GUIA DE USUÁRIO FRAMEWORK HÍBRIDA

Carlos Gabriel de Araujo Gewehr

carlos.gewehr@ecomp.ufsm.br

1. Overview e Considerações Gerais:

Este documento descreve os procedimentos de operação do *framework* a uma perspectiva de usuário. Assume-se que o leitor tem conhecimento prévio a respeito do funcionamento da plataforma a nível de sistema. Será dado enfoque em como executar o fluxo completo através de um exemplo, com comentários específicos a cada passo da execução serão feitos à medida que estes são evidenciados ao curso da execução exemplo.

O *framework* requer a instalação de Python 3 na máquina onde se deseja usá-lo, e devem estar presentes as bibliotecas (já incluídas em grande parte das distribuições):

- argparse,
- copy,
- json,
- math,
- os,
- random,
- sys

Uma execução completa do fluxo envolve duas etapas: de Descrição e de Execução.

Na etapa de Descrição devem ser elaboradas as descrições de Topologias, Aplicações, Workloads (a partir das descrições de Aplicações), Mapas de Alocação (AllocationMaps) e Cluster Clocks, através da manipulação das classes dos módulos *AppComposer* (para descrições de Aplicações) e *PlatformComposer* (para descrições de Topologias).

Na etapa de Execução são gerados os arquivos de configuração da plataforma e injetores a serem lidos pelo HDL, implementando as descrições definidas na etapa anterior.

O exemplo a ser explorado é da topologia abaixo, onde está mapeado um *workload* consistente de 3 aplicações PIP em paralelo (estão incluídos junto aos arquivos fonte todos os arquivos de descrição usados neste guia):

