
# More details on <library name>.cmake follow in the next sub-section
include(${CMAKE_CURRENT_SOURCE_DIR}/<library name>.cmake NO_POLICY_SCOPE)
collector_create (PROJECT_LIB_HEADERS "${CMAKE_CURRENT_SOURCE_DIR}")
collector_create (PROJECT_LIB_SOURCES "${CMAKE_CURRENT_SOURCE_DIR}")
include_directories(${CMAKE_BINARY_DIR}/include)
# Add subdirectories if there are any within src folder
# Note that the subdirectories ought to have their own CMakeLists.txt to
collect the sources and headers
add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/<subdir_1>)
collect (PROJECT_LIB_SOURCES <source file>)
collect (PROJECT_LIB_HEADERS <header file>)
collector_list (_sources PROJECT_LIB_SOURCES)
collector_list (_headers PROJECT_LIB_HEADERS)
file(COPY ${_headers} DESTINATION ${CMAKE_BINARY_DIR}/include)
add_library(<library name> STATIC ${_sources})
set_target_properties(<library name> PROPERTIES LINKER_LANGUAGE C)
install(TARGETS <library name> LIBRARY DESTINATION ${CMAKE_SOURCE_DIR}/
build ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR})
```

```
get_headers(${_headers})
set_target_properties(<library name> PROPERTIES
ADDITIONAL_CLEAN_FILES "${CMAKE_LIBRARY_PATH}/lib<library name>.a;$
{CMAKE_INCLUDE_PATH}/<rel_headers>;${clean_headers}")
install(TARGETS <library name> LIBRARY DESTINATION ${CMAKE_LIBRARY_PATH}
ARCHIVE DESTINATION ${CMAKE_LIBRARY_PATH})
install(DIRECTORY ${CMAKE_BINARY_DIR}/include DESTINATION $
{CMAKE_INCLUDE_PATH}/..)
```

As an example, the XilFPGA library has a file named `CMakeLists.txt` that has the following contents. For more details `CMAKE_MACHINE` and `CMAKE_SYSTEM_PROCESSOR`, see Appendix F: CMAKE_MACHINE and CMAKE_SYSTEM_PROCESSOR.

```
# SPDX-License-Identifier: MIT
cmake_minimum_required(VERSION 3.3)
project(xilfpga)
include(${CMAKE_CURRENT_SOURCE_DIR}/xilfpga.cmake NO_POLICY_SCOPE)
find_package(common)
collector_create (PROJECT_LIB_SOURCES "${CMAKE_CURRENT_SOURCE_DIR}")
collector_create (PROJECT_LIB_HEADERS "${CMAKE_CURRENT_SOURCE_DIR}")
include_directories(${CMAKE_BINARY_DIR}/include)
if("${CMAKE_MACHINE}" STREQUAL "Versal")
add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/interface/versal/)
else()
add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/interface/zynqmp/)
endif()
collect (PROJECT_LIB_SOURCES xilfpga.c)
collect (PROJECT_LIB_HEADERS xilfpga.h)
collector_list (_sources PROJECT_LIB_SOURCES)
collector_list (_headers PROJECT_LIB_HEADERS)
file(COPY ${_headers} DESTINATION ${CMAKE_BINARY_DIR}/include)
if (${NON_YOCTO})
file(COPY ${CMAKE_BINARY_DIR}/include/xfpga_config.h DESTINATION $
{CMAKE_INCLUDE_PATH}/)
endif()
add_library(xilfpga STATIC ${_sources})
set_target_properties(xilfpga PROPERTIES LINKER_LANGUAGE C)
get_headers(${_headers})
set_target_properties(xilfpga PROPERTIES ADDITIONAL_CLEAN_FILES "$
{CMAKE_LIBRARY_PATH}/libxilfpga.
a;${CMAKE_INCLUDE_PATH}/xfpga_config.h;${clean_headers}")
install(TARGETS xilfpga LIBRARY DESTINATION ${CMAKE_LIBRARY_PATH} ARCHIVE
DESTINATION
${CMAKE_LIBRARY_PATH})
install(DIRECTORY ${CMAKE_BINARY_DIR}/include DESTINATION $
{CMAKE_INCLUDE_PATH}/..)
```

Adding New Configuration Files

If the library has software configuration, two new files, `<library_name>.cmake` and `x<library_name>_config.h.in`, are added to facilitate this process.

The system device tree based flow makes use of open source CMAKE-based flow and replaces the existing .mld/.tcl based flows to configure the library. `<library_name>.cmake` and `x<library_name>_config.h.in` generates the required software configuration header file `<library_name>_config.h` based on the user configuration.

- **x<library_name>_config.h.in:** The `cmakedefine` macro in the `config.h.in` file can be used to generate a series of define statements (macros). This configuration file when processed with `<library name>.cmake` results into a .h (header) file which contains all the generated `#define` statements in C syntax. An example `xfpga_config_h_in` file is shown in the following snippet:

  ```
  #ifndef _XFPGA_CONFIG_H
  #define _XFPGA_CONFIG_H

  #include <xilfpga.h>
  @PCAP_INCLUDE@
  @PCAP_COMMON_INCLUDE@
  @VERSAL_INCLUDE@

  #cmakedefine XFPGA_OCM_ADDRESS  @XFPGA_OCM_ADDRESS@U
  #cmakedefine XFPGA_BASE_ADDRESS @XFPGA_BASE_ADDRESS@U
  #cmakedefine XFPGA_SECURE_MODE  @XFPGA_SECURE_MODE@
  #cmakedefine01 XFPGA_DEBUG  @XFPGA_DEBUG@
  #cmakedefine XFPGA_SECURE_READBACK_MODE @XFPGA_SECURE_READBACK_MODE@
  #cmakedefine XFPGA_SECURE_IPI_MODE_EN @XFPGA_SECURE_IPI_MODE_EN@

  #endif /* XFPGA_CONFIG_H */
  ```

  Refer to How To Write Platform Checks for more details on how to use the configuration files with CMAKE.

- **<library_name>.cmake:** Software configuration options can include a variety of data types. For example, they can be Boolean, hex, an entry among a list of possible options, or a number among a large range of numbers. All these software configurations have a default value which can later be changed. These choices can be described using the following CMAKE commands:

  - For configuring parameters that have Boolean properties, use `option(<option_variable> "help string describing option" [initial value])`. For more information, see option.
  - For configuring parameters whose values could be one in a list of values or a range of values, use `set(<variable> <value> [[CACHE <type> <docstring> [FORCE]] | PARENT_SCOPE])`. For more information, see set.

After you have set the CMAKE options and variables, call the `configure_file()` command. It calls the `config.h.in` file and generates the `<libray name>_config.h`. An example `xilfpga.cmake` is shown in the following snippet:

```
# SPDX-License-Identifier: MIT
cmake_minimum_required(VERSION 3.15)

option(XILFPGA_secure_mode "Enable secure Bitstream loading support" ON)
option(XILFPGA_secure_environment "Which is used to Enable the secure PL
configuration" OFF)
option(XILFPGA_secure_readback "Which is used to Enable the secure PL
configuration Read-back support" OFF)
option(XILFPGA_debug_mode "Which is used to Enable the Debug messages in
the library" OFF)
SET(XILFPGA_ocm_address 0xfffc0000 CACHE STRING "OCM Address which is used
for Bitstream Authentication")
SET(XILFPGA_base_address 0x80000 CACHE STRING "Bitstream Image Base
```

```
Address")

set(zynqmp_secure_env 0)
if("${CMAKE_MACHINE}" STREQUAL "Versal")
    set(VERSAL_INCLUDE "#include <xilfpga_versal.h>")
elseif("${CMAKE_MACHINE}" STREQUAL "ZynqMP")
    set(PLATFORM_ZYNQMP " ")
    if((NOT("${CMAKE_SYSTEM_PROCESSOR}" STREQUAL "pmu_microblaze")) AND
       ${XILFPGA_secure_environment})
        set(XFPGA_SECURE_IPI_MODE_EN " ")
        set(zynqmp_secure_env 1)
    else()
        set(PCAP_INCLUDE "#include <xilfpga_pcap.h>")
    endif()
    set(PCAP_COMMON_INCLUDE "#include <xilfpga_pcap_common.h>")
endif()

set(XFPGA_OCM_ADDRESS ${XILFPGA_ocm_address})
set(XFPGA_BASE_ADDRESS ${XILFPGA_base_address})
if (${XILFPGA_secure_mode})
   set(XFPGA_SECURE_MODE " ")
endif()
if (${XILFPGA_debug_mode})
   set(XFPGA_DEBUG " ")
endif()
if (${XILFPGA_secure_readback})
    set(XFPGA_SECURE_READBACK_MODE " ")
endif()

