DRP-AI Translator V1.84

Release Note

Introduction

This release note describes the improvements of the DRP-AI Translator.

Key Features and Enhancements

- Update python packages(libraries) version(e.g. : numpy, etcs.,)
- Relax restrictions
  - Conv 7x7, stride=2, pad=3333
- Improve DRP Libraries
  - normalize, SpaceToDepth, Focus, transpose, resize(PreProcessing)

Contents

1. Improvements

1.1 Relax Restriction

The following operator is changed the restrictions. Please refer to Section4 of User's Manual for more details.

- Convolution 7x7 stride 2 pad 3333

1.2 Improve DRP Libraries

Improved inference speed for particular input/output shapes, or attributes setting.

- normalize
- SpaceToDepth
- Focus(Combination of Slice and Cocnat operators)
- transpose
- resize(PreProcessing)

2. Fixed Issues

2.1 Bug Fix

- Fixed the issue where DRP-AI object file was not generated correctly when certain combinations of height/width/input channel/output channel are used.
  - ➢ Conv 3x3, st1, pad =1111. Input width <= 5 and Input height >= 5
- Fix the issue in Translation Error Report function.

3. Known Issues

3.1 resize_hwc (pre-processing by DRP)

- The channel size must be smaller than 513.

Appendix 1. How to use python sample scripts

From DRP-AI Translator V1.80, python APIs and its sample scripts have been provided to improve development environment. The features are followings:

- Help to make input files for DRP-AI Translator
  Pre & Post definition and address map definition file can be generated by python APIs.

- Support to run simulation including Pre & Postprocessing by DRP
  It is possible to check the execution result before execution on the device.

The following two types of code are included as sample scripts in the DRP-AI Translator package.

1. Sample script to run DRP-AI Translator

    - Generate pre & post definition file(.yaml) & address map definition file (.yaml) by python script

    - Call DRP-AI Translator on python script

2. Sample script to run simulation including DRP pre & post processing

    - Simulate pre & post processing on python script

    - Simulate AI model inference by onnxruntime

[Note] These functions are beta version. The features and APIs subject to change. Also, the simulation does not guarantee a complete match with the results from the device.
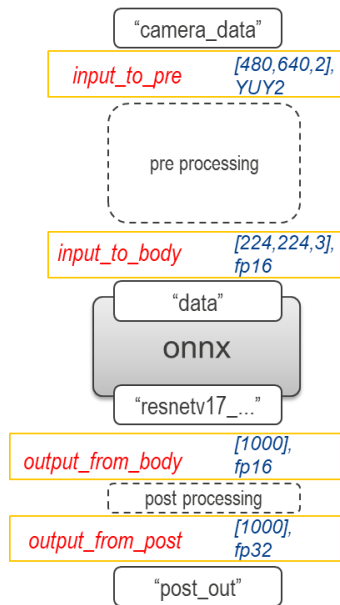

1. How to use running DRP-AI Translator script

- This section will explain the included sample code. The file name is below:

    <install dir>/UserConfig/sample_scripts/run_translator_sample_resnet50.py

- The steps to run the DRP-AI_Translator are as follows.

    A) Define and set pre & post nodes information to drp_prepost class

    B) Define and set pre & post processing classes to drp_prepost class

    C) Save pre & post definition file

    D) Initialize DRP-AI Translator class

    E) Make address map definition file & save

    F) Run DRP-AI Translator

- Before running sample script, please set environment variable(TRANSLATOR) to the directory where you installed DRP-AI Translator.

    e.g. $export TRANSLATOR=/home/user/drp-ai_translator_release/

A) Define and set pre & post nodes information to drp_prepost class

At first, define input & output node information about onnx model and pre & post processing

```python
pp = drp_prepost()
pp.set_input_to_body("data", shape=[224,224,3],
                     order="HWC",type="fp16",format="RGB")
pp.set_input_to_pre("camera_data", shape=[480,640,2],
                     order="HWC",type="uint8",format="YUY2")
pp.set_output_from_body("resnetv17_dense0_fwd",
                     shape=[1000],order="C",type="fp16")
pp.set_output_from_post("post_out",
                     shape=[1000],order="C",type="fp32")
```