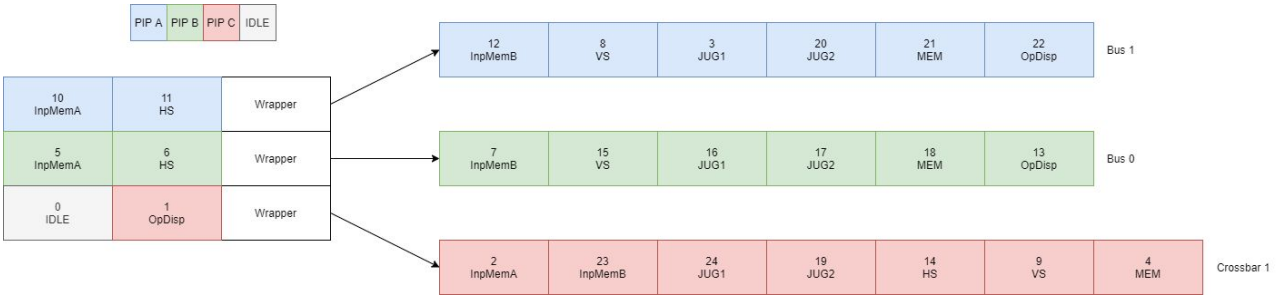

Figura 1 - Topologia de exemplo com workload *3PIP mapeado.*

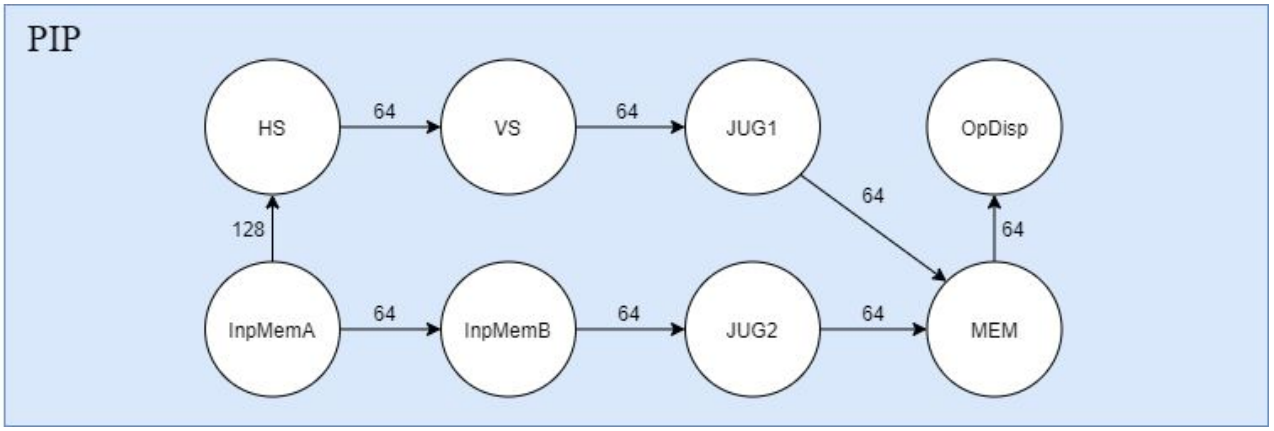

Figura 2 - Aplicação PIP caracterizada como grafo. Os vértices são threads *e arestas os volumes de comunicação entre* threads*, em MBps (Por exemplo, InpMemA envia 128 Megabytes/segundo para HS).*

2. Instalação

O primeiro passo é executar o *script* setup.py, localizado em "setup/setup.py". Este *script* gera dois arquivos que contém as definições de variáveis de ambiente necessárias para o *framework*, um arquivo com extensão ".source" para sistemas Linux e um arquivo com extensão ".bat" para sistemas Windows (Sistemas MacOS ou quaisquer outros não são compatíveis).

Deve se ressaltar que os comandos contidos nos arquivos gerados devem ser executados a cada nova janela do *shell* onde se deseja manipular o *framework*, através do comando "source" para sistemas Linux (*source <nomeDesejado>.source*), ou o comando "call" para sistemas Windows (*call <nomeDesejado>.bat*)[1].

O *script* requer os seguintes argumentos:

- *InstallName:* Nome que se deseja dar ao comando principal (através do qual são chamados os comandos secundários: *projgen*, *flowgen*, ...). Se não fornecido nenhum valor, será usado como *default* o nome "hibrida".
- *InstallPath:* Localização dos arquivos do *framework*. (Onde estão contidas as pastas: *doc, flowgenData, scripts, setup, src*). Se não fornecido nenhum valor, será usado como *default* o diretório pai do *script*.
- *DefaultProjDir*: Localização *default* para os projetos a serem criados pelo *framework*. Se não fornecido nenhum valor será usado como *default* o diretório "Desktop/HibridaProjects".

É possível também configurar individualmente os caminhos dos arquivos de descrição:

- *TopologiesPath*: Localização dos arquivos de descrição de topologias. Por *default*, "<InstallPath>/flowgenData/Topologies"
- *ApplicationsPath:* Localização dos arquivos de descrição de aplicações. Por *default*, "<InstallPath>/flowgenData/Applications"
- *WorkloadsPath:* Localização dos arquivos de descrição de *workloads*. Por *default*, "<InstallPath>/flowgenData/Workloads"

[1] Este processo pode ser automatizado através da inserção do comando "*source <InstallName>.source"* no arquivo .bash_aliases, localizado na home do usuário no qual se pretende utilizar o framework. Deste modo o comando é executado a cada nova janela de terminal aberta, eliminando a necessidade de fazê-lo manualmente.

- *AllocationMapsPath:* Localização dos arquivos de mapas de alocação (associam *threads* de uma aplicação a nós da rede). Por default, "<InstallPath>/flowgenData/AllocationMaps"
- *ClusterClocksPath:* Localização dos arquivos de frequências de *clock* de cada *cluster* da rede. Por default, "<InstallPath>/flowgenData/ClusterClocks"

Para uma instalação de nome "hibrida" e caminho de arquivos "/home/usr/ExUser/hibrida/" o *script* deve ser executado da forma:

python setup.py -InstallName hibrida -InstallPath /home/usr/ExUser/hibrida/

Serão criados no mesmo diretório do *script* os arquivos "hibrida.source" e "hibrida.bat" mencionados anteriormente.