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/xfpga_config.h.in $
{CMAKE_BINARY_DIR}/include/xfpga_config.h)
```

Adding a New Header File

As discussed, a new file named `<libray name>_config.h` is created by the `configure_file()` API when the `<library_name>.cmake` and `x<library_name>_config.h.in` files are processed together. This separates the library specific configurations into `<libray name>_config.h` file which, in the existing flow, is located within `xparameters.h`. This file is auto-generated as mentioned earlier in this document. An example `xfpga_config.h` is shown in the following snippet:

```
#ifndef _XFPGA_CONFIG_H
#define _XFPGA_CONFIG_H

#include <xilfpga.h>
/* Empty line as the PCAP_INCLUDE couldn't be set in xilfpga.cmake */
/* Empty line as PCAP_COMMON_INCLUDE couldn't be set in xilfpga.cmake */
#include <xilfpga_versal.h>

#define XFPGA_OCM_ADDRESS   0xfffc0000U
#define XFPGA_BASE_ADDRESS  0x80000U
#define XFPGA_SECURE_MODE
#define XFPGA_DEBUG  0
/* #undef XFPGA_SECURE_READBACK_MODE */
/* #undef XFPGA_SECURE_IPI_MODE_EN */

#endif /* XFPGA_CONFIG_H */
```

Making Changes in the Existing Source Files (.c and .h)

The following changes are required:

- All the calls of `LookupConfig()` API of every driver must be updated to use `BaseAddress` instead of `DeviceId`.
- All the references to the `XPAR_<drv_name>_DEVICE_ID` macro must be replaced with the macro defined in the `x<library_name>_config.h` file.
- The newly generated `x<library_name>_config.h` must be included in the .c files wherever needed.

Sample changes are shown in the following code:

```
--- a/lib/sw_services/xilloader/src/common/xloader_auth_enc.c
+++ b/lib/sw_services/xilloader/src/common/xloader_auth_enc.c
@@ -110,6 +110,7 @@
  *       am   06/19/23 Added KAT error code for failure cases
  *       sk   07/06/23 Added Jtag DAP config support for Non-Secure Debug
  *       am   07/03/23 Added authentication optimization support
+*       ng   07/13/23 Added support for system device tree flow
  *
  * </pre>
  *
@@ -135,6 +136,7 @@
 #include "xsecure_init.h"
 #include "xloader_plat_secure.h"
 #include "xloader_plat.h"
+#include "xplmi_config.h"

 /************************** Constant Definitions ****************************/

@@ -668,7 +670,7 @@ int XLoader_ImgHdrTblAuth(XLoader_SecureParams *SecurePtr)
        SecurePtr->AcPtr = AuthCert;

        /** - Get DMA instance */
-       SecurePtr->PmcDmaInstPtr = XPlmi_GetDmaInstance((u32)PMCDMA_0_DEVICE_ID);
+       SecurePtr->PmcDmaInstPtr = XPlmi_GetDmaInstance(PMCDMA_0_DEVICE);
        if (SecurePtr->PmcDmaInstPtr == NULL) {
                Status = XPlmi_UpdateStatus(XLOADER_ERR_IHT_GET_DMA, 0);
                goto END;
@@ -790,7 +792,7 @@ int XLoader_ReadAndVerifySecureHdrs(XLoader_SecureParams *SecurePtr,
        XPlmi_Printf(DEBUG_INFO,
                "Loading secure image headers and partition headers\n\r");
        /* Get DMA instance */
-       SecurePtr->PmcDmaInstPtr = XPlmi_GetDmaInstance((u32)PMCDMA_0_DEVICE_ID);
+       SecurePtr->PmcDmaInstPtr = XPlmi_GetDmaInstance(PMCDMA_0_DEVICE);
        if (SecurePtr->PmcDmaInstPtr == NULL) {
                Status = XPlmi_UpdateStatus(XLOADER_ERR_HDR_GET_DMA, 0);
                goto ERR_END;
@@ -3218,7 +3220,7 @@ static int XLoader_AuthJtag(u32 *TimeOut)
                goto END;
        }

-       SecureParams.PmcDmaInstPtr = XPlmi_GetDmaInstance((u32)PMCDMA_0_DEVICE_ID);
+       SecureParams.PmcDmaInstPtr = XPlmi_GetDmaInstance(PMCDMA_0_DEVICE);
        if (SecureParams.PmcDmaInstPtr == NULL) {
                Status = XPlmi_UpdateStatus(XLOADER_ERR_AUTH_JTAG_GET_DMA, 0);
                goto END;
```

Example Folder in Libraries

Changes needed in library examples are similar to that needed for driver examples. Refer to Examples Folder in Drivers.

```
--- a/lib/sw_services/xilloader/data/xilloader.yaml
+++ b/lib/sw_services/xilloader/data/xilloader.yaml
@@ -25,4 +25,12 @@ depends_libs:
  xilpm: {}
  xilpdi: {}
  xilffs: {}

+
+examples:
+  xilloader_add_image_store_pdi_example.c:
+    - supported_platforms:
+      - Versal
+  xilloader_load_pdi_example.c:
+    - supported_platforms:
+      - Versal
+  xilloader_update_multiboot.c:
+    - supported_platforms:
+      - Versal
```

```
--- a/lib/sw_services/xilloader/examples/
xilloader_add_image_store_pdi_example.c
+++ b/lib/sw_services/xilloader/examples/
xilloader_add_image_store_pdi_example.c
@@ -20,6 +20,7 @@
  *       bsv  08/18/2022   Fix typo in CmdId
  * 1.1   sk   03/10/2023   Updated changes to command format
  *       sk   04/18/2023   Added support for versalnet
+ * 1.9   ng   06/21/2023   Added support for system device-tree flow
  */
 #include <stdio.h>
 #include "xil_printf.h"
@@ -27,6 +28,12 @@
 #include "xipipsu.h"
 #include "xil_cache.h"

+#ifdef SDT
+#define IOMODULE_DEVICE (XPAR_XIOMODULE_0_BASEADDR)
+#else
+#define IOMODULE_DEVICE (XPAR_IOMODULE_0_DEVICE_ID)
+#endif
+
 #if defined(VERSAL_NET)
 #define TARGET_IPI_INT_MASK    XPAR_XIPIPS_TARGET_PSX_PMC_0_CH0_MASK
 #else
@@ -82,7 +89,7 @@ static int DoIpiTest(void)
        Xil_DCacheDisable();

        /* Look Up the config data */
-       IpiCfgPtr = XIpiPsu_LookupConfig(0U);
+       IpiCfgPtr = XIpiPsu_LookupConfig(IOMODULE_DEVICE);
        if (NULL == IpiCfgPtr) {
                goto END;
        }
```

Porting a Template Application

Template applications contain the following folders:

- `data`
- `src`

The new flow requires some changes to be made to these folders. These changes are described in the following sections.

Data Folder in Template Applications

A YAML file named `<app_name>.yaml` must be added inside the `data` folder. This YAML file is a replacement of the older AMD proprietary metadata files (*.mss, *.tcl). For example, the `data` folder in the `hello_world` template folder must contain a file named `hello_world.yaml`.

The YAML file contains the following sections:

- Headers, copyrights, versions, and generic information such maintainer and type.

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: <Description of the yaml file>

maintainers:
  - Name(s) of the maintainer(s)

type: <Describes the type of baremetal module the yaml file describes,
e.g, apps>

description:
  < A brief description of the Application >
```

- Specification(s) that help(s) picking up the application properly for a given platform (processor/OS configuration).

```
supported_processors:
  - <The list of CPUs the library supports>


supported_os:
  - <The list of OSes the library supports>
```

- Specification(s) that describes the dependencies of the application. An application can depend on certain driver/IP configurations which must be present in the system device tree for the application to be compiled. In addition, an application can also depend on the libraries and can, by default, need specific library configurations depending upon the use case.

```
depends:
# e.g. lwip applications need atleast one ethernet IP in the design
    <driver_name1>
        - <List of properties of driver/IP1 that the application relies
upon>
    <driver_name2>
        - <List of properties of driver/IP2 that the application relies
upon>

depends_libs:
    <Library 1 that should be configured>:
        <lib1 param1>: <lib1 val1>
        <lib1 param2>: <lib1 param2_0>
        <Family Name from bsp.yaml>:
            <lib1 param2>: <lib1 param2_1>
    <Library 2 that should be picked>:
        <lib2 param1>: <lib2 val1>
```

- Specification(s) that describes the OS-level and linker-related requirements of the application. These are used when the application requires some change in the OS level parameter or the linker heap and stack size. For more details on linker generation, refer to the Appendix G: Application: Hardware Dependency and Linker Script Generation section.

```
os_config:
    <os name i.e. freertos or standalone>:
        <os param name e.g. freertos_total_heap_size>: <os param value
e.g. 262140>

linker_constraints:
# Default Stack and Heap size is set to 0x2000 for Platforms other than
Microblaze.
# For Microblaze, the sizes are set to 0x400.
    stack: <Required Stack size e.g. 0xA000>
    heap: <Required Heap size e.g. 0xA000>
```

