Register Transfer Level (RTL) Design

RTL,
High-Level State Machines,
Design Process,
Design Considerations,
Multiple Processors
High Level Sequential Behavior

- Finite state machines can be used to capture simple sequential behavior using bit inputs.
- High level state machines can be used to capture more complex logic involving multi-bit variables.
Digital design is generally broken into different levels of abstraction.

- **Transistor Level**
  - Designing digital circuit with transistors directly
  - Difficult and cumbersome

- **Gate Level**
  - Design style we have studied so far
  - Build circuits out of gates

- **Register-Transfer Level**
  - Building circuits using registers, datapath components and controllers
  - Circuits are designed to control the transfer of data between registers through datapath components.

- **Transaction Level Modeling**
  - Abstracts communication mechanisms
  - We won’t discuss further
Processors

- ‘Processor’ is a generic term for a circuit designed using RTL principles
  - Programmable Processor
    - A generic processor designed that can run programs
    - I.E. Intel Processors
  - Custom Processor
    - Specialized design that implements specific functionality
    - I.E. A circuit to process digital TV signals.

- Processors can be designed using high level state machines
High Level State Machines (HLSM) extend FSMs with features that make it possible to capture more complex behaviors.

- Multi-bit data inputs and outputs
  - Assumed unsigned unless specified as signed
- Local storage
  - Registers loaded on rising clock edges (i.e. when leaving a state)
- Arithmetic operations
  - Add, Multiply, Compare, Bit Shift, etc.
HLSM Conventions

- HLSMs we discuss will follow these conventions:
  - All inputs, outputs and local storage are defined at the top of the HLSM diagram
    - Bit lengths are included in this definition
    - Registered outputs must be indicated as such.
      - The book assumes all outputs are registered, but we will explicitly specify.
    - Local storage values are always registered.
  - Registered values change on rising clock edges (i.e. when leaving a state)
  - Transition bits are implicitly ANDed with rising clock edge
  - Any unregistered output not explicitly assigned is 0.
  - Any registered value not explicitly assigned holds its value.
HLSM Conventions (continued)

- HLSMs we discuss will follow these conventions (continued):
  - Bits are designated by surrounding them with single quotes, integers have no quotes
    - i.e. ‘1’ is a bit value, 1 is an integer
  
  - “:=” assigns a value to a variable, “==” compares two values.
    - “=” is not used for anything.
  
  - “//” defines a comment, just as in C++.
HLSM Example:
Soda Dispenser Processor

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // ‘1’ dispenses soda
Local Storage: tot (8 bits)

c := '1'
tot := tot + a

c' • (tot < s)
tot := 0

c' • (tot >= s)

d := ‘1’
A Word of Warning About Registered Outputs

- Clocked storage items are not updated in the same clock cycle that their control signals are set
  - Must wait for a rising clock edge for the register to obtain a new value.

- Note: The register control bits set up in a state **prepare** the value on the register inputs to be captured on the **next clock cycle**!

- Example: In the previous example, when is the value of tot reset to 0?
HLSM Example: 8-bit Up/Down Counter

- Design an HLSM for an 8-bit up/down counter
  - What are the inputs?
  - What are the outputs?
  - What are the internal variables (if any)?
  - Draw the HLSM
Multi-bit Values in Timing Diagrams

- Multi-bit values in timing diagrams can be reduced to a single line representation
  - Value written between an upper & lower line
  - Lines “switch” when value changes.
Standard Processor Architecture

- The standard processor aids us in developing RTL based circuits just as the standard controller architecture aided FSM designs.