Se desejado especificar individualmente os caminhos dos arquivos de descrição, isto deve ser feito da forma:

python setup.py -InstallName hibrida --WorkloadsPath /home/usr/ExUser/hibrida/Workloads

Também é possível executar o script com o flag de help, onde são exibidos todos os argumentos possíveis e uma mensagem de ajuda específica a cada argumento, da forma:

python setup.py -h

Que produz a seguinte mensagem no terminal:

```
PS D:\GitKraken\FrameworkHibridav2\setup> python setup.py -h
usage: setup.py [-h] [-InstallName INSTALLNAME] [-InstallPath INSTALLPATH] [-DefaultProjDir DEFAULTPROJDIR] [-top TOPOLOGIESPATH] [-app APPLICATIONSPATH] [-wor WORKLOADSPATH]
                [-alo ALLOCATIONMAPSPATH] [-clo CLUSTERCLOCKSPATH]

optional arguments:
  -h, --help            show this help message and exit
  -InstallName INSTALLNAME
                        Name of command to be called from shell (<InstallName> flowgen , <InstallName> compile, ...). If not given, "hibrida" is used as default
  -InstallPath INSTALLPATH
                        Path to framework source files. If not given, this scripts parent directory is used as default
  -DefaultProjDir DEFAULTPROJDIR
                        Directory where projects will be created at as default. If not given, $HOME will be used as default
  -top TOPOLOGIESPATH, --TopologiesPath TOPOLOGIESPATH
                        Default directory where Topology JSON files are stored. If not given, <InstallPath>/flowgenData/Topologies will be used as default
  -app APPLICATIONSPATH, --ApplicationsPath APPLICATIONSPATH
                        Default directory where Application JSON files are stored. If not given, <InstallPath>/flowgenData/Applications will be used as default
  -wor WORKLOADSPATH, --WorkloadsPath WORKLOADSPATH
                        Default directory where Workload JSON files are stored. If not given, <InstallPath>/flowgenData/Workloads will be used as default
  -alo ALLOCATIONMAPSPATH, --AllocationMapsPath ALLOCATIONMAPSPATH
                        Default directory where Allocation Maps JSON files are stored. If not given, <InstallPath>/flowgenData/AllocationMaps will be used as default
  -clo CLUSTERCLOCKSPATH, --ClusterClocksPath CLUSTERCLOCKSPATH
                        Default directory where Cluster Clocks JSON files are stored. If not given, <InstallPath>/flowgenData/ClusterClocks will be used as default
```

Figura 3 - Mensagem de ajuda do script *setup.py*

3. Fase de Descrições

a. Descrição de Topologias

A descrição de Topologias se dá a partir da manipulação das classes *Platform* (do módulo *PlaformComposer*), *Bus* e *Crossbar* (do módulo *Structures*), onde a partir de um objeto da classe *Platform* são inseridos objetos das classes *Bus* e *Crossbar* através do método *Platform.addStructure()*

Retomando o exemplo mencionado no Capítulo 1, a descrição da topologia da Figura 1 pode ser feita da forma:

```
1  import sys
2  import PlatformComposer
3
4  # Creates base 3x3 NoC
5  Setup = PlatformComposer.Platform(BaseNoCDimensions=(3, 3), ReferenceClock=100)
6
7  # Adds crossbar containing 7 PEs @ base NoC position (2, 0)
8  CrossbarA = PlatformComposer.Crossbar(AmountOfPEs = 7)
9  Setup.addStructure(NewStructure=CrossbarA, WrapperLocationInBaseNoC=(2, 0))
10
11 # Adds bus containing 6 PEs @ base NoC position (2, 1)
12 BusA = PlatformComposer.Bus(AmountOfPEs = 6)
13 Setup.addStructure(NewStructure=BusA, WrapperLocationInBaseNoC=(2, 1))
14
15 # Adds bus containing 6 PEs @ base NoC position (2, 2)
16 BusB = PlatformComposer.Bus(AmountOfPEs = 6)
17 Setup.addStructure(NewStructure=BusB, WrapperLocationInBaseNoC=(2, 2))
18
19 Setup.updatePEAddresses()
20
21 Setup.toJSON(SaveToFile = True, FileName = "TopologiaExemplo")
```

Figura 4 - Exemplo de Descrição de Topologia

Primeiramente é criado um objeto da classe *Platform*, com os argumentos *BaseNoCDimensions*, que define, respectivamente, as dimensões em X e Y da NoC base; e *ReferenceClock*, a frequência de *clock* (em MHz) a qual os injetores devem operar.