- A sample application YAML (`freertos_lwip_udp_perf_client.yaml`):

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: FreeRTOS lwIP UDP Perf Client
maintainers:
    - Klmn rtyu <klmn.rtyu@domain.com>

type: apps

description: The FreeRTOS LwIP UDP Perf Client application is used for
creating UDP
    client and measure UDP uplink performance using light-weight IP stack
(lwIP). This
    application sets up the board to use default IP address 192.168.1.10,
with MAC address
    00:0a:35:00:01:02. This application creates UDP client on board and
make connection
    with UDP server running on host machine. It will display connection
```

```
information
    along with interim and average UDP statistics for data transfer.

supported_processors:
    - psu_cortexa53
    - psu_cortexr5
    - psv_cortexa72
    - psv_cortexr5
    - psx_cortexa78
    - psx_cortexr52
    - ps7_cortexa9
    - microblaze

supported_os:
    - freertos10_xilinx

depends:
    emaclite:
        - reg
        - interrupts
    axiethernet:
        - reg
        - interrupts
    emacps:
        - reg
        - interrupts

depends_libs:
    lwip213:
        lwip213_api_mode: SOCKET_API
        lwip213_dhcp_does_arp_check: true
        lwip213_dhcp: true
        lwip213_ipv6_enable: false
        lwip213_pbuf_pool_size: 16384
        lwip213_memp_n_pbuf: 1024
        lwip213_mem_size: 524288
        lwip213_n_tx_descriptors: 512
    xiltimer:
        XILTIMER_en_interval_timer: true

os_config:
    freertos:
        freertos_total_heap_size: 262140

linker_constraints:
    stack: 0xA000
    heap: 0xA000
```

Src Folder in Template Applications

A new file called `CMakeLists.txt` is introduced in this flow. Instead of Makefile, every application should have a `CMakeLists.txt` file. Following is a sample `CMakeLists.txt` for the hello_world template.

```
# SPDX-License-Identifier: MIT
cmake_minimum_required(VERSION 3.15)

# For more details on <App_name>Example.cmake, please refer Appendix G
include(${CMAKE_CURRENT_SOURCE_DIR}/Hello_worldExample.cmake)
include(${CMAKE_CURRENT_SOURCE_DIR}/UserConfig.cmake)
```

```
set(APP_NAME hello_world)
project(${APP_NAME})

enable_language(C ASM CXX)
find_package(common)

# Describe the dependency of this application on multiple archivers
collect(PROJECT_LIB_DEPS xilstandalone)
collect(PROJECT_LIB_DEPS xil)
collect(PROJECT_LIB_DEPS xiltimer)
collect(PROJECT_LIB_DEPS gcc)
collect(PROJECT_LIB_DEPS c)

collect (PROJECT_LIB_SOURCES platform.c)
collect (PROJECT_LIB_SOURCES helloworld.c)
collector_list (_sources PROJECT_LIB_SOURCES)
foreach (source ${_sources})
    get_filename_component(ext ${source} EXT)
    list(APPEND src_ext ${ext})
endforeach()

find_project_type ("${src_ext}" PROJECT_TYPE)

if("${PROJECT_TYPE}" STREQUAL "c++")
collect(PROJECT_LIB_DEPS stdc++)
endif()
collector_list (_deps PROJECT_LIB_DEPS)
list (APPEND _deps ${USER_LINK_LIBRARIES})

if("${PROJECT_TYPE}" STREQUAL "c++")
string (REPLACE ";" ",-l" _deps "${_deps}")
endif()
if(CMAKE_EXPORT_COMPILE_COMMANDS)
    set(CMAKE_CXX_STANDARD_INCLUDE_DIRECTORIES $
{CMAKE_CXX_IMPLICIT_INCLUDE_DIRECTORIES})
    set(CMAKE_C_STANDARD_INCLUDE_DIRECTORIES $
{CMAKE_C_IMPLICIT_INCLUDE_DIRECTORIES})
endif()
linker_gen("${CMAKE_CURRENT_SOURCE_DIR}/linker_files/")
string(APPEND CMAKE_C_FLAGS ${USER_COMPILE_OPTIONS})
string(APPEND CMAKE_CXX_FLAGS ${USER_COMPILE_OPTIONS})
string(APPEND CMAKE_C_LINK_FLAGS ${USER_LINK_OPTIONS})
string(APPEND CMAKE_CXX_LINK_FLAGS ${USER_LINK_OPTIONS})
set_source_files_properties(${_sources} OBJECT_DEPENDS "$
{CMAKE_LIBRARY_PATH}/*.a")
add_executable(${APP_NAME}.elf ${_sources})
set_target_properties(${APP_NAME}.elf PROPERTIES LINK_DEPENDS $
{CMAKE_SOURCE_DIR}/lscript.ld)
target_link_libraries(${APP_NAME}.elf -Wl,-T -Wl,\"${CMAKE_SOURCE_DIR}/
lscript.ld\" -L\"${CMAKE_SOURCE_DIR}/\" -L\"${CMAKE_LIBRARY_PATH}/\" -L\"$
{USER_LINK_DIRECTORIES}/\" -Wl,--start-group,-l${_deps} -Wl,--end-group)
target_compile_definitions(${APP_NAME}.elf PUBLIC $
{USER_COMPILE_DEFINITIONS})
target_include_directories(${APP_NAME}.elf PUBLIC $
{USER_INCLUDE_DIRECTORIES})
print_elf_size(CMAKE_SIZE ${APP_NAME})
```

In the existing source files (.c and .h), changes mentioned under the Src Folder in Drivers must be complied to in application sources as well. All the calls of `LookupConfig()` API of every driver must be updated to use `BaseAddress` instead of `DeviceId`. Sample changes are shown in the following code snippet:

```
--- a/lib/sw_apps/srec_spi_bootloader/src/bootloader.c
+++ b/lib/sw_apps/srec_spi_bootloader/src/bootloader.c
@@ -101,7 +102,11 @@ extern void init_stdout();
 uint8  grab_hex_byte (uint8 *buf);
 int FlashReadID(void);

+#if defined (SDT)
+#define SPI_DEVICE_BASEADDR     XPAR_AXI_QUAD_SPI_0_BASEADDR
+#else
 #define SPI_DEVICE_ID          XPAR_SPI_0_DEVICE_ID
+#endif

 /*
  * The instances to support the device drivers are global such that they
@@ -172,7 +177,11 @@ int main()
         * Initialize the SPI driver so that it's ready to use,
         * specify the device ID that is generated in xparameters.h.
         */
+#if defined (SDT)
+       Status = XSpi_Initialize(&Spi, SPI_DEVICE_BASEADDR);
+#else
        Status = XSpi_Initialize(&Spi, SPI_DEVICE_ID);
+#endif
        if(Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

--- a/lib/sw_apps/srec_spi_bootloader/src/platform.c
+++ b/lib/sw_apps/srec_spi_bootloader/src/platform.c
@@ -1,11 +1,14 @@
 #include "xparameters.h"
+#if !defined (SDT)
 #include "platform_config.h"
+#endif

-#ifdef STDOUT_IS_16550
+#if ( defined (STDOUT_IS_16550) || ( defined (SDT) && defined
(XPAR_STDIN_IS_UARTNS550)))
 #include "xuartns550_l.h"
 #endif

