MaxCompiler State Machine Cheat Sheet

v2 Interface

MAXELER Technologies

Value States

Type creation helper functions:

```
DFEsmValueType dfeBool()
DFEsmValueType dfeInt(int numBits)
DFEsmValueType dfeUInt(int numBits)
DFEsmValueType dfeValue(int numBits, DFEsmValueType.
  SignMode signMode)
```

Operators: cast, +, −, ∗, <, >, <=, >=, ===, !==, ˜, ˆ, &, |, >>, <<, <==, @, [], (+=, ∗=, >>= etc.)

Variable creation:

```
DFEsmStateValue state.value(DFEsmValueType type, boolean
  resetValue)
DFEsmStateValue state.value(DFEsmValueType type, long
  resetValue)
DFEsmStateValue state.value(DFEsmValueType type,
  BigInteger resetValue)
DFEsmStateValue state.value(DFEsmValueType type)
```

Warning: the initial (and reset) state of an unitialized state variable is *undefined*

Enumerated States

Java Type: DFEsmStateEnum<E extends Enum<E>>

Variable creation:

```
DFEsmStateEnum enumerated(Class<E> enumClass, E
  resetValue)
DFEsmStateEnum state.enumerated(Class<E> enumClass)
```

Example:

```
    enum States {
      STATE_1,
      STATE_2
    }
    DFEsmStateEnum<States> m_state;
```

Warning: the initial (and reset) state of an unitialized enumerated state variable is *undefined*

Intermediate Values

Note: creates no logic unless connected to a state

Variable: DFEsmValue

Operators: cast, +, −, ∗, <, >, <=, >=, ===, !==, ˜, ˆ, &, |, >>, <<, @, [], (+=, ∗=, >>= etc.)

Constant creation:

```
DFEsmValue constant.value(boolean value)
DFEsmValue constant.value(DFEsmValueType type, long value
  )
DFEsmValue constant.value(DFEsmValueType type, BigInteger
  value)
```

Intermediate Value Example 1:

```
    DFEsmStateValue a, b, c, d;
    ...
    DFEsmValue z = a+b;
    c <== z;
    d <== z;
```

Intermediate Value Example 2:

```
    DFEsmStateValue a, b, c;
    ...
    DFEsmValue DoAdd(DFEsmStateValue a,
             DFEsmStateValue b) {
        return a+b;
    }
    ...
    c <== DoAdd(a,b);
```

Inputs and Outputs

Stream i/o:

```
DFEsmInput io.input(String name, KernelType type)
DFEsmOutput io.output(String name, DFEsmValueType type)
DFEsmOutput io.output(String name, DFEsmValueType type,
  int latency)
```

e.g. `io.output("y", x, 3);`

Scalar i/o:

```
DFEsmInput io.scalarInput(String name, DFEsmValueType
  type)
DFEsmOutput io.scalarOutput(String name, DFEsmValueType
  type)
DFEsmOutput scalarOutput(String name, DFEsmValueType
  type, int latency)
```

Note: default output latency is 0

Memory

Common methods: getDepth, getLatency, getAddressWidth

Single-Port ROMs:

```
DFEsmSinglePortROM mem.rom(DFEsmValueType type, int...
  content)
DFEsmSinglePortROM mem.rom(DFEsmValueType type,
  Latency latency , int... contents)
DFEsmSinglePortROM mem.rom(DFEsmValueType type,
  Latency latency, long... contents)
DFEsmSinglePortROM mem.rom(DFEsmValueType type,
  Latency latency, List<BigInteger> contents)
DFEsmSinglePortMappedROM mem.romMapped(String name,
  DFEsmValueType type, int depth)
DFEsmSinglePortMappedROM mem.romMapped(String name,
  DFEsmValueType type, int depth, Latency latency)
```

Inputs: address

Outputs: dataOut

Dual-Port ROMs:

```
DFEsmDualPortROM mem.romDualPort(DFEsmValueType
  type, Latency latency, int... contents)
DFEsmDualPortROM mem.romDualPort(DFEsmValueType
  type, Latency latency, long... contents)
DFEsmDualPortROM mem.romDualPort(DFEsmValueType
  type, Latency latency, List<BigInteger> contents)
```

Inputs: addressA, addressB

Outputs: dataOutA, dataOutB

Single-port RAMs:

```
DFEsmSinglePortRAM mem.ram(DFEsmValueType type, int
  depth, SinglePortRAMMode portMode, Latency latency)
```

Inputs: address, dataIn, writeEnable

Outputs: dataOut

SinglePortRAMMode{READ_FIRST, WRITE_FIRST}

Dual-port RAMs:

```
DFEsmDualPortRAM mem.ramDualPort(DFEsmValueType
  type, int depth, DualPortRAMMode portModeA,
  DualPortRAMMode portModeB, Latency latency)
```

Inputs: addressA, addressB, dataInA, dataInB, writeEnableA, writeEnableB

Outputs: dataOutA, dataOutB

SinglePortRAMMode{READ_ONLY, RW_READ_FIRST, RW_WRITE_FIRST, WRITE_ONLY}

Latency:

Latency{ONE_CYCLE,TWO_CYCLES,THREE_CYCLES}

Note: default latency is Latency.ONE_CYCLE

Simple State Machine Example

```
public class SimpleSM extends KernelStateMachine {
  public enum States {
    COUNTING_UP,
    COUNTING_DOWN
  }
  // I/Os
  private final DFEsmInput m_max;
  private final DFEsmInput m_reverse;
  private final DFEsmOutput m_count;
  // State
  private final DFEsmStateValue m_counter;
  private final DFEsmStateEnum<States> m_state;
  public SimpleSM(KernelLib owner, int width) {
    super(owner);
    DFEsmValueType counterType = dfeUInt(width);
    // I/Os
    m_reverse = io.input("reverse", dfeBool());
    m_count = io.output("count", counterType);
    m_max = io.scalarInput("max", counterType);
    // State
    m_state = state.enumerated(States.class, States.
         COUNTING_UP);
    m_counter = state.value(counterType, 0);
  }
  @Override
  public void nextState() {
    SWITCH(m_state) {
      CASE(States.COUNTING_UP) {
        IF(m_reverse === 1)
          m_state.next <== States.COUNTING_DOWN;
        IF(m_counter === m_max) {
          m_counter.next <== m_counter - 1;
          m_state.next <== States.COUNTING_DOWN;
        } ELSE
          m_counter.next <== m_counter + 1;
      } OTHERWISE {
        IF(m_reverse === 1)
          m_state.next <== States.COUNTING_UP;
        IF(m_counter === 0) {
          m_counter.next <== m_counter + 1;
          m_state.next <== States.COUNTING_UP;
        } ELSE
          m_counter.next <== m_counter - 1;
      }
    }
  }

  @Override
  public void outputFunction() {
    m_count <== m_counter;
  }
}
```

Kernel Integration Example

```
public class SimpleKernel extends Kernel {
  public SimpleKernel(KernelParameters parameters) {
    super(parameters);
  ...
    SMIO MySimpleSM = addStateMachine("MySimpleSM",
         new SimpleSM(this,8));
    MySimpleSM.connectInput("reverse", reverse === 1);
    DFEVar count = MySimpleSM.getOutput("count");
  ...
  }
}
```

Debugging

```
debug.simPrintf(message, args)
debug.simPrintf(stream_name, message, args)
```

%s can be used to print DFEsmStateEnum's as strings.

Version 2014.1 © Maxeler Technologies