Neste objeto são então adicionados Buses/Crossbars (objetos *Bus*/*Crossbar*) através do método *addStructure()*, que requer os argumentos *NewStructure*, o objeto Bus/Crossbar a ser adicionado à plataforma; e *WrapperLocationInBaseNoC*, a localização desta estrutura na NoC base, em coordenadas XY.

É necessária a chamada do método *updatePEAddresses()* para "efetivar" as mudanças de topologia da plataforma, atualizando o endereço de cada PE conforme as alterações realizadas anteriormente.

Finalmente, chamando o método *toJSON()* são salvas as informações de topologia em um arquivo "TopologiaExemplo.json", que será utilizado posteriormente na etapa de Execução.

```
{
    "AmountOfPEs": 25,
    "AmountOfWrappers": 3,
    "BaseNoCDimensions": [
        3,
        3
    ],
    "ReferenceClock": 100,
    "SquareNoCBound": 5,
    "WrapperAddresses": [
        0,
        1,
        2,
        8,
        2,
        3,
        4,
        5,
        8,
        2,
        6,
        7,
        8,
        5,
        2,
        5,
        5,
        5,
        5,
        2,
        8,
        8,
        8,
        2,
        2
    ],
    "IsStandaloneBus": false,
    "AmountOfBuses": 2,
    "AmountOfPEsInBuses": [
        6,
        6
    ],
    "BusWrapperAddresses": [
        5,
        8
    ],
    "IsStandaloneCrossbar": false,
    "AmountOfCrossbars": 1,
    "AmountOfPEsInCrossbars": [
        7
    ],
    "CrossbarWrapperAddresses": [
        2
    ]
}
```

Figura 5 - Arquivo de formato JSON contendo as informações de topologia.

b. Descrição de Aplicações

Para as descrições de aplicações é necessário a manipulação das classes *Application*, *Thread* e *Flow*, inclusas no módulo *AppComposer*: Objetos da classe *Application* reúnem objetos da classe *Thread*, e objetos da classe *Flow* relacionam quantitativamente (volume de comunicação, em MBps) dois objetos da classe *Thread*.

Para a aplicação PIP do exemplo, é possível descrevê-la da forma:

```
import AppComposer

# Make Application
PIP = AppComposer.Application(AppName = "PIP")

# Make Threads
InpMemA = AppComposer.Thread(ThreadName = "InpMemA")
HS = AppComposer.Thread(ThreadName = "HS")
VS = AppComposer.Thread(ThreadName = "VS")
JUG1 = AppComposer.Thread(ThreadName = "JUG1")
InpMemB = AppComposer.Thread(ThreadName = "InpMemB")
JUG2 = AppComposer.Thread(ThreadName = "JUG2")
MEM = AppComposer.Thread(ThreadName = "MEM")
OpDisp = AppComposer.Thread(ThreadName = "OpDisp")

# Add Threads to applications
PIP.addThread(InpMemA)
PIP.addThread(HS)
PIP.addThread(VS)
PIP.addThread(JUG1)
PIP.addThread(InpMemB)
PIP.addThread(JUG2)
PIP.addThread(MEM)
PIP.addThread(OpDisp)

# Add Flows to Threads (Bandwidth parameter must be in Megabytes/second)
InpMemA.addFlow(AppComposer.Flow(TargetThread = HS, Bandwidth = 128))       # InpMemA -- 128 -> HS
InpMemA.addFlow(AppComposer.Flow(TargetThread = InpMemB, Bandwidth = 64))   # InpMemA -- 64 -> InpMemB
HS.addFlow(AppComposer.Flow(TargetThread = VS, Bandwidth = 64))             # HS -- 64 -> VS
VS.addFlow(AppComposer.Flow(TargetThread = JUG1, Bandwidth = 64))           # VS -- 64 -> JUG1
JUG1.addFlow(AppComposer.Flow(TargetThread = MEM, Bandwidth = 64))          # JUG1 -- 64 -> MEM
InpMemB.addFlow(AppComposer.Flow(TargetThread = JUG2, Bandwidth = 64))      # InpMemB -- 64 -> JUG2
JUG2.addFlow(AppComposer.Flow(TargetThread = MEM, Bandwidth = 64))          # JUG2 -- 64 -> MEM
MEM.addFlow(AppComposer.Flow(TargetThread = OpDisp, Bandwidth = 64))        # MEM -- 64 -> OpDisp

# Save App to JSON
PIP.toJSON(SaveToFile = True, FileName = "PIP")