@@ -13,7 +16,7 @@ void
 init_stdout()
 {
        /* if we have a uart 16550, then that needs to be initialized */
-#ifdef STDOUT_IS_16550
+#if ( defined (STDOUT_IS_16550) || ( defined (SDT) && defined
(XPAR_STDIN_IS_UARTNS550)))
        XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, 9600);
        XUartNs550_SetLineControlReg(STDOUT_BASEADDR, XUN_LCR_8_DATA_BITS);
 #endif
```

FAQs

- **Why is AMD shifting to a system device tree based flow?:** The earlier approach of using .mdd, .mld, and .mss files along with the .xsa as a handoff file involved AMD proprietary tools that were required to build the software stack. This approach only uses open source file formats and tools once the handoff file is generated by the hardware persona.
- **How is the system device tree different than the Linux device tree?:** The Linux device tree contains only the Linux/APU (or MicroBlaze™ processor) related peripheral information at a time. The system device tree contains the whole system information (includes information related to all processors such as APU, RPU, MicroBlaze, and PMC)
- **How to check if the component is a driver, library, or an application?:** The `type` key in YAML determines this. Available options for this key are: `driver`, `library`, `apps`, and `os`.
- **How is the driver probed or picked while building libxil?:** Lopper backend checks for a given node compatible against the compatibility specified in the driver YAML file under the properties section. If there is a match, it picks the driver and generates `_g.c` file in the subsequent process.
- **How are the _g.c entries and the xparameters generated for drivers?:** Lopper backend generates the `_g.c` entries based on the properties specified in the "required" section of the YAML file. Make sure these entries adhere to the driver config structure order. Refer to Appendix C: Options Available in the "required" Section and Appendix D: Sample YAML for a Driver for more details.
- **If the config structure entry has a dependency on another node base address, then how to handle it?:** The metadata representation for this is `<property-name>: phandle`. If a property dependent on a child node is available, it reads the `reg` property of that node and generates its value in this entry; otherwise, it generates zero. Refer to Appendix C: Options Available in the "required" Section and Appendix D: Sample YAML for a Driver for more details.
- **If the config structure has more than one base address in sequence, then how to add that information in the metadata to generate proper addresses in the config structure?:** The metadata representation for the above requirement is `reg:<number of entries>`. Refer to Appendix C: Options Available in the "required" Section for more details.
- **If the property mentioned in the required section is not present in the device tree node, then what value is generated in the _g.c file?:** Zero (0x0)
- **How to add multiple configurations for a library?:** Addition of `<library_name>.cmake` and `x<library_name>_config.h.in` facilitates the library software configuration. Refer to the Src Folder in Libraries section for more information.

- **How to use platform or a processor specific piece of code?:** CMAKE_MACHINE and CMAKE_SYSTEM_PROCESSOR variables inside CMAKE can be used to differentiate a platform or a processor specific piece of code. Refer to Appendix F: CMAKE_MACHINE and CMAKE_SYSTEM_PROCESSOR to know the available values for this CMAKE variable.
- **Where are the library-related configuration-specific macros located? It is missing in xparameters.h?:** Those are relocated to the `<library>_config.h` file. Refer to the Src Folder in Libraries section for more information.
- **How to define dependencies of a library or an application?:** The `depends` and `depends_libs` keys in a library YAML describes the hardware and software dependency of the library, respectively. Refer Data Folder in Libraries (or under Porting a Template Application) section with Appendix F: CMAKE_MACHINE and CMAKE_SYSTEM_PROCESSOR for more details.
- **How to define the archiver dependencies of the application?:** `collect(PROJECT_LIB_DEPS xil)` statements in Application CMakeLists define this dependency. This statement means that the application depends on `libxil.a`.
- **How is the linker script generated?:** An intermediary file named `lscript.ld.in` kept within the embeddedsw repository is copied into the application source folder. This file and the `<Application>Example.cmake` generates the final `lscript.ld`. Refer to Appendix G: Application: Hardware Dependency and Linker Script Generation for more details.
- **How to set a different stack and heap size for the application that does not fit in the default size?:** The `linker_constraints` key in the application YAML is used to modify the default stack and heap sizes. Refer to Appendix G: Application: Hardware Dependency and Linker Script Generation for more details.

Known Issues

See the Wiki for known issues and resolutions.

Generating a System Device Tree Using SDTGen

The system device tree represents complete hardware information in the form of device trees. All peripheral information and its properties, memories, clusters, SoC peripheral information, and soft IP information that are available in the hardware design are represented in a Linux-like device tree structure.

SDTGen, also known as DTG++, is a Tcl based tool that uses HSI APIs to read the hardware information from the XSA and put it in device tree format. The Tcl source files for this package can be found in the `<Installed Vitis Path>/data/system-device-tree-xlnx` location. The tool sources the `device_tree.tcl` file from `<above_path>/device_tree/data/device_tree.tcl` and exports the following three procs as commands:

- `set_dt_param`: Takes user inputs such as the Vivado XSA file and the device tree output directory. It can also be used to set the device tree parameters such as the board file and custom dts file. Example usage:

```
# Setting the device tree parameters. Multiple parameter setting in one
line is also allowed.
sdtgen% set_dt_param -dir output_dts
sdtgen% set_dt_param -xsa design_1_wrapper.xsa -board_dts zcu102-rev1.0

# Enabling the trace i.e. the flow of tcl procs that are getting invoked
during sdt generation. The default trace option is disabled
sdtgen% set_dt_param -trace enable

#Command Help
sdtgen% set_dt_param -help
Usage: set/get_dt_param [OPTION]
    -xsa                    Vivado hw design file
    -board_dts              board specific file
    -dir                    Directory where the dt files will be generated
    -trace                  Enable sdtgen traces
```

- `get_dt_param`: Returns the value set for a given parameter. Example usage:

```
# Checking the dt parameters
sdtgen% get_dt_param -board_dts
zcu102-rev1.0
sdtgen% get_dt_param -dir
output_dts
sdtgen% get_dt_param -xsa
design_1_wrapper.xsa

sdtgen% get_dt_param -help
```

```
Usage: set/get_dt_param [OPTION]
    -xsa                       Vivado hw design file
    -board_dts                 board specific file
    -dir                       Directory where the dt files will be generated
    -trace                     Enable sdtgen traces
```

- `generate_sdt`: Generates the system device tree with the set parameters.

To generate a system device tree using SDTGen, follow these steps:

1. Launch SDTGen.
2. Execute your code. A usage example is shown below:

```
# Generic command for building any SDT targeting Bare-metal use cases.
Launch sdtgen from Vitis tool
linux# sdtgen
sdtgen% set_dt_param -dir dts -xsa design_1_wrapper.xsa
sdtgen% generate_sdt

# Command for building Zynq SDT targeting Linux use cases
Launch sdtgen from Vitis tool
linux# sdtgen
sdtgen% set_dt_param -board_dts zc702 -dir dts -xsa design_1_wrapper.xsa
sdtgen% generate_sdt
```

System Device Tree Output Directory

The generated dts directory can contain following files:

- `soc.dtsi`, which is a static SoC-specific file:
  - `versal.dtsi` for AMD Versal™ adaptive SoCs
  - `zynqmp.dtsi` for AMD Zynq™ UltraScale+™ MPSoCs
  - `zynq-7000.dtsi` for AMD Zynq™ 7000 devices
  - `versal2.dtsi` for AMD Versal 2VE and 2VM devices

  ***Note*:** An SoC-specific dtsi file is not available for MicroBlaze processors.
- `pl.dtsi` which contains information about soft IPs.
- `board.dtsi`, which is board specific. This is required only when the `-board_dts` option is set. For example: `versal-vck190-rev1.1.dtsi` for the Versal VCK190 board, `zcu102-rev1.0.dtsi` for the Zynq UltraScale+ MPSoC ZCU102 board, `zc702.dtsi` for the Zynq 7000 ZCU102 board, and `kc705-lite.dtsi` for MicroBlaze processors.
- `clk.dtsi`, which contains clocking information. For example, `versal-clk.dtsi` for Versal devices, `zynqmp-clk-ccf.dtsi` for Zynq UltraScale+ MPSoC devices. There are no clock framework files for Zynq 7000 devices and MicroBlaze processors.

- `system-top.dts`, which is a top level system information which contains memory, clusters, and aliases.
- `pcw.dtsi`, which contains peripheral configuration wizard information for the peripherals.
- `psu_init*` files for Zynq 7000 and Zynq UltraScale+ MPSoCs, and pdi files for Versal devices.
- For Versal designs, a folder with design name that contains the pdi_files contents.

Note: For more information on system device tree generation, see System Device Tree Generator (SDTGen / DTG++).

Appendix B

Usage Information for "compatible"

There are different ways to use the `compatible` string in the `properties` key of driver YAML files:

- When there is one IP and one driver:

```
properties:
  compatible:
    items:
      - const: xlnx,zynqmp-csudma-1.0
```

- When multiple IPs have the same driver:

```
properties:
  compatible:
    OneOf:
      - items:
        - enum:
          - xlnx,zynqmp-8.9a
          - arasan,sdhci-8.9a
        - enum:
          - xlnx,versal-8.9a
          - arasan,sdhci-8.9a
        - const:
          - xlnx,versal-net-emmc
```

- When the same IP has different versions:

