system builder library

May 5, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

System builder is a python library that brings other build systems under one execution flow. The idea isn't to replace cmake, fusesoc, or other build systems. Its to glue them together in a single scripted process to generate a end result. This allows these parts to be built using systems that make sense for their end goals. Overall the goal is to make complete system generation for a Flash chip, SDCARD or other target easier and effortless for the end user.

The library has three seperate objects that help with this task. The first is a yaml parser that takes two yaml files, and commands and projects file, that are parsed and combined to create a series of dictionaries and nested lists. This data is then given to the executor which creates threads that lanuch the commands and monitors them for errors. The last, and unimplimented, is a dependency check object. This will check for all dependencies and report if they are installed and how to install them if they are not.

The executor will show each target project and have its own current status to show its own progress bar. This shows the time elapsed, percent complete, status, and name of current target being build. This can be disabled if needed. One note the progress bar up to the Target name is fixed to 80 columns. Anthing smaller then 80 will fail. Anything over 80 will show more of the target name. Meaning, if you have only 80 columns, the target name will be cut off on the screen.

1.2 Dependencies

The following are the dependencies of the cores.

- python 3.X
- python progressbar2

1.3 Example Creator

The creator is responsable for instantiating the library, preparing the project, and launching the system builder library to build the project.

```
from system.build import *

cmd_compiler = commandCompiler("projects.yml", "commands.
    ↪ yml")

cmd_compiler.create()
```

```
projects = cmd_compiler.getResult()

cmd_exec = commandExecutor(projects, args.dryrun)

cmd_exec.runProject()
```

1.4 Example Commands Yaml file

This yaml file is injested so commands are executed with certain steps for the final end product.

```
fusesoc:
  cmd_0: ["__CHECK_SKIP__{_pwd}/output/hdl/{_project_name
      ↪ }/AFRL_project_veronica_axi_baremetal_1.0.0.bit"]
  cmd_1: ["fusesoc", "--cores-root", "{path}", "run", "--
      ↪ build", "--work-root", "output/hdl/{_project_name
      ↪ }", "--target", "{target}", "{project}"]
script:
  cmd_1: ["{exec}", "{file}", "{_project_name}", "{args}"
      ↪ ]
gcc_riscv32:
  cmd_0: ["__CHECK_SKIP__{_pwd}/output/bin/riscv/bin/
      ↪ riscv32-unknown-elf-gcc"]
  cmd_1: ["make", "-C", "{_pwd}/{path}", "clean"]
  cmd_2: ["__CWD__{_pwd}/{path}", "./configure", "--
      ↪ prefix={_pwd}/output/bin/riscv", "--disable-linux
      ↪ ", "--with-arch=rv32imac", "--with-abi-ilp32"]
  cmd_3: ["make", "-C", "{_pwd}/{path}", "-j", "8"]
openocd_riscv:
  cmd_0: ["__CHECK_SKIP__{_pwd}/output/bin/openocd/bin/
      ↪ openocd"]
  #cmd_1: ["__CWD__{_pwd}/{path}", "git", "apply", "--
      ↪ ignore-whitespace", "../patch/pmpregs.patch"]
  cmd_2: ["__CWD__{_pwd}/{path}", "./bootstrap"]
  cmd_3: ["__CWD__{_pwd}/{path}", "./configure", "--
      ↪ prefix={_pwd}/output/bin/openocd", "--enable-ftdi
      ↪ ", "--enable-dummy", "--enable-jtag_vpi"]
  cmd_4: ["make", "-C", "{_pwd}/{path}", "-j", "8"]
  cmd_5: ["make", "install", "-C", "{_pwd}/{path}"]
fpga_baremetal_examples:
  cmd_0: ["__CHECK_SKIP__{_pwd}/output/apps"]
  cmd_1: ["mkdir", "-p", "{_pwd}/{path}/cmake"]
  cmd_2: ["__CWD__{_pwd}/{path}/cmake", "__ENV_PATH__{
      ↪ _pwd}/output/bin/riscv/bin/", "cmake", "../", "-
```

```
      ↪ DCMAKE_RUNTIME_OUTPUT_DIRECTORY={_pwd}/output/
      ↪ apps", "-DCMAKE_PREFIX_PATH={_pwd}/output/bin/
      ↪ riscv/bin/", "-DBUILD_EXAMPLES_ALL=ON", "-
      ↪ DCMAKE_TOOLCHAIN_FILE={_pwd}/{path}/arch/riscv/
      ↪ riscv.cmake"]
  cmd_3: ["__ENV_PATH__{_pwd}/output/bin/riscv/bin/", "
      ↪ make", "-C", "{_pwd}/{path}/cmake"]
buildroot:
  cmd_1: ["rm", "-rf", "{_pwd}/output/linux/{
      ↪ _project_name}"]
  cmd_2: ["make", "-C", "{path}", "distclean"]
  cmd_3: ["make", "O={_pwd}/output/linux/{_project_name}"
      ↪ , "-C", "{path}", "{config}"]
  cmd_4: ["make", "O={_pwd}/output/linux/{_project_name}"
      ↪ , "-C", "{path}"]
genimage:
  cmd_1: ["mkdir", "-p", "{_pwd}/output/genimage/tmp/{
      ↪ _project_name}"]
  cmd_2: ["genimage", "--config", "{path}/{_project_name
      ↪ }.cfg"]
```