- Composed of
  - Controller
  - Datapath
RTL Design Process

1. Capture a HLSM
   - Create a HLSM diagram to describe the system’s intended behavior.

2. Convert to a Circuit
   1. Create a datapath
      - Create a datapath to carry out the data operations of the HLSM.
      - Use components from a library
      - Include registered outputs.
   2. Connect the datapath to a controller
      - Connect all control signals to the circuit
   3. Derive the controller’s FSM.
      - Convert the HLSM to a FSM for the controller
         - Replace data operations with setting and reading of control signals to and from the datapath.
         - Create a circuit for the controller from the FSM
RTL Design Process Example: Threadmill Speed Controller

- Design a system to control the speed of the conveyor belt on a treadmill
  - Speed is a 4 bit value that is controlled by two buttons
    - Up button increases speed by one
    - Down button decreases speed by one
    - If both are pushed, no change in speed occurs.
  - Speed must initialize to zero upon startup
RTL Design Process Example: Threadmill Speed Controller

1. Capture a HLSM

   □ Create a HLSM diagram to describe the system’s intended behavior.

Inputs: up (bit), down (bit)
Outputs: speed (4 bit reg)
Internal Storage: n/a
RTL Design Process Example: Threadmill Speed Controller

2. Convert to a Circuit
   1. Create a data path
      - Create a data path to carry out the data operations of the HLSM.
      - Use components from a library
      - Include registered outputs.

```
load
rst
clk

D[3:0] Q[3:0]
Load
Rst QN[7:0]

D_Reg_PL_Rst_4bit

a[3:0] S[3:0]
Incr-Decr_4bit

dir
co

Speed
```
RTL Design Process Example: Threadmill Speed Controller

2. Convert to a Circuit
   2. Connect the datapath to a controller
      - Connect all control signals to the circuit
RTL Design Process Example: Threadmill Speed Controller

2. Convert to a Circuit
3. Derive the controller’s FSM.
   - Convert the HLSM to a FSM for the controller
     - Replace data operations with setting and reading of control signals to and from the datapath.
   - Create a circuit for the controller from the FSM
Arrays in an HLSM

- An **array** is an ordered list of items.
- An array can be used in an HLSM just as it is in C
  - Use the array[index] notation.
  - The first index in an array is 0.
- An array in a HLSM corresponds to a register file or memory in a real circuit
RTL with an Array Example: Minimal Value Chooser

- Design a circuit that returns the index to the highest of 4 values stored in a 4x8-bit array when the input eval is set to 1.
  - eval starts the operation, but does not need to remain high for the duration of the operation.
  - Only one array element may be read at a time.
  - This operation may take multiple cycles
  - Set the output fin to 1 for one cycle when the operation is complete.
RTL with an Array Example: Minimal Value Chooser

1. Capture a HLSM

Create a HLSM diagram to describe the system’s intended behavior.

Inputs: eval (bit), a[4] (8 bit reg)
Outputs: fin (bit), imax (2 bit reg)
Internal Storage: i (2 bit reg)
RTL with an Array Example: Minimal Value Chooser

2. Convert to a Circuit
   1. Create a datapath
      - Create a datapath to carry out the data operations of the HLSM.
      - Use components from a library
      - Include registered outputs.

![Diagram showing an array example with components labeled as:
- Incrementer
- Data comparator
- Index comparator
- Registers for data, max, init, load, etc.
]
RTL with an Array Example: Minimal Value Chooser

2. Convert to a Circuit
   2. Connect the datapath to a controller
      - Connect all control signals to the circuit
2. Convert to a Circuit

3. Derive the controller’s FSM.
   - Convert the HLSM to a FSM for the controller
     - Replace data operations with setting and reading of control signals to and from the data path.
   - Create a circuit for the controller from the FSM
RTL with an Array Example: Minimal Value Chooser

- Our circuit integrated with an external register file.
Multiple Processors

- Complex designs can be difficult to model with a single HLSM.
- A much simpler design methodology is to break the complex design up into multiple smaller processors.
- Global components can be shared between processors to facilitate this breakdown:
  - Signals
  - Data path components
  - Etc.
Multi-Processor Examples

Example 1:
- Button debouncer processor used to debounce mechanical button inputs to another processor.

Example 2:
- Reconsider the speed controller. How could it be redesigned to allow two modes of operation: manual control and oscillating?