```
properties:
  compatible:
    OneOf:
      - items:
        - enum:
          - xlnx,axi-cdma-4.1
          - xlnx,axi-cdma-1.00.a
```

Options Available in the "required" Section

The following options are available in the `required` section:

Key	Notes	Config Structure	XPARAMETERS	Usage Of Padding

compatible	The existing embedded software driver uses "DEVICE_ID" in the driver config structure to get the peripheral instance. This approach is now being replaced with Base Address.
	Base Address of the peripheral instance is derived used the reg property.
		Filler for DEVICE_ID.	`#define XPAR_{node_label_name}_COMPATIBLE {value}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_COMPATIBLE {value}`
				Not allowed

reg	Device tree nodes usually have reg property in <base_addr size> format.
	This reg property is used to derive BASE_ADDRESS and HIGH_ADDRESS of the IP instance.
		Filler for Base Address.	`#define XPAR_{node_label_name}_BASEADDR {hex(base_addr)}`
			`#define XPAR_{node_label_name}_HIGHADDR {hex(base_addr + size -1)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_BASEADDR {hex(base_addr)}`
			`#define XPAR_X{driver_name}_{index}_HIGHADDR {hex(base_addr + size -1)}`
				For example: reg: 2.
				This is needed if the config structure has more than one base address in sequence. Check GIC node and its config structure for more information.

Key	Notes	Config Structure	XPARAMETERS	Usage Of Padding

interrupts	Interrupt property format for a mapped interrupt in a system device tree looks like: `<PPI/SPI interrupt_id trigger_type>`. This property format is converted to generate the Interrupt Property which has the following format:
	bits[11:0] interrupt-id
	bits[15:12] trigger type and level flags
	bits[19:16] CPU Mask
	bit[20] interrupt-type (1: PPI, 0: SPI)
		Generated the interrupt hex value.	`#define XPAR_{node_label_name}_INTERRUPTS {hex(derived_intr_format)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_INTERRUPTS {hex(derived_intr_format)}`
				For example, interrupts: 2.
				This is needed when a peripheral can have multiple interrupts attached.
				• Config_structure: A C list containing all the interrupts in the earlier described format is added to the config structure.
				• XPARAMETERS:
				◦ `#define XPAR_{node_label_name}_INTERRUPTS_<x> {hex(derived_intr_format[x])}`, where x ranges from 0 to padding - 1.
				◦ Examples:
				- `#define XPAR_{node_label_name}_INTERRUPTS_0 {hex(derived_intr_format[0])}`
				- `#define XPAR_{node_label_name}_INTERRUPTS_1 {hex(derived_intr_format[1])}`

interrupt-parent	Adds the base address of the interrupt-parent, which is usually the base address of GIC or INTC.	Base address of the interrupt-parent.	`#define XPAR_{node_label_name}_INTERRUPT_PARENT {hex(intr_parent_addr)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_INTERRUPT_PARENT {hex(intr_parent_addr)}`
				Not allowed

Key	Notes	Config Structure	XPARAMETERS	Usage Of Padding

child,required	Generates the sub nodes/arrays.	A C array of child node's property values.	`#define XPAR_{node_label_name}_{j}_{p.upper()} {val}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{j}_{<property>.upper()} {val}`
				Not allowed

Key	Notes	Config Structure	XPARAMETERS	Usage Of Padding

Any other property (<property> or xlnx,<property>)	Property value is of type string.	Value is used as it is.	`#define XPAR_{node_label_name}_{<property>.upper()} {string value}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{<property>.upper()} {string value}`
				For example: xlnx,num_slcr_addr: 2.
				• Mainly used for 64-bit addresses and floating values converted to hex.
				• Concatenates multiple 32-bit hex number/addresses into one 64-bit hex number/ address.
				• XPARAMETERS: `#define XPAR_{label_name}_{<property>.upper()} {combined_address_prop}`

	Property value is of type phandle:
	If the IP property points to a different device-tree node reference, the YAML key must be denoted as: `<property>: phandle`.
		Base Address of the IP on which this property depends is used.	`#define XPAR_{node_label_name}_{prop.upper()} {hex(base_addr of the new IP)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{prop.upper()} {hex(base_addr of the new IP)}`	

	Property value is of type int.	Convert into hex and use the hex value.	`#define XPAR_{node_label_name}_{<property>.upper()} {hex(int_value)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{<property>.upper()} {hex(int_value)}`	

	Property value is of type list.	List values are fomatted as a C list, int value is converted to hex, while string value is used as default.	`#define XPAR_{node_label_name}_{<property>.upper()}_<x> {hex(int_value)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{<property>.upper()}_<x> {hex(int_value)}`	

Key	Notes	Config Structure	XPARAMETERS	Usage Of Padding

Any other property (<property> or xlnx,<property>)	Property value is of type /bits/ which is a Linux-specific way to show certain properties.	Hex value kept at the end of the property value is converted into int with base 16.	`#define XPAR_{node_label_name}_{<property>.upper()}_<x> {hex(int_value)}`
			Canonical define:
			`#define XPAR_X{driver_name}_{index}_{<property>.upper()}_<x> {hex(int_value)}`
				For example: xlnx,num_slcr_addr: 2.
				• Mainly used for 64-bit addresses and floating values converted to hex.
				• Concatenates multiple 32-bit hex number/addresses into one 64-bit hex number/ address.
				• XPARAMETERS: `#define XPAR_{label_name}_{<property>.upper()}{combined_address_prop}`

Note: Check canonical macros in the `xparameters.h` file. In the legacy flow, drivers had canonical macros defined in different ways. These macros have now been made uniform across drivers and follows the XPAR_X<driver_name>_<instance_number>_<PROPERTY> syntax. Changes are therefore needed to conform to the new syntax. If you want to have a name that is different from the IP name inside canonical definition, you must define the “xparam_prefix” key with that name. X<driver_name> is replaced with the value of xparam_prefix in the new syntax.

Sample YAML for a Driver

Following are some sample YAMLs for drivers:

- A comprehensive sample YAML file for drivers. Note that data points are only for syntax-related understanding and are not related to any IP.

```
# SPDX-License-Identifier: MIT
%YAML 1.2
---
title: Bindings for IPI controller available in the zynqMP and Versal
platform

maintainers:
    - ASDF GHJKL <asdf.ghjkl@domain.com>
    - ZXCV VBNM <zxcv.vbnm@domain.com>

type: driver

device_type: ipi

properties:
    compatible:
        items:
            OneOf:
                - items:
                    - enum:
                        - xlnx,zynqmp-ipi-mailbox
                        - xlnx,psu-ipi-1.0
    reg:
        description: Standard reg property
    ipi-bitmask:
        description: foo0 bar0 abc0 xyz0
    interrupts:
        description: foo1 bar1 abc1 xyz1
    interrupt-parent:
        description: foo2 bar2 abc2 xyz2
    num_slcr_addr:
        description: foo3 bar3 abc3 xyz3
    axistream-connected:
        description: foo4 bar4 abc4 xyz4
    child-required:
        description: foo5 bar5 abc5 xyz5
    ipi-id:
        description: foo6 bar6 abc6 xyz6
    max-intr-size:
        description: foo7 bar7 abc7 xyz7

config:
    - XIpiPsu_Config

required:
    - compatible
    - reg
    - xlnx,ipi-bitmask
    - interrupts
    - interrupt-parent
    - xlnx,num_slcr_addr: 2
    - axistream-connected: phandle
    - child,required:
        - xlnx,ipi-bitmask
        - xlnx,ipi-buf-index

optional:
    - xlnx,ipi-id

additionalProperties:
    - xlnx,max-intr-size

depends:
    clockps: []
    resetps: []

depends_libs:
    libmetal: {}

examples:
    xdrv1_selftest_example.c::
        - reg
    xdrv1_polltimeout_interrupt_example.c:
        - dependency_files:
            - xdrv1_flash_config.h
        - interrupts
        - compatible : xlnx,versal-wwdt-1.0
    xdrv1_generic_interrupt_example.c:
        - interrupts
```

- A sample device tree data for an example IP:

```
ipi3: mailbox@ff330000 {
    status = "okay";
    compatible = "xlnx,zynqmp-ipi-mailbox";
    interrupt-parent = <&gic_a72>;
    interrupts = <0x0 0x1e 0x4>;
    reg = <0x0 0xff330000 0x0 0x20>;
    xlnx,ipi-bitmask = <0x4>;
    xlnx,ipi-id = <0x2>;
    xlnx,ipi-buf-index = <0x2>;
    xlnx,ip-name = "psv_ipi";
    xlnx,num_slcr_addr = <0x00000006 0x00000000>;
    xlnx,max-intr-size = <0x20>;
    xlnx,name = "CIPS_0_pspmc_0_psv_ipi_0";
    axistream-connected = <&axi_dma_0>;
    phandle = <0x6d>;

    ipi0_0: child@0 {
            xlnx,ipi-bitmask = <0x4>;
            xlnx,ipi-rsp-msg-buf = <0xff3f04a0>;
            xlnx,ipi-id = <0x2>;
            xlnx,ipi-buf-index = <0x2>;
            xlnx,ipi-req-msg-buf = <0xff3f0480>;
            phandle = <0xe1>;
    };

    ipi0_1: child@1 {
            xlnx,ipi-bitmask = <0x8>;
```

```
                xlnx,ipi-rsp-msg-buf = <0xff3f04e0>;
                xlnx,ipi-id = <0x3>;
                xlnx,ipi-buf-index = <0x3>;
                xlnx,ipi-req-msg-buf = <0xff3f04c0>;
        };
}

axi_dma_0: axi_dma@a4050000 {
        xlnx,num-s2mm-channels = <1>;
        xlnx,ip-name = "axi_dma";
        reg = <0x0 0xa4050000 0x0 0x10000>;
        xlnx,s2mm-burst-size = <16>;
};
```

- Generated Config structure file for the IPIPSU driver:

```
#include "xipipsu.h"

