Sequence Store Generator

Autocoder User Guide

1 Description

The purpose of this generator is to provide a user friendly way of defining the memory slots for the sequence store component. The generator takes a YAML model file as input which specifies the location and size of each memory slot. Each memory slot is intended to hold a single sequence.

Note the example shown in this documentation is used in the unit test of this component so that the reader of this document can see it being used in context. Please refer to the unit test code for more details on how this generator can be used.

2 Schema

The following pykwalify schema is used to validate the input YAML model. The schema is commented to show what each of the available YAML keys are and what they accomplish. Even without knowing the specifics of pykwalify schemas, you should be able to gleam some knowledge from the file below.

```
---
# This schema describes the yaml format for a sequence store.
type: map
mapping:
  # Description of the sequence store.
  description:
    type: str
    required: False
  # List of slots to include in the store.
  slots:
    seq:
      - type: map
        mapping:
          # The address of the slot. This can be an address value or reference
          # a variable within another package.
          address:
            type: str
            required: True
          # The length of the slot in bytes. This can be an integer or reference
          # a variable within another package.
          length:
            type: str
            required: True
    # A  sequence store must have at least one slot.
    range:
      min: 1
    required: True
```

3 Example Input

The following is an example sequence store input yaml file. Model files must be named in the form *name.sequence_ store.yaml* where *name* is the name of the sequence store to be generated. Generally this file is created in the same directory or near to the assembly model file that includes the instance of the Sequence Store component that will be initialized by the autocoded sequence store. This example adheres to the schema shown in the previous section, and is commented to give clarification.

```yaml
---
# Optional - Description of sequence store
description: This is an example sequence store layout.
# Required - A list of slots that the store should manage. These
# slots should not overlap in anyway. Overlap will be checked for
# within the component Init.
slots:
  # Required - Address of first slot. This may be a numerical address
  # or a reference to a variable defined elsewhere. It is recommended
  # to use this model in conjunction with the memory_map model to
  # enforce no overlap.
  - address: Test_Slots.Slot_0_Memory'Address
  # Required - Length of first slot. This may be a numerical length
  # or a reference to a variable defined elsewhere. It is recommended
  # to use this model in conjunction with the memory_map model to
  # enforce no overlap.
    length: Test_Slots.Slot_0_Memory'Length
  - address: Test_Slots.Slot_1_Memory'Address
    length: Test_Slots.Slot_1_Memory'Length
  - address: Test_Slots.Slot_2_Memory'Address
    length: Test_Slots.Slot_2_Memory'Length
  - address: Test_Slots.Slot_3_Memory'Address
    length: Test_Slots.Slot_3_Memory'Length
```

As can be seen, specifying the sequence store layout simply consists of listing the addresses and lengths of each slot. The model file specifies start addresses and length for each slot. You can specify these values either as raw integers, or as names of variables defined elsewhere. Be careful to avoid overlapping any of the slot regions, as this will be checked within the component Init function at startup. It is recommended to declare sequence slots using the memory_map autocoder, which will ensure no overlap. You can then reference the variables produced in the memory_map autocoder within the sequencer store model.

4 Example Output

The example input shown in the previous section produces the following Ada output. The `slots_Access` variable should be passed into the Sequence Store component's `Init` procedure during assembly initialization.

The main job of the generator in this case was to verify the input YAML for validity and then to translate the data to an Ada data structure for use by the component.

```
------------------------------------------------------------------------------
-- This file was autogenerated from /vagrant/adamant/src/components/sequence_store/test/test_assembly/test_sequence_store.sequence_store.yaml on
-- 2022-04-01 19:37.
--
-- Copyright: The University of Colorado, Laboratory for Atmospheric and Space Physics (LASP)
------------------------------------------------------------------------------
```

```
6
7   -- Standard includes:
8   with Memory_Region;
9   with Sequence_Store_Types;
10  with Test_Slots;
11
12  -- This is an example sequence store layout.
13  package Test_Sequence_Store is
14
15    --
16    -- A list of the slots for the  component.
17    --
18    Slot_0 : constant Memory_Region.T := (
19      Address => Test_Slots.Slot_0_Memory'Address,
20      Length => Natural (Test_Slots.Slot_0_Memory'Length)
21    );
22
23    Slot_1 : constant Memory_Region.T := (
24      Address => Test_Slots.Slot_1_Memory'Address,
25      Length => Natural (Test_Slots.Slot_1_Memory'Length)
26    );
27
28    Slot_2 : constant Memory_Region.T := (
29      Address => Test_Slots.Slot_2_Memory'Address,
30      Length => Natural (Test_Slots.Slot_2_Memory'Length)
31    );
32
33    Slot_3 : constant Memory_Region.T := (
34      Address => Test_Slots.Slot_3_Memory'Address,
35      Length => Natural (Test_Slots.Slot_3_Memory'Length)
36    );
37
38    --
39    -- The slots type which is a list of all the slots:
40    --
41
42    Slots : aliased Sequence_Store_Types.Sequence_Slot_Array := (
43      0 => Slot_0,
44      1 => Slot_1,
45      2 => Slot_2,
46      3 => Slot_3
47    );
48
49    Slots_Access : constant Sequence_Store_Types.Sequence_Slot_Array_Access
50      := Slots'Access;
51
52  end Test_Sequence_Store;
```