```

Figura 6 - Exemplo de Descrição de Aplicação

A aplicação é descrita de modo *top-down*: Primeiro é criado o objeto da classe *Application* (maior nível hierárquico), e então, um objeto da classe *Thread* para cada Thread da aplicação (correspondentes aos vértices do grafo que caracteriza a aplicação). A esses objetos *Thread* são adicionados objetos *Flow* (correspondentes às arestas do grafo que caracteriza a aplicação).

De modo similar à descrição de Topologias, o objeto da classe *Application* é exportado em formato “json” no arquivo “PIP.json”, criado ao final da execução deste *script*.

c. Descrição de Workloads

Do mesmo modo que objetos *Application* reúnem objetos *Thread*, objetos *Workload* reúnem objetos *Application*. É possível tomar proveito disso reutilizando descrições de Aplicações previamente feitas em Workloads distintos (a mesma descrição de Aplicação pode ser utilizada em diferentes Workloads).

Usando a Aplicação PIP descrita na etapa anterior, um Workload "PIP_WL" consistente de 3 Aplicações PIP pode ser descrito da forma:

```
import os
import AppComposer

# Makes Workload object
PIP_WL = AppComposer.Workload(WorkloadName = "PIP_WL")

# Opens PIP App json file
with open(os.getenv("FLOWGEN_APPLICATIONS_PATH") + "/PIP.json") as PIP_JSON:

    # Builds 3 PIP Apps from json and adds them to PIP_WL Workload
    for i in range(3):

        PIP = AppComposer.Application(AppName = "PIP_" + str(i))
        PIP.fromJSON(PIP_JSON.read())
        PIP_WL.addApplication(PIP)
        PIP_JSON.seek(0)

# Exports Workload to json format
PIP_WL.toJSON(SaveToFile = True, FileName = "PIP_WL")
```

Figura 7 - Exemplo de Descrição de Workload

No *script* da Figura 7 é lida a descrição em formato "json" elaborada na etapa anterior da Aplicação PIP, e 3 cópias desta são adicionadas ao Workload PIP_WL. Este Workload é então exportado em formato "json" no arquivo "PIP_WL.json".

d. Descrição de Allocation Maps

A descrição de Allocation Maps é mais simples do que as descrições de Topologia, Aplicações e Workloads, pois não requer a utilização dos módulos *PlatformComposer* e *AppComposer*, é necessário apenas as funcionalidades fundamentais do Python.

