Vitis RTL Kernel Design Flow(using tcl)

Overview

Generate IP→ Package RTL Kernel→ Build .xclbin File

IP Generation(gen_ip.tcl)

각 ip core에 대해 다음 과정을 거친다.

- create_ip → set_property → generate_target all
- generate_target all : 지정된 IP 코어에 대해 타겟 파일을 생성함

  e.g. fifo_generator

```
# ------------------------------------------------------------------------
# generate fifo_256×512 for bias loader
# ------------------------------------------------------------------------
create_ip -name fifo_generator \
          -vendor xilinx.com \
          -library ip \
          -version 13.2 \
          -module_name fifo_256×512 \
          -dir ./ip_generation
set_property -dict [list CONFIG.Component_Name {fifo_256×512} \
                   CONFIG.Input_Data_Width {256} \
                   CONFIG.Input_Depth {512} \
                   CONFIG.Output_Data_Width {256} \
                   CONFIG.Output_Depth {512} \
                   CONFIG.Valid_Flag {true} \
                   CONFIG.Data_Count_Width {9} \
                   CONFIG.Write_Data_Count_Width {9} \
                   CONFIG.Read_Data_Count_Width {9} \
                   CONFIG.Full_Threshold_Assert_Value {511} \
                   CONFIG.Full_Threshold_Negate_Value {510}] \
                   [get_ips fifo_256×512]

generate_target all [get_files ./ip_generation/fifo_256×512/fifo_256×512.xci]
```

  생성된 ip의 구조는 다음과 같다.

- 📁 fifo_256×512
  - 📁 doc
  - 📁 hdl //IP 코어의 HDL implementation 파일
    - 📄 fifo_256×512 _rd_*.v
  - 📁 sim // wrapper for simulation
    - 📄 fifo_256×512 .v

- 📁 synth // file for synthesis
  - 📄 fifo_256×512 .v
- 📄 fifo_256×512 _ooc.xdc // Out-of-Context (OOC)
- 📄 fifo_256×512 .xdc // 제약 조건 관련 Waiver(타이밍 예외, DRC 경고 무시..) 파일
- 📄 fifo_256×512 .veo // Verilog template for top module
- 📄 fifo_256×512 .vho // VHDL template
- 📄 **fifo_256×512 .xci // Xilinx Core Instance : IP core의 구성 정보가 포함된 메인 파일**
- 📄 fifo_256×512 .xml // metadata

> 💡 `uram.v` 와 `uram.veo` 의 차이:
>
> - `uram.v` : Vivado가 설계의 합성 및 구현 단계에서 사용하는 핵심 구현 파일로, IP의 내부 동작을 정의.
> - `uram.veo` : 설계자가 IP 코어를 상위 설계에서 간편하게 통합할 수 있도록 제공되는 인스턴스화 **template**.

Package RTL Kernel(package_kernel.tcl)

1. Create IP project and IP packaging project

```
# Create project NPU_3D
create_project NPU_3D ./NPU_3D -part [lindex $argv 0]

# Add design sources(axi_files) into project
set AXI_files {axi_file1 axi_file2 axi_file3}
add_files -norecurse $AXI_files

# Set NPU_3D_top as top module
set_property top NPU_3D_top [current_fileset]

# Update compile order of sources_1
update_compile_order -fileset sources_1

# Create IP package project
ipx::package_project -root_dir ./NPU_3D_ip -vendor xilinx.com -library user -taxonomy /UserIP -import_file
s -set_current true
```

> 💡 ipx란?
>
> **IPX**는 Xilinx의 IP 관리 도구로, Vivado에서 사용자 정의 IP를 생성, 수정, 패키징, 재사용할 수 있도록 지원하는 프레임워크

2. Inference clock, reset and associate with AXI ports

a. Inference clock and reset signals as AXI bus signals

- ap_clk와 ap_restn을 xilinx clock rtl 표준 인터페이스로 정의

  → RTL kernel에 clock이 하나면 생략 가능하지만, 여러 clock을 정의할 때는 각 clock마다 infer해야함

```
# ap_clk inference
ipx::infer_bus_interface ap_clk xilinx.com:signal:clock_rtl:1.0 [ipx::current_core]

# ap_restn inference
ipx::infer_bus_interface ap_aresetn xilinx.com:signal:reset_rtl:1.0 [ipx::current_core]
```

b. Associate AXI interface with ap_clk

- for each interface

```
# Association of interface s_axilite with ap_clk
ipx::associate_bus_interfaces \
-busif s_axilite     \
-clock ap_clk [ipx::current_core]
```

c. Associate ap_restn with ap_clk

```
# Association of ap_restn with ap_clk
ipx::associate_bus_interfaces -clock ap_clk -reset ap_aresetn [ipx::current_core]
```

3. Set the definition of AXI control slave registers

a. Add RTL kernel registers

b. Set RTL kernel registers property

```
# Register Argument for CTRL
ipx::add_register CTRL          [ipx::get_address_blocks reg0 -of_objects [ipx::get_memory_maps s_axilite -of
_objects [ipx::current_core]]]

# Property Setting of Register CTRL
set_property address_offset {0x000} [ipx::get_registers CTRL -of_objects [ipx::get_address_blocks reg0 -o
f_objects [ipx::get_memory_maps s_axilite -of_objects [ipx::current_core]]]]
set_property size           {32}    [ipx::get_registers CTRL -of_objects [ipx::get_address_blocks reg0 -of_objec
ts [ipx::get_memory_maps s_axilite -of_objects [ipx::current_core]]]]
```