"camera_data"

input_to_pre   [480,640,2], YUY2

pre processing

input_to_body   [224,224,3], fp16

"data"

onnx

"resnetv17_…"

output_from_body   [1000], fp16

post processing

output_from_post   [1000], fp32

"post_out"

B) Define and set pre & post processing classes to drp_prepost class
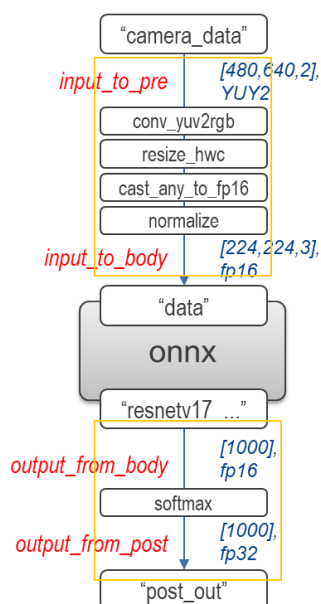
Next, initialize pre & post processing class, and get its instance object. Please refer section 4-2 in User's Manual about parameters of each process. To define sequence order of pre & post processing, make list data which stored instances. Then set pre & post processing sequence to drp_prepost class instance.

```
pre1_conv_yuv2rgb = conv_yuv2rgb(DOUT_RGB_FORMAT=0)
pre2_resize_hwc   = resize_hwc(shape_out=[224,224],RESIZE_ALG=1)
pre3_cast         = cast_any_to_fp16()
pre4_normalize    = normalize(cof_add=[-123.675, -116.28, -103.53],\
                        cof_mul= [0.01712475, 0.017507, 0.01742919])

post1_softmax     = softmax(DOUT_FORMAT=1)

pre_sequence  = [pre1_conv_yuv2rgb,pre2_resize_hwc,pre3_cast,pre4_normalize]
post_sequence = [post1_softmax]

pp.set_preprocess_sequence(src=["camera_data"],dest=["data"],pp_seq=pre_sequence)
pp.set_postprocess_sequence(src=["resnetv17_dense0_fwd"],dest=["post_out"],pp_seq=post_sequence)
```


"camera_data"

input_to_pre   [480,640,2], YUY2

conv_yuv2rgb

resize_hwc

cast_any_to_fp16

normalize

input_to_body   [224,224,3], fp16

"data"

onnx

"resnetv17 …"

output_from_body   [1000], fp16

softmax

output_from_post   [1000], fp32

"post_out"

C) Save pre & post definition file

To check setting of pre & post processing definition, show_param() function can be used. The result will show on the console. And by using save() function, pre & post processing definition file is saved.

```
# show setting info
pp.show_params()

# Save data as yaml file
pp.save("output.yaml")
```

```
# Initialize DRP-AI Transaltor class
drp_tran = drp_ai_translator()

# Choose device of run script
drp_tran.set_translator("V2M")

# Make address map file
drp_tran.make_addrfile("0x5F000000",addr_file="./addr_map.yaml")
```

D)   Initialize DRP-AI Translator class

To run DRP-AI Translator, drp_ai_translator class instance object is required. Please initialize and set the target device.

```
# Initialize DRP-AI Translator class
drp_tran = drp_ai_translator()

# Choose device of run script
drp_tran.set_translator("V2M")
```

E)   Make address map definition file & save

To make address map definition file, use make_addrfile() function. Please set the start address of the physical memory to map the DRP-AI Object files.

```
# Make address map file
drp_tran.make_addrfile("0x5F000000",addr_file="./addr_map.yaml")
# Check onnx model file
ONNX_MODEL="../../onnx/resnet50v1.onnx"
```

F)   Run DRP-AI Translator

By call run_translate() function, DRP-AI Translator will execute. The execution result is the same as using "./run_DRP-AI_translator_V2M/L.sh".

```
# Check onnx model file
ONNX_MODEL="../../onnx/resnet50v1.onnx"
drp_tran.run_translate("resnet_test",\
                    onnx=cur_path + "/" + ONNX_MODEL,\
                    prepost=cur_path+"/output.yaml",\
                    addr=cur_path+"/addr_map.yaml"
    )
```

2.   How to run simulation including DRP pre-post processing

- This section will explain the included sample code. The file name is below:

  <install dir>/UserConfig/sample_scripts/prepost_sim_sample_resnet50.py

- The steps for running a simulation including pre & post processing are as follows.

  A)   Define pre & post processing classes

  B)   Load input yuy2 file

  C)   Run pre processing simulation

RENESAS

D)   Run onnx simulation by onnxruntime

E)   Run post processing simulation

F)   Check the inference result

A)   Define pre & post processing classes

Generate instance objects of pre processing which is used for its simulation. Please refer section 4-2 in User's Manual about parameters of each process.

```
pre1_conv_yuv2rgb = conv_yuv2rgb(DOUT_RGB_FORMAT=0)
pre2_resize_hwc   = resize_hwc(shape_out=[224,224],RESIZE_ALG=1)
pre3_cast         = cast_any_to_fp16()
pre4_normalize    = normalize(cof_add=[-123.675, -116.28, -103.53],\
                            cof_mul= [0.01712475, 0.017507, 0.01742919])
post1_softmax     = softmax(DOUT_FORMAT=1)
```

B)   Load input yuy2 file

Read input data file for simulation.

```
yuy2_file = "./sample_input.bin"
yuy2_bin = np.fromfile(yuy2_file,dtype=np.uint8).reshape((480,640,2))
```

C)   Run pre processing simulation

Set the input data with numpy format for the pre processing instance object. The instance will return the simulation result with numpy format.

```
val1 = pre1_conv_yuv2rgb(yuy2_bin) # Pre1 : Conv_yuy2rgb
val2 = pre2_resize_hwc(val1)       # Pre2 : resize
pilImg = Image.fromarray(val2)     # <for debug> check resized data
pilImg.save('val2.png')
val3 = pre3_cast(val2)             # Pre3 : Cast
val4 = pre4_normalize(val3)        # Pre4 : Normalize
```

D)   Run onnx simulation by onnxruntime

Onnxruntime is used for simulation of onnx part. Since the data format of onnxruntime is CHW, it is necessary to run "transpose" processing.

```
_val4 = val4.transpose(2,0,1)
_val4 = _val4.reshape((1,_val4.shape[0],_val4.shape[1],_val4.shape[2]))
_val4 = _val4.astype(np.float32)
ONNX_MODEL="../../onnx/resnet50v1.onnx"
sess = rt.InferenceSession(ONNX_MODEL)
pred_onnx = sess.run(None, {"data": _val4}) # Run onnx runtime
output = pred_onnx[0] # get onnxruntime output
```

E)   Run post processing simulation

Set the input data with numpy format for the instance object of post processing. The instance will return the simulation result with numpy format.

```
output = output.reshape(1,1,1000)
val5 = post1_softmax(output) # Post1 : Softmax
```

F)   Check the inference result

Finally, check the inference result. This process varies by application. The sample code is for a classification task.

```
print(">> Inference result : ")
get_label(val5[0][0])
```

3. API reference

- Details of each API are described below. These APIs are beta versions and subject to change in the future.

CLASS : python_api.drp_prepost

drp_prepost. set_input_to_pre(node_name, shape=None, order=None, type=None, format=None)

Set information about input to preprocessing to drp_prepost class instance object.

Parameters :

node_name : str, input node name. e.g. "camera_in"

shape : list, data shape. e.g. [480,640,2]

order : str, data order. e.g. "HWC" / "CHW"

type : str, data type. e.g. "fp16" / "uint8"

format : str, data format. e.g. "RGB" / "YUY2"

Return :

None

drp_prepost. set_input_to_body(node_name, shape=None, order=None, type=None, format=None)

Set information about input to body to drp_prepost class instance object. "input to body" means input node name of onnx model.

Parameters :

node_name : str, input node name. e.g. "input"

shape : list, data shape. e.g. [224,224,3]

order : str, data order. e.g. "HWC"

type : str, data type. e.g. "fp16"

format : str, data format. e.g. "RGB"

Return :

None

drp_prepost. set_output_from_body(node_name, shape=None, order=None, type=None)

Set information about output from body to drp_prepost class instance object.

"output from body" means output node name of onnx model.

Parameters :

node_name : str, input node name. e.g. "output"

shape : list, data shape. e.g. [1000]

order : str, data order. e.g. "HWC" / "C"

type : str, data type. e.g. "fp16"

Return :

None

drp_prepost. set_output_from_post(node_name, shape=None, order=None, type=None, format=None)

Set information about output from post drp_prepost class instance object. "output from post" means output node of Postprocessing.

Parameters :

node_name : str, input node name. e.g. "post_out"

shape : list, data shape. e.g. [1000]

order : str, data order. e.g. "C"

type : str, data type. e.g. "fp32" / "fp16"

Return :

None

drp_prepost. set_preprocess_sequence(src=[], dest=[], pp_seq=[])

Set preprocess sequence to drp_prepost class instance object.

Parameters :

src : list, source node name of preprocess sequence. e.g. ["camera_in"]

dest : list, destination node name of preprocess sequence. shape. e.g. ["input"]

pp_seq : list, preprocess instance list. e.g. [conv_yuv2rgb, resize_hwc, cast, normalize]

Return :

None

drp_prepost. set_postprocess_sequence(src=[], dest=[], pp_seq=[])

Set postprocess sequence to drp_prepost class instance object.

Parameters :

src : list, source node name of postprocess sequence. e.g. ["output"]

dest : list, destination node name of postprocess sequence. shape. e.g. ["post_out"]

pp_seq : list, preprocess instance list. e.g. [softmax]

Return :

None

drp_prepost. show_param()

Show prepopst processing definition status to console.

Parameters :

None

Return :

None

drp_prepost. save(file_name)

Save prepost processing definition information as yaml file .

Parameters :

file_name : str, output file name. e.g. "output.yaml"

Return :

    None

CLASS : python_api.drp_ai_translator

drp_ai_translator.set_translator(device)

    Select target device.

Parameters :

    device : str, target device. e.g. "V2M" / "V2L"

Return :

    None

drp_ai_translator.make_addrfile(start_addr, addr_file=None)

    Save address map definition file

Parameters :

    start_addr: str, start address of the physical memory to map the DRP-AI Object files. e.g. "0x5F00000"

    addr_file : str, output yaml file name. e.g. "./addr_map.yaml"

Return :

    None

drp_ai_translator.run_translate(prefix, onnx=None, prepost=None, addr=None)

    Run DRP-AI Translator

Parameters :

    prefix : str, prefix is output folder name. e.g. "Resnet50_test"

    onnx : str, input onnx file path. e.g. "../onnx/resnet50.onnx"

    prepost : str, prepost processing definition file path. e.g. "./prepost.yaml"

    addr : str, address map definition file path. e.g. "./addrmap.yaml"

Return :

    None

CLASS : python_api. <Pre and Postprocessing>

drp_ai_translator.<Pre and Postprocessing>(**Parameters)

    Each Pre & Postprocessing has python class which is used as input data for drp_prepost class.

    And those class have a function to run simulation.

Parameters :

    Please refer section 5-3 in User's Manual.

Return :

    Pre/Postprocessing instance object or simulated output

After initialization, the instance object can be obtained as a return value.

When input data is set to the instance, it runs the simulation, and returns the execution results.

Supported Pre/Postprocessing :

transpose, conv_yuv2rgb, conv_x2gray, resize_hwc, cast_any_to_fp16, normalize, memcopy, crop,

softmax, cast_fp16_fp32, argminmax

Examples :

>> # Generate preprocessing instance objects

>> pre1_cast   = cast_any_to_fp16()

>> pre2_normalize   = normalize(cof_add=[-123.675, -116.28, -103.53], cof_mul= [0.01712475, 0.017507, 0.01742919])

>> # Run simulation. The input data must be numpy array.

>> pre1_out   = pre1_cast(input)

>> pre2_out   = pre2_normalize(pre1_out)