Hints for Assignment 3

Question 1

These sample answers to the questions in the first part of Assignment 3 are provided to reduce the time required to complete the assignment. You do not need to use this design. You should still provide answers to the questions posed in the assignment, even if you use the solution given below.

The product of two 16-bit numbers can have up to 32 bits. The running sum, `c`, must therefore be stored in a 32-bit register. The value `b` will be shifted left up to 15 times before the algorithm terminates so it must be stored in a register of at least 31 (16+15) bits wide. The value of `a` is never more than 16 bits so it can be stored in a 16-bit register.

The registers required to implement the algorithm are thus:

```
signal rc unsigned (31 downto 0) ;
signal rb unsigned (30 downto 0) ;
signal ra unsigned (15 downto 0) ;
```

where I have used the variable `ra` to distinguish the input port `a` from the register `ra`.

From the description of the algorithm, the following datapath operations are required:

- set `rc` to zero, `ra` to `a`, and `rb` to `b`
- set `rc` to the sum of `rc` and `rb`
- shift `ra` right by 1 bit and shift `rb` left by 1 bit

other choices are possible as long as some sequence of the chosen operations will implement the multiplication algorithm.

In this case we can use three independent control signals to control each of the above operations, these will be called:

- `load`
- `add`
- `shift`

respectively. Again, other choices are possible as long as the operations can be mapped into unique combinations of the control signals. In this case only four combinations of the three operations will turn out to required so we could also have mapped these four operations into two control signals.

Only two status signals are required from the datapath: one to indicate that the value of `a` has reached zero and one to indicate whether the least-significant bit of `a` is one or zero:

- `azero` - register `a` is zero
- `lsbset` - L.S. bit of `a` is a '1'

The following is a possible choice of controller states and the corresponding operations:

state	operations
`init`	none
`add_shift`	add `rb` to `rc` if LS bit of `ra` is 1 and shift `ra` and `rb`
`done`	none

The state transition table is as follows:

state	reset	azero	lsbset	next state
X	1	X	X	`init`
`init`	0	0	X	`add_shift`
`init`	0	1	X	`done`
`add_shift`	0	0	X	`add_shift`
`add_shift`	0	1	X	`done`

The following example shows how the multiplier would multiply `a=` 5 and and `b=` 3. Each line shows the `reset` input, the controller state and the contents of the registers. The state transitions and register loads occur at the clock edges (which can be imagined to lie in between the lines). It is assumed that the `a` and `b` entity inputs are set to 5 and 3 during the cycle 0.

cycle	`reset`	state	`ra`	`rb`	`rc`
0	1	X	X	X	X
1	0	`init`	5	3	0
2	0	`add_shift`	5	3	0
3	0	`add_shift`	2	6	3
4	0	`add_shift`	1	12	3
5	0	`add_shift`	0	24	15
6	0	`done`	0	24	15
>=7	0	`done`	0	24	15

When comparing numeric values that might be uninitialized you should first convert them to `std_logic_vector` types. The comparison will then be between strings rather than numeric values and will not generate warnings.

Since `a` and `b` are only valid during the one clock cycle during which `reset` is active, you need to load the registers `ra` and `rb` at the end of that cycle. This means the controller will have to be implemented as a Mealy state machine (the `load` output is a function of the input as well as (possibly) the state).

The controller design may have redundant states.