1.5 Example Projects Yaml file

This takes the commands and fills out the information required and order of operations for creating the finaly end result project.

```
zed_fmcomms2-3_linux_busybox_sdcard:
  concurrent:
    fusesoc: &fusesoc_fmcomms2-3
      path: hdl
      project: AFRL:project:fmcomms2-3:1.0.0
      target: zed_bootgen
    buildroot: &buildroot_fmcomms2-3
      path: sw/linux/buildroot-afrl
      config: zynq_zed_ad_fmcomms2-3_defconfig
  sequential:
    script: &output_files_fmcomms2-3
      exec: python
      file: py/output_gen.py
      args: "--rootfs␣output/linux␣--bootfs␣output/hdl␣--
          ↪ dest␣output/sdcard"
    genimage:
      path: img_cfg
zc702_fmcomms2-3_linux_busybox_sdcard:
  concurrent:
```

```
    fusesoc:
      <<: *fusesoc_fmcomms2-3
      target: zc702_bootgen
    buildroot:
      <<: *buildroot_fmcomms2-3
      config: zynq_zc702_ad_fmcomms2-3_defconfig
  sequential:
    script:
      <<: *output_files_fmcomms2-3
    genimage:
      path: img_cfg
```

2 Code Documentation

Natural docs is used to generate documentation for this project. The next lists the following sections.

- **init** python init code
- **builder** system builder library.
- **creator** system builder example code.

__init__.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/08

INFORMATION

Brief

builder define for packages

License MIT

builder.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/08

INFORMATION

Brief

parse yaml file to execute build tools

commandCompiler

commandCompiler

Parse yaml files into commands for command executor

FUNCTIONS

__init__

```
def __init__(
self,
yaml_projects
 =
None,
```

```
 yaml_commands
  =
 None,
 target
  =
 None
)
```

Setup class

listCommands

```
def listCommands(
self
)
```

Print a list of all the commands available

listProjects

```
def listProjects(
self
)
```

Print a list of all the

clear

```
def clear(
self
)
```

Clear results of create and current yaml file pointer.

setProjects

```
def setProjects(
self,
yaml_projects
)
```

Set a yaml file for project commands

setCommands

```
def setCommands(
self,
yaml_commands
)
```

Set a yaml file for commands available

setProjectsWithYamlData

```
def setProjectsWithYamlData(
self,
yaml_data
)
```

Set internal variable with yaml data directly.

setCommandsWithYamlData

```
def setCommandsWithYamlData(
self,
yaml_data
)
```

Set internal variable with yaml data directly.

create

```
def create(
self,
yaml_projects
 =
None,
yaml_commands
 =
None,
target
 =
None
)
```

Pass a yaml file to use for processing into format for commandExecutor.

getResult

```
def getResult(
self
)
```

Return dict of dicts that contains lists with lists of lists to execute with subprocess {'project': { 'concurrent': [[["make", "def_config"], ["make"]], [["fusesoc", "run", "--build", "--target", "zed_blinky", "::blinky:1.0.0"]]], 'sequential': [[]]}}

_checkTarget

```
def _checkTarget(
self
)
```

_process

```
def _process(
self
)
```

Create dict of dicts that contains lists with lists of lists to execute with subprocess {'project': { 'concurrent': [[["make", "def_config"], ["make"]], [["fusesoc", "run", "--build", "--target", "zed_blinky", "::blinky:1.0.0"]]], 'sequential': [[]]}}

_load_yaml

```
def _load_yaml(
self,
yaml_file
)
```

Internal function that will load the yaml file that contains project commands or commands.

commandExecutor

commandExecutor

Execute commands create from commandCompileris not a valid selection

FUNCTIONS

clear

```
def clear(
self
)
```

Clear state of commandExecutor to init with no values passed

completed

```
def completed(
self
)
```

return a bool if all items have been completed.

failed

```
def failed(
self
)
```

return a bool indicating failure

stop

```
def stop(
self
)
```

Kill and current processes in build threads

setDryrun

```
def setDryrun(
dryrun
 =
False
)
```

Setting to True sets the project run everything up to the subprocess.Popen call, which is skipped.

setProjects

```
def setProjects(
self,
projects
)
```

Set the projects for the executor to build

runProject

```
def runProject(
self,
projects
 =
None
)
```

Call the internal execute method to start calling up each of the projects to build.

_execute

```
def _execute(
self
)
```

Call subprocess as a thread an_gen_build_cmdsd add it to a list of threads for wait to check on. iterate over projects avaiable and execute commands per project

_subprocess

```
def _subprocess(
self,
list_of_commands
```

```
)
```

Responsible for taking a list of commands and launching threads concurrently or singurely.

_project_cmd_count

```
def _project_cmd_count(
self,
run_types
)
```

Number of commands in a project.

_thread_exception

```
def _thread_exception(
self,
args
)
```

Used to kill all threads once one has failed.

_bar_thread

```
def _bar_thread(
self
)
```

Creates progress bar display in terminal for end user display.

creator.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/08

INFORMATION

Brief

Example program for using system.builder library.

@license MIT Copyright 2025 Jay Convertino

FUNCTIONS

main

```
def main()
```

main execution function

list_deps

```
def list_deps(
deps_file
)
```

open deps text file and print all executable names.

deps_check

```
def deps_check(
deps_file
)
```

Check each line of txt file programs

submodule_init

```
def submodule_init(
repo
)
```

Make sure submodules have been pulled. If not, pull them.

clean

```
def clean()
```

Clean up folders used for output (output and log)

parse_args

```
def parse_args(
argv
)
```

Parse args for tuning build

logger_setup

```
def logger_setup(
debug
)
```

Setup logger for log file