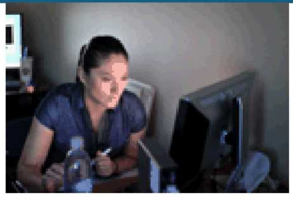
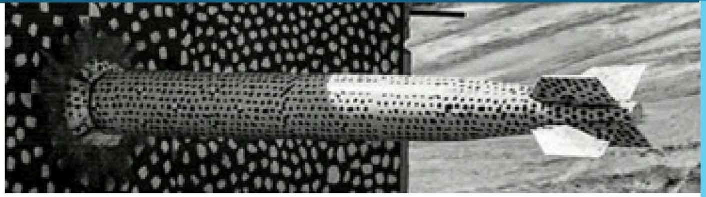
SAND2020-9765PE

Manipulating Jaqal with Jaqalpaq

Overview

Parsing Jaqal

Generating Jaqal

Programmatically creating IR

Manipulating IR

Parsing Jaqal

Use **parse_jaqal_string** or **parse_jaqal_file**

Found in **jaqalpaq/parser/parser.py**

Input is code as a Python (Unicode) string or text file.

Output is **Circuit** object (referred to as Intermediate Representation or IR in this presentation).

Parsing currently is slow for large files. We will address this in a future release.

Parameters

jaqal: The Jaqal code.

override_dict: An optional dictionary that overrides let statements in the Jaqal code. Note: all keys in this dictionary must exist as let statements or an error will be raised.

expand_macro: Replace macro invocations by their body while parsing.

expand_let: Replace let constants by their value while parsing.

expand_let_map: Replace let constants and mapped qubits while parsing. expand_let is ignored if this is True.

return_usepulses: Whether to both add a second return value and populate it with the usepulses statement.

inject_pulses: If given, use these pulses specifically.

autoload_pulses: Whether to employ the usepulses statement for parsing. Requires appropriate gate definitions.

Autoload Pulses

Looks up pulse files using standard Python module lookup

- I.e. Search through PYTHONPATH

Used in Emulator

Not used as part of JaqalPaw

See **qscout-gatemodels** project **qscout/v1/std/__init__.py** for the canonical example

- Package **std** contains **NATIVE_GATES** list of **GateDefinition** objects

Generating Jaqal

Use **generate_jaqal_program**

Found in **jaqalpaq/generator/generator.py**

Input is **Circuit** object.

Output is code as a Python (Unicode) string or text file.

Roughly the inverse of the parsing method.

Programmatically Creating IR

Object-Oriented API

- CircuitBuilder
- SequentialBlockBuilder
- ParallelBlockBuilder

“Undocumented” Lisp-like API

- Available Upon Request

Programmatic IR Creation Example

Jaqal

```
register q[2]
let angle0 0.123
let angle1 0.987
prepare_all
MS q[0] q[1] angle0 angle1
measure_all
```

IR Creation

```
gates = \
 qscout.v1.std.NATIVE_GATES
b = \
 CircuitBuilder(native_gates=gates)
q = b.register(“q”, 2)
b.let(“angle0”, 0.123)
b.let(“angle1”, 0.987)
b.gate(“prepare_all”)
b.gate(“MS”, q[0], q[1],
       “angle0”, “angle1”)
b.gate(“measure_all”)
b.build()
```

Circuit Class

Container for statements and metadata

- Let constants
- Macros
- Registers
  - Fundamental Registers
  - Map aliases
- Native Gates
- Body

Immutable – All changes must be done by creating a new Circuit

Found in **jaqalpaq/core/circuit.py**

Modifying Existing Circuit

Say we want to change

```
register q[2]
prepare_all
H q[0]
H q[1]
measure_all
```

to

```
register q[2]
loop 100 {
 prepare_all
 H q[0]
 H q[1]
 measure_all
}
```

```
c = \
parse_jaqal_file(“hadamard2.jaqal”)
reg = c.registers[“q”]
b = CircuitBuilder(native_gates=gates)
r = b.register(reg.name, reg.size)
b.loop(100, c.body)
b.build()
```