Tips

a)
There are only four inputs to the system you are designing, so creating a truth table is entirely feasible.

Thinking ahead though, do you really need a truth table for TS_3 and TS_0? In what situations (i.e., for what combinations of inputs) would TS_3 be active? In what situations would TS_0 be *in*active? Don't just immediately try to brute-force logic problems; there are very often simplifications that you can find by thinking about the entire system.

A similar line of thought *might* be productive for TS_1 and TS_2, but if you don't see any, falling back on tools like K-maps is also a reasonable approach to the design.

b)
One possible approach to this design would be to break it into smaller, more manageable pieces that more closely align with behavior you can get from the listed building blocks. A reasonable division could be 1) how could the "tally" numbers be converted to a form more conducive to summing, and then 2) assuming that the result of that summing is in an unsigned binary representation, how could it be converted back to a right-justified "tally" representation?

For the first part: in the "tally" number system, what does every single bit *numerically* represent? Given that, how could you sum them if you have unsigned binary adders?

For the second part: what does the counting sequence (0, 1, 2, 3, etc.) look like in six bits of right-justified tally counting? Does that look *similar* to the behavior of any of the common building blocks? If it doesn't *exactly* match, can some simple logic be added to achieve what you need? (don't forget that simple logic gates are available)