Object Oriented Design of Digital Circuits

Michael Jones
INFOTECH Master Student

Advisor: Prof. Martin Radetzki
Seminar: Architectures and Design Methods for Embedded Systems
OO Design of Digital Circuits

1. Introduction - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead…
6. Summary
Why OO in hardware?

- To manage complexity
  - The “Design Gap”: advances in production techniques are increasing faster than advances in design capabilities
    - The transistor count in modern SoC’s is increasing at a rate faster than the productivity of designers.
  - Bridge gap from specification and modeling to implementation
  - Efficient design => Take better advantage of high transistor count
  - Faster time-to-market => compete in a tight market
  - Gate => RTL => System level abstraction
OK, but why OO?

- Designing at the RT level doesn’t allow for the necessary level of abstraction.

- OO has a successful history in software development of enabling the design of complex systems using
  - Objects for reuse
  - Abstraction
  - Data encapsulation
  - Inheritance
  - Polymorphism
OO Design of Digital Circuits

1. Intro - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead…
6. Summary
Components of OO

- **Classes**: define a new data type and methods for accessing the type.
- **Objects**
  - maintain a defined state.
  - State space is defined by class definition.

```plaintext
FIFO
# m_buffer[Size]: Type
# m_empty: bool
# m_full: bool
# m_top: int
# m_bottom: int
# m_numberOfEntries: int
# nextIndex(index: int): int
+ FIFO(): FIFO
+ Reset()
+ Get(): Type
+ Write(t: Type)
```
Components of OO

- Attributes: data associated with objects.
  - Cross product of available attributes comprise the object’s state space

```c
FIFO
# m_buffer[Size] : Type
# m_empty : bool
# m_full : bool
# m_top : int
# m_bottom : int
# m_numberOfEntries : int
# nextIndex(index : int) : int
+ FIFO() : FIFO
+ Reset()
+ Get() : Type
+ Write(t : Type)
```
Components of OO

- Methods: provide indirect access to object’s state.

```plaintext
FIFO
# m_buffer[Size] : Type
# m_empty : bool
# m_full : bool
# m_top : int
# m_bottom : int
# m_numberOfEntries : int
# nextIndex(index : int) : int
+ FIFO() : FIFO
+ Reset()
+ Get() : Type
+ Write(t : Type)
```
Components of OO

- Inheritance
- Polymorphism
  - Determine at run-time the actual class to which an object belongs
  - Able to change the object’s class at run-time

Buffer
+ put(value : short)
+ get(value : short)

FIFO
- first : int = 0
- next : int = 0
+ put(value : short)
+ get(value : short)

LIFO
- index : short = 0
+ put(value : short)
+ get(value : short)
OO Design of Digital Circuits

1. Intro - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead
6. Summary
Fundamental difference between SW and HW.

- Software is dynamic and (generally) executed sequentially.
- Hardware is static and extremely parallel.
Odette System Synthesis Subset (OSSS)

- Product of the ODETTE (Object-oriented co-Design and functional Test Techniques) project.
- Goal of ODETTE- “to develop object-oriented hardware design methodology”
- Based on C++/SystemC
  - Retain many advantages of SystemC
- Restrictions on SystemC to enable synthesis
- Intended to provide
  - a synthesizable language which is also suitable for modeling.
  - the important object-oriented features of C++ for use in hardware design.
OSSS restrictions

- No dynamically allocated memory (new and delete) – can’t create/destroy hardware components
- No pointers – can’t synthesize
- No destructors
- No static non-const data members
- Polymorphic objects declared as such
OOHWLib

- C++ class library
- Enables OO HW components
  - Class templates for
    - Global object instantiation
    - Polymorphic objects
  - Macros for declaring guarded methods
  - Scheduler classes
OSSS Synthesis and Simulation

(from Grimpe, et. al.)
FIFO Code example

```cpp
#include "OOHWLib.h"
#define Size_t unsigned int

template<class Type, Size_t Size>
class FIFO
{

    public:

    // Default constructor. Creates an empty FIFO queue.
    FIFO();

    // If the queue is not empty, read the top of the queue and remove it.
    GUARDED_METHOD( virtual const Type, Get(), !isEmpty());

    // Checks if the queue is empty
    GUARDED_METHOD( virtual bool, isEmpty() const, true );

protected:
    Size_t nextIndex( const Size_t index );

    Type m_buffer[Size]; // Buffer for the entries.
    bool m_empty; // True iff buffer empty, false otherwise.
    bool m_full; // True iff buffer is full, false otherwise.
    Size_t m_top; // Points to the top of the FIFO.
    Size_t m_bottom; // Points to the bottom of the FIFO.
    Size_t m_numberOfEntries;

private:
};
```

Consideration for OO for Hardware
OO Design of Digital Circuits

1. Intro - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead
6. Summary
Synthesis of objects

We need to be able to synthesize:
- Attributes
- Methods
- Mechanism for inheritance
- Mechanism for polymorphism
- Communication
  - Connectivity among objects
  - Mutual exclusion
  - Fairness
- Object viewed as an FSM
  - Attributes define state space
  - Methods define state transitions
  - But
    - State space too large
    - Methods not combinational
Attributes

- Maintain the state of the object
- can be encoded into a bit vector

(from Grimpe, et al.)
Methods

(from Radetzki)
Inheritance

- Delegation: instantiate parent class, then extend
  - Simpler synthesis
  - Inefficient hardware

- Expansion: incorporate derived with parent class at compile time

(from Radetzki)
Polymorphism

- In C++, achieved with pointers.
- To synthesize, use “polymorphic objects”
  - Capacity to hold state of any potential object
  - Restricted set of potential objects
- Add tag to attributes
- Dynamic binding

```cpp
template<class Type, Size_t Size>
class FIFO
{
public:
    POLYMORPHIC(FIFO)
...
```
Communication

- Channels of communication created between client/server pairs
- If multiple clients, arbiter is instantiated

(from Grimpe, et al.)
OO Design of Digital Circuits

1. Intro - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead
6. Summary
Expectations

- The Design Gap continues to grow
- Search for new approach to HW design
  - Higher abstraction- system level language
    - Supports modeling and specification
    - Broad acceptance
  - IP reuse
  - Verification
Can OO meet these challenges?

- Addresses important problems
- Applying a software solution to a hardware problem
- Overhead to synthesize HW objects
  - Arbiter
  - Global objects
  - Communication
OO Design of Digital Circuits

1. Intro - motivation
2. Components of OO
3. Considering OO for hardware
4. Synthesis of objects in hardware
5. Looking ahead
6. Summary
Danke für Ihre Aufmerksamkeit.

Gibt es noch Fragen?
References

