Introduction to Register-Transfer Level (RTL) Design: from Generalized ASM's to a Microarchitecture [PART I]

Generalized ASM's: combine both **datapath operations** and **control** in a single flow-chart-like specification.

(See D. Gajski, *Principles of Digital Design*, Prentice Hall, 1997, for more details.)

Example #1: Specify and Design a "1's Counter"

[More material on this example, using a somewhat different approach, is in Gajski, ch. 8.3.]

Step #0. Verbal Description of Algorithm:

Given a 32-bit binary number placed on "*Input*" bus:
- count the # of 1 bits in this number
- place result on "*Output*" bus

Goal: design both control (FSM) and datapath (select and connect datapath blocks)
= ***"microarchitecture"***

Assume:
- global input: *Start*
  → wait for *Start =1* before running algorithm
- global output: *Done*
  → assert *Done =1* for 1 cycle when algorithm done

Step #1. Pseudo-Code of Algorithm *(behavioral specification)*

```
Done :=0;                                   // Initialization
Ocount := 0;                                                    // Note #1
Data := Input;                                                  // Note #2

repeat                                      // Loop Body
        if (Data_LSB = 1) {
                Ocount := Ocount + 1; }                         // increment 1's count
        Data := Data >> 1;                                      // shift right by 1
until (Data = 0);                                               // Note #3


                                            // Final Steps
Output := Ocount;                                               // Note #4
Done := 1;
```

Notes:

#1: Variable *Ocount* stores the current count of the # of 1's.

#2: Variable *Data* stores the binary input number, read from the *Input* bus, which is iteratively shifted to the right. After each shift, the least significant bit (LSB) of *Data* is examined and removed. (Each right shift pushes a 0 bit into the MSB.)

#3: The loop body ("repeat … until") allows an ***early exit from the loop:*** if the remainder of the bits in the number *Data* is all 0's, then exit → no further 1's can be found.