XIpiPsu_Config XIpiPsu_ConfigTable[] __attribute__ ((section
(".drvcfg_sec"))) = {
        {
                "xlnx,zynqmp-ipi-mailbox", /* compatible */
                0xff330000, /* reg */
                0x4, /* xlnx,ipi-bitmask */
                0x401e, /* interrupts */
                0xf9000000, /* interrupt-parent */
                0x600000000, /* xlnx,num_slcr_addr */
                0xa4050002, /* axistream-connected */
                {
                        {
                                4, /* xlnx,ipi-bitmask */
                                2 /* xlnx,ipi-buf-index */
                        },
                        {
                                8, /* xlnx,ipi-bitmask */
                                3 /* xlnx,ipi-buf-index */
                        }
                },
                0x2 /* xlnx,ipi-id */
        },
        {
                 NULL
        }
};
```

- Generated `xparameters.h` for the IPIPSU driver:

```
/* Definitions for peripheral IPI3 */
#define XPAR_IPI3_COMPATIBLE xlnx,zynqmp-ipi-mailbox
#define XPAR_IPI3_BASEADDR 0xff330000
#define XPAR_IPI3_HIGHADDR 0xff33001f
#define XPAR_IPI3_IPI_BITMASK 0x4
#define XPAR_IPI3_INTERRUPTS 0x401e
#define XPAR_IPI3_INTERRUPT_PARENT 0xf9000000
#define XPAR_IPI3_IPI_NUM_SLCR_ADDR 0x600000000
#define XPAR_IPI3_0_AXISTREAM-CONNECTED 0xa4050002
#define XPAR_IPI3_0_IPI_BITMASK 0x4
#define XPAR_IPI3_0_IPI_BUF_INDEX 0x2
#define XPAR_IPI3_1_IPI_BITMASK 0x8
#define XPAR_IPI3_1_IPI_BUF_INDEX 0x3
#define XPAR_IPI3_MAX_INTR_SIZE 0x20

/* Canonical definitions for peripheral IPI3 */
```

```
#define XPAR_XIPIPSU_3_BASEADDR 0xff330000
#define XPAR_XIPIPSU_3_HIGHADDR 0xff33001f
#define XPAR_XIPIPSU_3_COMPATIBLE xlnx,zynqmp-ipi-mailbox
#define XPAR_XIPIPSU_3_IPI_BITMASK 0x4
#define XPAR_XIPIPSU_3_INTERRUPTS 0x401e
#define XPAR_XIPIPSU_3_INTERRUPT_PARENT 0xf9000000
#define XPAR_XIPIPSU_3_IPI_NUM_SLCR_ADDR 0x600000000
#define XPAR_XIPIPSU_3_AXISTREAM-CONNECTED 0xa4050002
#define XPAR_XIPIPSU_3_MAX_INTR_SIZE 0x20
```

***Note*:** Canonical entries follow the `XPAR_X<driver_name>_<instance_number>_<PROPERTY>` syntax.

Including a Driver Example in the Peripheral Test Application

Changes to the Data Folder of the Driver

The following changes should be made to the `data` folder of the driver:

- Add a tapp section in the `<driver>.yaml` as mentioned in the Data Folder in Drivers section. In the existing flow, `<driver>_tapp.tcl` used to facilitate this action. Following is a sample "tapp" section in a `<driver>.yaml`:

```
tapp:
    x<driver>_selftest_example.c:
        declaration: X<Driver>_SelfTestExample
    x<driver>_generic_interrupt_example.c:
        declaration: X<Driver>_IntrExample
        hwproperties:
            - interrupts
```

- All the function declarations mentioned in the above tapp section must be available in the `<driver>_header.h` file.
- If the examples have dependency on other files those can be mentioned via dependency_files syntax. The example changes are shown in the following code snippet:

```
tapp:
    xaxiethernet_example_intr_sgdma.c:
        dependency_files:
            - xaxiethernet_example_util.c
            - xaxiethernet_example.h
        axistream-connected: 2
        declaration: AxiEthernetSgDmaIntrExample
        hwproperties:
             - interrupts

    xaxiethernet_example_intr_mcdma.c:
        dependency_files:
            - xaxiethernet_example_util.c
            - xaxiethernet_example.h
        axistream-connected: 3
        hwproperties:
             - interrupts
```

- For self tests, example function declaration contains only one argument, `BaseAddress`.
- For other example functions, declaration contains two arguments, the instance name and `BASEADDRESS`.

Example changes are shown in the following code snippets:

```
--- a/XilinxProcessorIPLib/drivers/csudma/data/csudma.yaml
+++ b/XilinxProcessorIPLib/drivers/csudma/data/csudma.yaml
@@ -39,4 +39,13 @@ examples:
         - reg
     xcsudma_selftest_example.c:
         - reg
+
+tapp:
+   xcsudma_selftest_example.c:
+        declaration: XCsuDma_SelfTestExample
+   xcsudma_intr_example.c:
+        declaration: XCsuDma_IntrExample
+        hwproperties:
+            - interrupts
+...
```

```
--- a/XilinxProcessorIPLib/drivers/csudma/data/csudma_header.h
+++ b/XilinxProcessorIPLib/drivers/csudma/data/csudma_header.h
@@ -11,6 +11,10 @@
 #include "xil_assert.h"
 #include "xstatus.h"

+#ifdef SDT
+int XCsuDma_SelfTestExample(UINTPTR BaseAddress);
+int XCsuDma_IntrExample(XCsuDma *CsuDmaInstance, UINTPTR BaseAddress);
+#else
 int XCsuDma_SelfTestExample(u16 DeviceId);
 #ifdef XPAR_SCUGIC_0_DEVICE_ID
 int XCsuDma_IntrExample(XScuGic *IntcInstancePtr, XCsuDma *CsuDmaInstance,
@@ -22,3 +26,4 @@ int XCsuDma_IntrExample(XIntc *IntcInstancePtr, XCsuDma
*CsuDmaInstance,
 #endif
 #endif
 #endif
+#endif
```

Changes to the Example Folder of the Driver

Refer to the Examples Folder in Drivers section.

CMAKE_MACHINE and CMAKE_SYSTEM_PROCESSOR

To differentiate a platform or a processor specific piece of code in CMAKE, `CMAKE_MACHINE` and `CMAKE_SYSTEM_PROCESSOR` variables can be used. A list of available `CMAKE_MACHINE` values is shown in the following table:

Table 1: **CMAKE_MACHINE Values**

Platform Name	CMAKE_MACHINE
AMD Zynq™ 7000 devices	Zynq
AMD Zynq™ UltraScale+™ MPSoCs	ZynqMP
AMD Versal™ adaptive SoCs	Versal
Kintex or MicroBlaze™ boards	Name depends on family of the design. It can be kintex7, kintexu, virtex, etc.
VersalNet and Versal 2VE_2VM devices	VersalNet

A list of available `CMAKE_SUBMACHINE` values is shown in the following table:

Table 2: **CMAKE_SUBMACHINE Values**

Platform Name	CMAKE_SUBMACHINE
VersalNet devices	VersalNet
Versal 2VE and 2VM Devices	Versal_2VE_2VM

A list of available `CMAKE_SYSTEM_PROCESSOR` values is shown in the following table:

Table 3: **CMAKE_SYSTEM_PROCESSOR Values**

Processor Name	CMAKE_SYSTEM_PROCESSOR (value)
Soft MicroBlaze	microblaze
Soft RISCV MicroBlaze	microblaze_riscv
PMU MicroBlaze	pmu_microblaze
PSM MicroBlaze	microblaze
PLM MicroBlaze	plm_microblaze
Arm® Cortex®-A53	cortexa53
Arm Cortex-R5F	cortexr5

Table 3: **CMAKE_SYSTEM_PROCESSOR Values** *(cont'd)*

Processor Name	CMAKE_SYSTEM_PROCESSOR (value)
Arm Cortex-A72	cortexa72
Arm Cortex A9	cortexa9
Arm Cortex-A78	cortexa78
Arm Cortex-R52	cortexr52

Application: Hardware Dependency and Linker Script Generation

The `<Component>Example.cmake` file is an auto-generated file inside the library and application's `src` folders. It contains the required peripheral information in the CMAKE format. The Lopper backends deduce the required peripherals data mentioned under the "depends" section from the system device tree and retains them in the `Example.cmake` file. The `CMakeLists.txt` located in the `src` folder includes these data into the build system and generates the required software level configurations.

For applications, the CMAKE also contains the available memory related information which translates into the linker script. Together with the `lscript.ld.in` file, the CMAKE results into the `lscript.ld` file for the final application.

Sample Dependencies and Linker Sections

```
depends:
    emaclite:
        - reg
        - interrupts
    axiethernet:
        - reg
        - interrupts
    emacps:
        - reg
        - interrupts
    tmrctr:
        - reg
        - interrupts
    ttcps:
        - reg
        - interrupts

# Linker Constraints belong only to Application.
linker_constraints:
    stack: 0xA000
    heap: 0xA000
```

Sample Lwip_echo_serverExample.cmake