4. Associate AXI master port to pointer argument and set data width

- 특정 레지스터를 AXI 포트와 연결하여, 해당 레지스터가 지정된 인터페이스를 통해 동작하도록 설정

  a. Define association between register and parameter

  b. Set paramter value as AXI port

  → associate register and AXI master port

```
# Add register paramter ASSOCIATED_BUSIF for register IFM_ADDR
ipx::add_register_parameter ASSOCIATED_BUSIF [ipx::get_registers IFM_ADDR -of_objects [ipx::get_ad
dress_blocks reg0 -of_objects [ipx::get_memory_maps s_axilite -of_objects [ipx::current_core]]]]
```

```
# Set ASSOCIATED_BUSIF as ddr_m00_axi
set_property value {ddr_m00_axi}          [ipx::get_register_parameters ASSOCIATED_BUSIF \
                        -of_objects [ipx::get_registers IFM_ADDR \
                        -of_objects [ipx::get_address_blocks reg0 \
                        -of_objects [ipx::get_memory_maps s_axilite \
                        -of_objects [ipx::current_core]]]]]
```

- AXI 인터페이스에 파라미터를 추가하고, 파라미터 값을 설정한다

5. Package the Vivado IP and generate the Vitis kernel file

a. Set required property for Vitis kernel

b. Package the Vivado project into Vivado IP

c. Generate the Vitis XO file

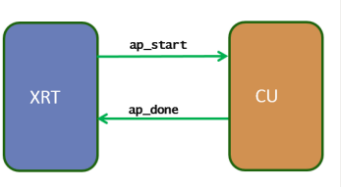
   → .xo file: Vivado 환경에서 디자인 된 RTL IP를 Vitis 환경에서 활용 가능

   → the tool generates the kernel description XML file automatically

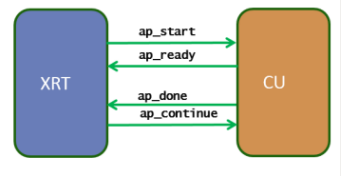
```
# Set required property for Vitis kernel
set_property sdx_kernel true [ipx::current_core]
set_property sdx_kernel_type rtl [ipx::current_core]

# Packaging Vivado IP
ipx::update_source_project_archive -component [ipx::current_core]
ipx::save_core [ipx::current_core]

# Generate Vitis XO(NPU_3D.xo) from Vivado IP
# kernel name: NPU_3D
# control protocol: ap_ctrl_hs
package_xo -force -xo_path ../NPU_3D.xo -kernel_name NPU_3D -ctrl_protocol ap_ctrl_hs -ip_directory ./N
PU_3D_ip -output_kernel_xml ../NPU_3D.xml
```

Kernel Control Protocols Options

1. ap_ctrl_hs
   - Handshake 신호 기반.
   - `ap_start`, `ap_done` 상태 신호를 포함.



2. ap_ctrl_chain:
   - input handshake: `ap_start`, `ap_ready`
   - output handshake: `ap_continue`, `ap_done`



3. ap_ctrl_none:
4. user_managed:

생성된 project의 구조는 다음과 같다.

- 📁 vivado_pack_krnl_project
  - 📁 NPU_3D // Main Project
    - 📁 NPU_3D.cache
    - 📁 NPU_3D.hw
    - 📁 NPU_3D.ip_user_files
    - 📄 NPU_3D.xpr
  - 📁 NPU_3D_ip // IP packaging project
    - 📁 src
    - 📁 xgui
- 📄 **NPU_3D.xo // output file → compile result of RTL**
- 📄 NPU_3D.xml // metadata file

Build xclbin file

v++

v++는 FPGA에서 실행되는 hardware kernel을 build, compile, link, 및 .xclbin 파일로 package.

v++는 이전의 **xocc** 명령어를 대체하며, 다음과 같은 작업을 수행한다:

- Kernel compile: HLS 또는 RTL로 작성된 kernel을 .xo(Xilinx Object) 파일로 변환.
- Kernel link: 여러 .xo 파일을 연결하고 최종 FPGA 바이너리 파일(.xclbin) 생성.
  →Generate executable file on the hardware or in an emulation environment
- Optimization and Debugging

```
# v++ [options] --config <.cfg> -o <.xclbin> <.xo>

# Platform: target platform(xilinx_u280_gen3×16_xdma_1_202211_1)
# Target: hw or hw_emu
XOCCFLAGS := --platform $(PLATFORM) -t $(TARGET)  -s -g
XOCCLFLAGS := --link --optimize 3

v++ $(XOCCLFLAGS) $(XOCCFLAGS) $(DEBUG_OPT) --config NPU_3D_krnl.cfg -o NPU_3D_krnl_$(TARGE
T).xclbin NPU_3D.xo
```

Reference

https://www.xilinx.com/content/dam/xilinx/publications/presentations/c_04_rtl_kernel_Vitis_Tutorial_webinar.pdf

https://xilinx.github.io/Vitis-Tutorials/2021-2/build/html/docs/Hardware_Acceleration/Design_Tutorials/05-bottom_up_rtl_kernel/doc/krnl_cbc.html