O Allocation Map relaciona nós da rede com *threads*, da forma "*AllocationMap[PEPos] = App.Thread*". Para o exemplo sendo explorado:

```python
import json

# AllocMap[PEPos] = App.Thread
AllocArray = [None] * 25

AllocArray[0] = None
AllocArray[1] = "PIP_B.OpDisp"
AllocArray[2] = "PIP_B.InpMemA"
AllocArray[3] = "PIP.JUG1"
AllocArray[4] = "PIP_B.MEM"
AllocArray[5] = "PIP_A.InpMemA"
AllocArray[6] = "PIP_A.HS"
AllocArray[7] = "PIP_A.InpMemB"
AllocArray[8] = "PIP.VS"
AllocArray[9] = "PIP_B.VS"
AllocArray[10] = "PIP.InpMemA"
AllocArray[11] = "PIP.HS"
AllocArray[12] = "PIP.InpMemB"
AllocArray[13] = "PIP_A.OpDisp"
AllocArray[14] = "PIP_B.HS"
AllocArray[15] = "PIP_A.InpMemB"
AllocArray[16] = "PIP_A.JUG1"
AllocArray[17] = "PIP_A.JUG2"
AllocArray[18] = "PIP_A.MEM"
AllocArray[19] = "PIP_B.JUG2"
AllocArray[20] = "PIP.JUG2"
AllocArray[21] = "PIP.MEM"
AllocArray[22] = "PIP.OpDisp"
AllocArray[23] = "PIP_B.InpMemB"
AllocArray[24] = "PIP_B.JUG1"

AllocJSONString = json.dumps(AllocArray, sort_keys = False, indent = 4)

with open("ExampleAllocMap.json", "w") as JSONFile:
    JSONFile.write(AllocJSONString)
```

Figura 8 - Exemplo de Descrição de Allocation Map

e. Descrição de Cluster Clocks

A descrição de Cluster Clocks se refere ao período de *clock* de cada *cluster* (Bus/Crossbar individuais e cada roteador da NoC). Para o exemplo sendo explorado, há 9 clusters (cada posição da NoC base), 3 Bus/Crossbar e 6 Roteadores da NoC sem Bus/Crossbar associados. Para todos os clusters a uma frequência de *clock* de 100 MHz (período de 10 ns):

```python
import json

# ClusterClocks[BaseNoCPos] = Clock Period (in ns)
ClusterClocks = [None] * 9

ClusterClocks[0] = 10
ClusterClocks[1] = 10
ClusterClocks[2] = 10
ClusterClocks[3] = 10
ClusterClocks[4] = 10
ClusterClocks[5] = 10
ClusterClocks[6] = 10
ClusterClocks[7] = 10
ClusterClocks[8] = 10

ClocksJSONString = json.dumps(ClusterClocks, sort_keys = False, indent = 4)

with open("ExampleClusterClocks.json", "w") as JSONFile:
    JSONFile.write(ClocksJSONString)
```

Figura 9 - Exemplo de Descrição de Cluster Clocks

4. Fase de Execução

Na fase de Execução as descrições elaboradas na fase anterior são utilizadas para gerar os arquivos de configuração da plataforma e injetores, a serem lidos pelos arquivos VHDL, implementado a topologia e *workload* desejados.

Cada etapa da fase de Execução é implementada como um sub-comando do comando pai definido pelo script de instalação (Capítulo 2). Para o exemplo sendo explorado, o comando principal (do Capítulo 2, "hibrida") e os sub-comandos ("projgen", "flowgen", ...) são executados da forma ("hibrida projgen ..", "hibrida flowgen ...", …).

a. Comando projgen

Neste comando é criada a estrutura de diretórios necessária a um projeto do *framework*, onde estarão localizados os arquivos de configuração da plataforma ("*platform/*"); de injetores ("*flow/*"); logs dos injetores ("*log/*"); e arquivos de descrição, elaborados na etapa anterior ("*src_json*").

O comando espera apenas 2 argumentos: *ProjectDirectory*, diretório onde o novo projeto deverá ser criado (por *default*, usa o diretório *default* definido no arquivo ".source/.bat" gerado pelo *script* setup.py); e *ProjectName*, nome da pasta a ser criada em <*ProjectDirectory*> que contém os arquivos do projeto.

Deste modo será criado, se não existir, o diretório "<*ProjectDirectory*>/<*ProjectName*>/", e, dentro deste, os diretórios "*platform/*", "*flow/*", "*log/*" e "*src_json/*".