```
set(AXIETHERNET_NUM_DRIVER_INSTANCES "")
set(EMACLITE_NUM_DRIVER_INSTANCES "")
set(EMACPS_NUM_DRIVER_INSTANCES
"CIPS_0_pspmc_0_psv_ethernet_0;CIPS_0_pspmc_0_psv_ethernet_1")
set(EMACPS0_PROP_LIST "0xff0c0000;0x4038")
```

```
list(APPEND TOTAL_EMACPS_PROP_LIST EMACPS0_PROP_LIST)
set(EMACPS1_PROP_LIST "0xff0d0000;0x403a")
list(APPEND TOTAL_EMACPS_PROP_LIST EMACPS1_PROP_LIST)
set(TMRCTR_NUM_DRIVER_INSTANCES "")
set(noc_lpddr4_ddr_memory_DDR_CH_1 "0x50000000000;0x200000000")
set(noc_ddr4_ddr_memory_DDR_LOW_1 "0x800000000;0x180000000")
set(noc_ddr4_ddr_memory_DDR_LOW_0 "0x0;0x80000000")
set(CIPS_0_pspmc_0_psv_ocm_ram_0_memory_0 "0xfffc0000;0x40000")
set(DDR noc_ddr4_ddr_memory_DDR_LOW_0)
set(CODE noc_ddr4_ddr_memory_DDR_LOW_0)
set(DATA noc_ddr4_ddr_memory_DDR_LOW_0)
set(TOTAL_MEM_CONTROLLERS
"noc_lpddr4_ddr_memory_DDR_CH_1;noc_ddr4_ddr_memory_DDR_LOW_1;noc_ddr4_ddr_m
emory_DDR_LOW_0;CIPS_0_pspmc_0_psv_ocm_ram_0_memory_0")
set(MEMORY_SECTION "MEMORY
{
    psv_pmc_ram_psv_pmc_ram : ORIGIN = 0xF2000000, LENGTH = 0x20000
    psv_r5_0_atcm_global_MEM_0 : ORIGIN = 0xFFE00000, LENGTH = 0x40000
    psv_r5_1_atcm_global_MEM_0 : ORIGIN = 0xFFE90000, LENGTH = 0x10000
    psv_r5_1_btcm_global_MEM_0 : ORIGIN = 0xFFEB0000, LENGTH = 0x10000
    noc_lpddr4_ddr_memory_DDR_CH_1 : ORIGIN = 0x50000000000, LENGTH =
0x200000000
    noc_ddr4_ddr_memory_DDR_LOW_1 : ORIGIN = 0x800000000, LENGTH =
0x180000000
    noc_ddr4_ddr_memory_DDR_LOW_0 : ORIGIN = 0x0, LENGTH = 0x80000000
    CIPS_0_pspmc_0_psv_ocm_ram_0_memory_0 : ORIGIN = 0xfffc0000, LENGTH =
0x40000
}")
set(STACK_SIZE 0xa000)
set(HEAP_SIZE 0xa000)
```

Sample lscript.ld.in File

```

_STACK_SIZE = DEFINED(_STACK_SIZE) ? _STACK_SIZE : @STACK_SIZE@;
_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : @HEAP_SIZE@;

_EL0_STACK_SIZE = DEFINED(_EL0_STACK_SIZE) ? _EL0_STACK_SIZE : 1024;
_EL1_STACK_SIZE = DEFINED(_EL1_STACK_SIZE) ? _EL1_STACK_SIZE : 2048;
_EL2_STACK_SIZE = DEFINED(_EL2_STACK_SIZE) ? _EL2_STACK_SIZE : 1024;

@MEMORY_SECTION@

/* Specify the default entry point to the program */

ENTRY(_vector_table)

/* Define the sections, and where they are mapped in memory */

SECTIONS
{
.text : {
   KEEP (*(.vectors))
   *(.boot)
   *(.text)
   *(.text.*)
   *(.gnu.linkonce.t.*)
   *(.plt)
   *(.gnu_warning)
   *(.gcc_execpt_table)
   *(.glue_7)
   *(.glue_7t)
   *(.ARM.extab)
   *(.gnu.linkonce.armextab.*)
} > @CODE@

.note.gnu.build-id : {
   KEEP (*(.note.gnu.build-id))
} > @CODE@


.init (ALIGN(64)) : {
   KEEP (*(.init))
} > @CODE@

.fini (ALIGN(64)) : {
   KEEP (*(.fini))
} > @CODE@

.interp : {
   KEEP (*(.interp))
} > @CODE@

.note-ABI-tag : {
   KEEP (*(.note-ABI-tag))
} > @CODE@

.rodata : {
   . = ALIGN(64);
   __rodata_start = .;
   *(.rodata)
   *(.rodata.*)
   *(.gnu.linkonce.r.*)
   __rodata_end = .;
} > @DATA@

.rodata1 : {
   . = ALIGN(64);
   __rodata1_start = .;
   *(.rodata1)
   *(.rodata1.*)
   __rodata1_end = .;
} > @DATA@

.sdata2 : {
   . = ALIGN(64);
   __sdata2_start = .;
   *(.sdata2)
   *(.sdata2.*)
   *(.gnu.linkonce.s2.*)
   __sdata2_end = .;
} > @DATA@

.sbss2 : {
   . = ALIGN(64);
   __sbss2_start = .;
   *(.sbss2)
   *(.sbss2.*)
   *(.gnu.linkonce.sb2.*)
   __sbss2_end = .;
} > @DATA@

.data : {
   . = ALIGN(64);
   __data_start = .;
```

```
    *(.data)
    *(.data.*)
    *(.gnu.linkonce.d.*)
    *(.jcr)
    *(.got)
    *(.got.plt)
    __data_end = .;
} > @DATA@

.data1 : {
    . = ALIGN(64);
    __data1_start = .;
    *(.data1)
    *(.data1.*)
    __data1_end = .;
} > @DATA@

.got : {
    *(.got)
} > @DATA@

.got1 : {
    *(.got1)
} > @DATA@

.got2 : {
    *(.got2)
} > @DATA@

.ctors : {
    . = ALIGN(64);
    __CTOR_LIST__ = .;
    ___CTORS_LIST___ = .;
    KEEP (*crtbegin.o(.ctors))
    KEEP (*(EXCLUDE_FILE(*crtend.o) .ctors))
    KEEP (*(SORT(.ctors.*)))
    KEEP (*(.ctors))
    __CTOR_END__ = .;
    ___CTORS_END___ = .;
} > @DATA@

.dtors : {
    . = ALIGN(64);
    __DTOR_LIST__ = .;
    ___DTORS_LIST___ = .;
    KEEP (*crtbegin.o(.dtors))
    KEEP (*(EXCLUDE_FILE(*crtend.o) .dtors))
    KEEP (*(SORT(.dtors.*)))
    KEEP (*(.dtors))
    __DTOR_END__ = .;
    ___DTORS_END___ = .;
} > @DATA@

.fixup : {
    __fixup_start = .;
    *(.fixup)
    __fixup_end = .;
} > @DATA@

.eh_frame : {
    *(.eh_frame)
} > @DATA@
```

```
.eh_framehdr : {
   __eh_framehdr_start = .;
   *(.eh_framehdr)
   __eh_framehdr_end = .;
} > @DATA@

.gcc_except_table : {
   *(.gcc_except_table)
} > @DATA@

.mmu_tbl0 (ALIGN(4096)) : {
   __mmu_tbl0_start = .;
   *(.mmu_tbl0)
   __mmu_tbl0_end = .;
} > @DATA@

.mmu_tbl1 (ALIGN(4096)) : {
   __mmu_tbl1_start = .;
   *(.mmu_tbl1)
   __mmu_tbl1_end = .;
} > @DATA@

.mmu_tbl2 (ALIGN(4096)) : {
   __mmu_tbl2_start = .;
   *(.mmu_tbl2)
   __mmu_tbl2_end = .;
} > @DATA@

.ARM.exidx : {
   __exidx_start = .;
   *(.ARM.exidx*)
   *(.gnu.linkonce.armexidix.*.*)
   __exidx_end = .;
} > @DATA@

.preinit_array : {
   . = ALIGN(64);
   __preinit_array_start = .;
   KEEP (*(SORT(.preinit_array.*)))
   KEEP (*(.preinit_array))
   __preinit_array_end = .;
} > @DATA@

.init_array : {
   . = ALIGN(64);
   __init_array_start = .;
   KEEP (*(SORT(.init_array.*)))
   KEEP (*(.init_array))
   __init_array_end = .;
} > @DATA@

.fini_array : {
   . = ALIGN(64);
   __fini_array_start = .;
   KEEP (*(SORT(.fini_array.*)))
   KEEP (*(.fini_array))
   __fini_array_end = .;
} > @DATA@