#4: The result is placed a shared output bus, *Output.*

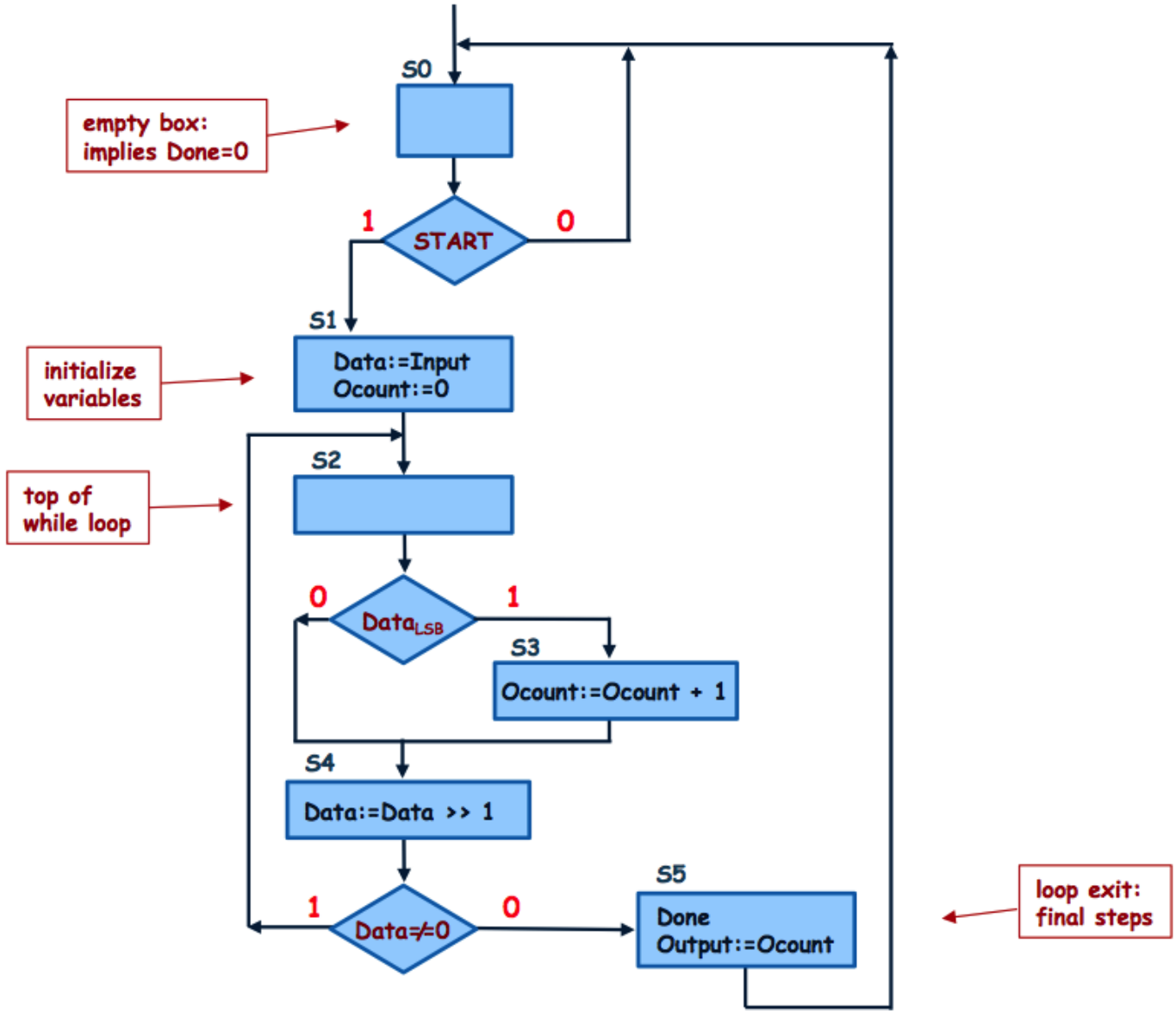
Alternative Pseudo-Code: An alternative pseudo-code algorithm could have the loop test (i.e. checking for loop exit) at top instead of bottom, by replacing the "repeat-until" construct by a "while loop" construct.

Step #2. Write RTL Specification: "Generalized ASM" – MOORE VERSION

A generalized ASM is similar to the basic control ASM presented in Brown/Vranesic book (and Katz book, on reserve), but the former models both datapath operations and control operations in a single unified specification (see Gajski, ch. 8.3, for more details).

Let's do a **Moore version** of a generalized ASM specification (i.e. no "output boxes").

Note: In B/V vs. Gajski vs. Katz books, for ASM's, some of the flowchart box names, notations and symbols are slightly different. We will switch between the notations at times!



Step #3. Allocate (= Select) Datapath Blocks for Operations

Sub-steps: Consider each of the variables in the generalized ASM of Step #2. For each variable, ***(i) determine what functions are required, (ii) select an appropriate datapath block from a library***, and ***(iii) add any "hardwired" (i.e. fixed) inputs.***

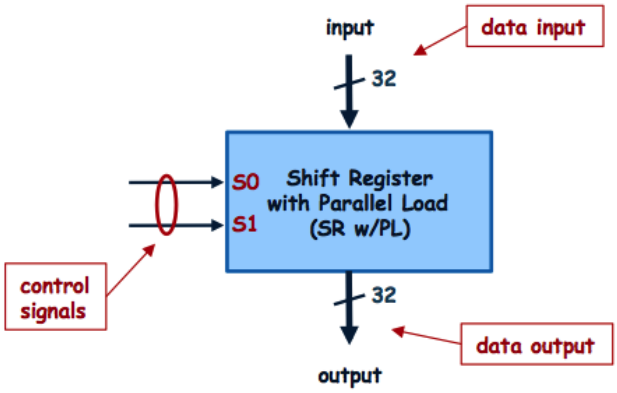
(a) Variable "Data":

Functions required:

- **"load input" (State S1)**
- **"shift-right-by-1" (State S4)**
- **implicit (most states): "hold"**

Selected datapath block (from Gajski, ch. 7) = Shift Register with Parallel Load (SR w/ PL)

Block Diagram:



Truth Table for Control:
(from Gajski, ch. 7)

S1 S0	Operation
0 0	*no change (hold)*
0 1	*LD = load inputs*
1 0	*SHL = shift left*
1 1	*SHR = shift right*

Comments: We only need 3 of the 4 operations: *no change, load inputs, SHR-by-1* (not SHL-by-1). However, we cannot "hardwire" any data or control inputs: we still need both S1 and S0 to select the 3 functions.

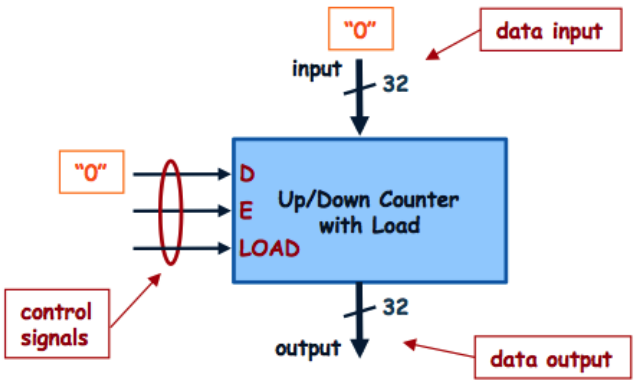
(b) Variable "Ocount":

Functions required:

- **"load input" [constant 0] (State S1)**
- **"increment-by-1" (State S3)**
- **implicit (most states): "hold"**

Selected datapath block (from Gajski, ch. 7) = Up/Down Counter with Parallel Load

Block Diagram:



Truth Table for Control:
(from Gajski, ch. 7)

LOAD	E	D	Operation
0	0	X	*no change (hold)*
0	1	0	*INC 1*
0	1	1	*DEC 1*
1	X	X	*Load inputs*

Bit-width: we will actually only need a version with 6-bit data inputs/outputs

Comments: We only need 3 of the 4 operations: *no change, INC 1 (i.e. increment-by-1), load inputs* (not DEC 1 [decrement-by-1]). In this case, we can optimize the control signals based on this domain-specific usage (see #1 below).

Note that the library component in Gajski ch. 7 lists a bit-width of 4 bits for data inputs/outputs. We assume that the design is parametrizable, and comes in various sizes. The figure above lists a typical 32-bit design, but actually we will select a customized 6-bit version for the final implementation.

Optimization #1: Connect "D" control signal to 0 = "hardwired" input. We can still select the 3 operations using only the *LOAD* and *E* control signals.

Optimization #2: Connect all data inputs to the "hardwired" constant 0 value (i.e. 6 input bits). This optimization is possible because the generalized ASM only lists a load of "0" into Ocount, and no other value.

(c) Variable "Output":

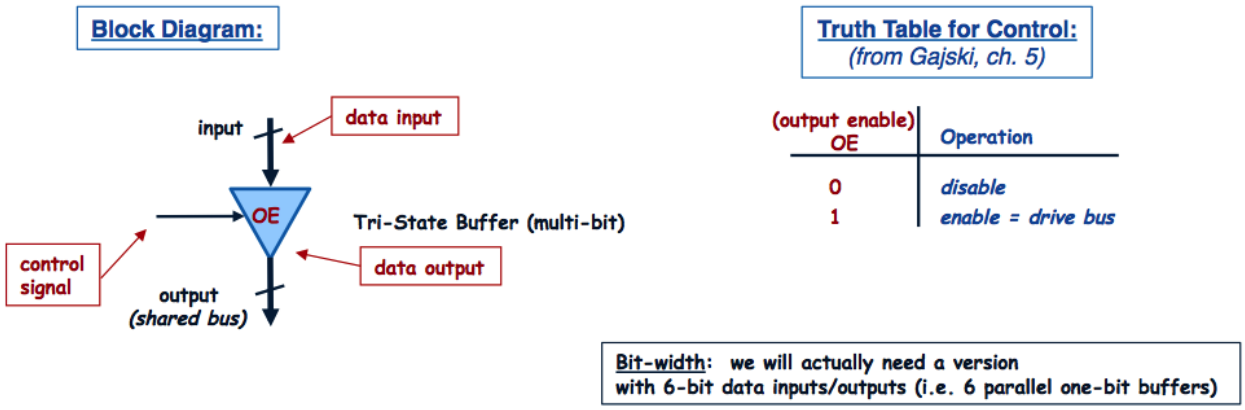
This variable is given as a shared bus (see Gajski, ch. 5). We only want to place a value on (i.e. "drive") the bus when we are done.

The bus is connected to the system by a tri-state driver,which is normally disabled, and only enabled when the algorithm is completed.

Functions required:

- **"enable" (drive bus) (State S5)**
- **implicit (all other states): "disable"**

Selected datapath block (from Gajski, ch. 5) = Tri-State Buffer



Comments: We could generate a separate *OE* control signal from the control unit, to control the tri-state buffer. However, through later analysis of the control ASM, we can re-use an existing wire, and avoid generating a separate *OE* signal:

Optimization #1: In the generalized ASM, note that the control output *Done* is asserted high only in state S5, which is also the only state where the output bus is enabled. So, as an optimization, we do not need a separate *OE* control signal in the final micro-architecture: we can re-use *Done* (control output) to also serve as the control signal for the tri-state buffer.

Comments: *It looks like we are done, but we are not!:* ***How is the status signal "Data ≠ 0" produced?***

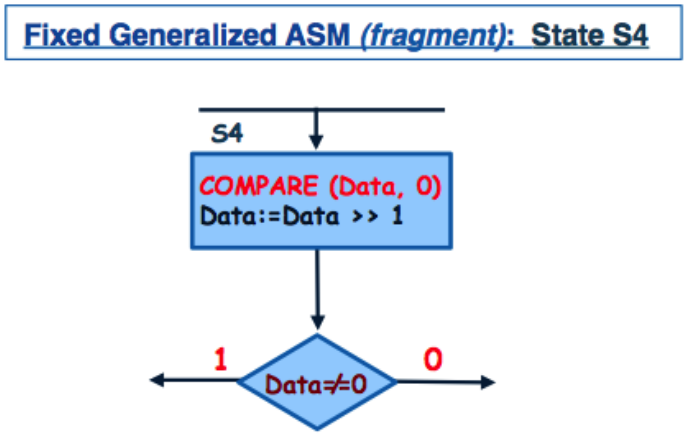
Discussion: Focus on the "decision box" after state S4 in the generalized ASM. It indicates a test of whether "Data ≠ 0". This decision box represents a check of a result (= a status signal) of some datapath operation.

The datapath operation is a "COMPARISON"; since we are checking its result, it must have been computed somewhere, implicitly.

So: let us correct the generalized ASM, and explicitly add the COMPARISON datapath operation (combinational) into state box S4. See below.

(d) "COMPARE" Operation:

Fixed generalized ASM:



Selected datapath block:

Now, let's allocate a datapath block for this operation.

Possibility #1. We could pick a combinational MAGNITUDE COMPARATOR, which compares 2 arbitrary numbers (see Gajski, ch. 5).

Possibility #2. However, since we have a very simple fixed comparison – with 0! – we can select much simpler hardware: a 32-input OR network. The output is 1 whenever the 32-bit data input is not all-0's. The input of the block is the 32-bit *Data* output. The output of the block is the 1-bit status signal, "Data ≠ 0".