Por exemplo, retomando o comando principal definido como "hibrida", conforme Capítulo 2 e, se tomarmos <*ProjectDirectory*> como "*C:/HibridaProjects/*" e <*ProjectName*> como "*ExampleProject*", o comando *projgen* deve ser executado como:

hibrida projgen --ProjectDirectory C:/HibridaProjects --ProjectName ExampleProject

Que resulta em:

```
PS C:\HibridaProjects\ExampleProject> ls


    Directory: C:\HibridaProjects\ExampleProject


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        21-Aug-20  02:25 AM                flow
d-----        21-Aug-20  02:25 AM                log
d-----        21-Aug-20  02:25 AM                platform
d-----        21-Aug-20  02:25 AM                src_json
```

Figura 10 - Resultado da execução do comando projgen

Deve-se notar que a estrutura de diretórios não estará povoada por nenhum arquivo, isto será feito ao executar, na próxima etapa, o comando *flowgen*.

b. Comando flowgen

Neste comando são gerados os arquivos ".json" a serem lidos pelos arquivos de *hardware* (plataforma/injetores), implementando as descrições elaboradas tal como ao longo do Capítulo 3.

O comando *flowgen* requer um grande número de argumentos:

- *ProjectDir*: Diretório criado pelo comando *projgen* na etapa anterior;
- *TopologyFile*: Arquivo de descrição de Topologia;
- *WorkloadFile*: Arquivo de descrição de *Workload*;
- *AllocMapFile*: Arquivo de descrição de Allocation Map;
- *ClusterClocksFile*: Arquivo de descrição de *clocks* dos *clusters*.

Os arquivos de descrição devem passados como argumento ao comando sem a extensão ".json".

Além destes, também é possível individualmente definir os caminhos para os diretórios onde os arquivos de descrição passados como argumentos (acima) estão localizados (por *default*, estes caminhos serão tal como definido no arquivo .source/.bat conforme Capítulo 2):

- *TopologiesPath*: Diretório onde se encontra o arquivo <TopologyFile>.json;
- *WorkloadsPath*: Diretório onde se encontra o arquivo <WorkloadFile>.json;
- *AllocationMapsPath*: Diretório onde se encontra o arquivo <AllocMapFile>.json;
- *ClusterClocksPath*: Diretório onde se encontra o arquivo <ClusterClocksFile>.json.

Para o exemplo sendo explorado, com os arquivos de descrição tal como no Capitulo 3 e projeto criado anteriormente pelo comando *projgen*, o comando *flowgen* deve ser executado da forma:

hibrida flowgen --ProjectDir C:/HibridaProjects/ExampleProject --TopologyFile TopologiaExemplo --WorkloadFile PIP_WL --AllocMapFile ExampleAllocMap --ClusterClocksFile ExampleClusterClocks

Como não foram definidos explicitamente os caminhos para os arquivos de descrição, serão usados os caminhos *default* definidos no arquivo .source/.bat (estes podem ser definidos por *--TopologiesPath foo*, *--WorkloadsPath bar*, ...).

Ao executar o comando, o diretório "*<ProjectDir>/flow*" será povoado com os arquivos de configuração dos injetores, implementando o fluxo descrito pelo arquivo de descrição "*<WorkloadFile>.json*".

No diretório "*<ProjectDir>/platform*" estarão localizados o arquivo de configuração da plataforma "*PlatformConfig.json*", que define a topologia descrita por "*<TopologyFile>.json*" e o arquivo "*<ClusterClocks>.json*, que define os períodos de *clock* de cada *cluster* da plataforma.

Em "*<ProjectDir>/src_json*" estarão os arquivos de descrição *<TopologyFile>*, *<WorkloadFile>*, *<AllocMapFile>* e *<ClusterClocksFile>* passados como argumento ao comando *flowgen*, de modo a verificar mais facilmente os arquivos de descrição associados ao projeto em questão.

Finalmente, em "*<ProjectDir>/log*" se encontrarão os *logs* gerados pelos injetores quando o projeto for compilado e simulado.