.drvcfg_sec : {
    . = ALIGN(8);
     __drvcfgsecdata_start = .;
    KEEP (*(.drvcfg_sec))
    __drvcfgsecdata_end = .;
    __drvcfgsecdata_size = __drvcfgsecdata_end - __drvcfgsecdata_start;
} > @DATA@

.ARM.attributes : {
   __ARM.attributes_start = .;
   *(.ARM.attributes)
   __ARM.attributes_end = .;
} > @DATA@

.sdata : {
   . = ALIGN(64);
   __sdata_start = .;
   *(.sdata)
   *(.sdata.*)
   *(.gnu.linkonce.s.*)
   __sdata_end = .;
} > @DATA@

.sbss (NOLOAD) : {
   . = ALIGN(64);
   __sbss_start = .;
   *(.sbss)
   *(.sbss.*)
   *(.gnu.linkonce.sb.*)
   . = ALIGN(64);
   __sbss_end = .;
} > @DATA@

.tdata : {
   . = ALIGN(64);
   __tdata_start = .;
   *(.tdata)
   *(.tdata.*)
   *(.gnu.linkonce.td.*)
   __tdata_end = .;
} > @DATA@

.tbss : {
   . = ALIGN(64);
   __tbss_start = .;
   *(.tbss)
   *(.tbss.*)
   *(.gnu.linkonce.tb.*)
   __tbss_end = .;
} > @DATA@

.bss (NOLOAD) : {
   . = ALIGN(64);
   __bss_start__ = .;
   *(.bss)
   *(.bss.*)
   *(.gnu.linkonce.b.*)
   *(COMMON)
   . = ALIGN(64);
   __bss_end__ = .;
} > @DATA@

_SDA_BASE_ = __sdata_start + ((__sbss_end - __sdata_start) / 2 );

_SDA2_BASE_ = __sdata2_start + ((__sbss2_end - __sdata2_start) / 2 );

/* Generate Stack and Heap definitions */
```

```
.heap (NOLOAD) : {
   . = ALIGN(64);
   _heap = .;
   HeapBase = .;
   _heap_start = .;
   . += _HEAP_SIZE;
   _heap_end = .;
   HeapLimit = .;
} > @DATA@

.stack (NOLOAD) : {
   . = ALIGN(64);
   _el3_stack_end = .;
   . += _STACK_SIZE;
   __el3_stack = .;
   _el2_stack_end = .;
   . += _EL2_STACK_SIZE;
   . = ALIGN(64);
   __el2_stack = .;
   _el1_stack_end = .;
   . += _EL1_STACK_SIZE;
   . = ALIGN(64);
   __el1_stack = .;
   _el0_stack_end = .;
   . += _EL0_STACK_SIZE;
   . = ALIGN(64);
   __el0_stack = .;
} > @DATA@


_end = .;
}
```

Appendix H

Conditional Settings in Library YAMLs

Introduction

Conditional entries are supported only for supported examples, supported processors, and depends sections in Library YAMLs.

Following are the Variables exported from the build flow to YAML: proc, platform, and variant.

Possible values of exported variables are:

- **proc:** psu_cortexa53, psu_cortexr5, psu_pmu, psv_cortexa72, psv_cortexr5, psv_pmc, psv_psm, psx_cortexa78, cortexa78, psx_cortexr52, cortexr52, psx_pmc, psx_psm, asu, ps7_cortexa9, microblaze, and microblaze_riscv.
- **platform:** microblaze, microblaze_riscv, ZynqMP, Zynq, Versal, VersalNet, and Versal_2VE_2VM.
- **variant:** spartanuplus, spartanuplusaes1, Versal_2VP, Versal_2VE_2VM_Small, and so on.

***Note*:** For a PS+Soft MB/MBV design, the platform is set to the PS family. Platform is set to microblaze or microblaze_riscv for only PL designs. Different MicroBlaze families can be distinguished based on the "variant" name.

Variables imported to the build flow from YAML after processing have the same names as their corresponding YAML sections: `depends`, `examples`, and `supported_procs`.

Steps to Add Conditions in the YAML File

This section describes how to add steps in the YAML file.

1. Under a valid section, create a dictionary entry named `condition`.
2. The `condition` should be defined as a Python code snippet, represented as a YAML string, which determines the section's return values based on the variables exported from the build flow: `platform`, `proc`, and `variant`.

3. The `condition` can be a multi-line python conditional code.
4. The python code must contain a variable with the YAML section name.
5. Ensure the condition:
   - Uses input parameters `proc`, `platform`, and `variant`.
   - Modifies the variable (having the same name as YAML section's name) to produce the desired output.
   - Is syntactically correct Python code.
   - Contains proper indentation (2 spaces per level, as per YAML standards).
6. Validate YAML Syntax using a YAML linter or validator to check for syntax errors.

Example

How to add conditions under "examples" section of a library YAML:

```
examples:
  xilpuf_enc_dec_data_example.c:
  xilpuf_example.c:
    - dependency_files:
      - xilpuf_example.h
  xilpuf_enc_dec_data_client_example.c:
  xilpuf_client_example.c:
    - dependency_files:
      - xilpuf_example.h
  xilpuf_versal_net_example.c:
    - dependency_files:
      - xilpuf_versal_net_example.h
  xilpuf_spartan_ultrascale_plus_example.c:
    - dependency_files:
      - xilpuf_spartan_ultrascale_plus_example.h
  xilpuf_ssit_client_example.c:
    - dependency_files:
      - xilpuf_example.h

  condition : |
    examples = []
    if platform in ["Versal", "VersalNet", "Versal_2VE_2VM"]:
      if proc in ["microblaze", "psv_cortexa72", "psv_cortexr5",
"psx_cortexa78", "psx_cortexr52", "cortexa78", "cortexr52"]:
        examples += ["xilpuf_enc_dec_data_example.c", "xilpuf_example.c",
"xilpuf_client_example.c",
                     "xilpuf_enc_dec_data_client_example.c"]
    if platform in ["VersalNet", "Versal_2VE_2VM"]:
      if proc in ["microblaze", "psx_cortexa78", "psx_cortexr52",
"cortexa78", "cortexr52"]:
        examples += ["xilpuf_versal_net_example.c"]
    if platform in ["Versal"]:
      if proc in ["microblaze", "psv_cortexa72", "psv_cortexr5"]:
        examples += ["xilpuf_ssit_client_example.c"]
    if "spartanuplus" in variant:
        examples += ["xilpuf_spartan_ultrascale_plus_example.c"]
```

How the Condition Works:

- `xilpuf_enc_dec_data_example.c`, `xilpuf_example.c`, `xilpuf_client_example.c`, `xilpuf_enc_dec_data_client_example.c`: These examples are valid for APU, RPU and Soft MB of all the Versal, Versal Net and Versal 2VE and 2VM devices. These are not supported for PMC, PSM, and ASU processors.
- `xilpuf_versal_net_example.c`: This example is valid for Versal Net and Versal 2VE and 2VM devices and are supported for APU, RPU and Soft MBs only.
- `xilpuf_ssit_client_example.c`: This example is valid only for Versal devices and are supported for APU, RPU and Soft MBs only.
- `xilpuf_spartan_ultrascale_plus_example.c`: This example is supported only for spartanuplus variant of MicroBlaze/MicroBlaze RISCV devices.

Other Examples can be referred from the following links:

https://github.com/Xilinx/embeddedsw/blob/xlnx_rel_v2025.2/lib/sw_services/xilmailbox/data/xilmailbox.yaml#L23

https://github.com/Xilinx/embeddedsw/blob/xlnx_rel_v2025.2/lib/sw_services/xilsem/data/xilsem.yaml#L18

Appendix I

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Note: For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

These documents provide supplemental material useful with this guide:

1. *Bootgen User Guide* (UG1283)
2. *Software Debugger Reference Guide* (UG1725)
3. *BSP and Libraries Document Collection* (UG643)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/03/2025 Version 2025.2	
Appendix H: Conditional Settings in Library YAMLs	Added a new Appendix.
Entire document	Changed the hyperlinks based on the release 2025.2.
06/11/2025 Version 2025.1	
Porting a Template Application	Support to have platform specific library configurations in a template applications.
Entire document	Added support for Versal 2VE_2VM devices. Migration of sdtgen from XSCT package to an independent tool.
Appendix G: Application: Hardware Dependency and Linker Script Generation	Linker section updates to handle code and data segments separately.
11/27/2024 Version 2024.2	
Porting a Driver	Updated Data Folder in Drivers.
Appendix E: Including a Driver Example in the Peripheral Test Application	Added a point in "Changes to the Data Folder of the Driver."
06/21/2024 Version 2024.1	
Porting a Driver	Updated Data Folder in Drivers.
Appendix A: Generating a System Device Tree Using SDTGen	Added link to GitHub repository.
Appendix C: Options Available in the "required" Section	Updated note for canonical syntax.
Appendix D: Sample YAML for a Driver	Updated YAML file.

Section	Revision Summary
11/10/2023 Version 2023.2	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright