Rebuttal of Implementation Concerns for Bit Entanglement

Document number: P0990R0
Date: 2018-04-01
Audience: EWG
Reply-to: Tony Van Eerd. shortlong at forecode.com

We must admit we were somewhat disappointed with the large number of questions from implementors about entanglement of bits, particularly time-entanglement, which we thought was an obvious implementation technique. Even though TE (time-entanglement) is just an optimization and not required (implementors are free to use spatial entanglement), we will happily clarify that, as well as other concerns.

Time Entanglement

There are at least two implementation strategies for TE:

1. Relative Time Entanglement:

A half-bit may be entangled with itself 10 seconds into the future. Flipping the half-bit from `0` to `1` now, immediately causes it to flip 10 seconds from now. (ie assuming no other operations in between, the bit flips to `0` 10 seconds after flipping to `1`). Since in the future it is still entangled with its +10 seconds self, the bit must again immediately flip (back to `0`) 10 more seconds later, etc, thus oscillating between `0` and `1` every 10 seconds indefinitely (unless otherwise modified). Implementers should be careful at program exit, as the (hardware) entangled bit may continue to oscillate after exit, affecting future programs. (More importantly, if they *don't* continue to flip indefinitely, the last flip will not have happened, thus nor the penultimate, etc, ie *no* previous flips will have happened.)

2. Absolute Time Entanglement:

With *absolute time entanglement*, a half-bit isn't entangled with a relative always-10seconds-later self. Instead it is entangled with its *specific* future self, and the future self is entangled with its past self (also known as mirrored entanglement). So a half-bit flipped now causes it's future self to flip, which in turn causes its original self to flip. Which, in fact, it had done. For example, consider a "now" half-bit entangled with its 10s later half-bit. When the now-half-bit is set to `1`, this immediately causes the 10s future half-bit to be set to `0` 10 seconds from now. 10 seconds from now, the half-bit will have flipped to `0`, causing the original/old 'now-half-bit' to be flipped to `1`, which is exactly what had happened when we started. Of course, any subsequent changes require a *new* entanglement. Thus any flip must either be causing a future flip, or be consistent with a past flip.

No Free Lunch

Implementors may be tempted to just entangle over long time distances, such as 10 years instead

of 10 seconds - in an attempt to get what looks like a full bit (for the next 10 years). This is fine *if* they can guarantee the future flip(s). If the future flips turn out not to have happened, then the original bit flipping will have now had failen. There is no free lunch.

Clarification of initial states/values:

It has been noted that if spatially entangled half-bits are contiguous *and* have the same initial value (ie both 0), then they will continue to share the same value throughout the program lifetime (as they both must flip at the same time). Some implementors have noticed that this may lead to significant implementation simplification and optimization, and have thus either assumed or requested that we enforce the constraint of 'same initial value'.

This is of course nonsense. Consider:

```
short short short short short short int x = 0;
short short short short short short int y = 1;
```

This is in no way unreasonable code.

It is not a hardship for implementors, if wishing to use contiguous entanglement, to entangle these half-bits. With a bit of thinking, an intelligent implementor will realize that all that is required for contiguous spatial entanglement is strategic uses of the 'not' operation on one of the entangled bits.

Alternatively, one could choose to only entangle bits that *do* have the same initial value - note there will be at most two half-bits "unpaired" using this technique (1 leftover `1` and 1 leftover `0`), so extra "wasted" half-bits will be required, and that trade off needs to be considered. This is one reason why entanglement pairing is left implementation defined. Similarly for other fractions of bits (1/4, 1/8 etc) - note there can be more leftover, but they are smaller. And two quarter bits each initted to `1` can be entangled together, and then further entangled with a half-bit that is also initialized to `1`. Sorry for not initially highlighting the option of entanglements between different bit fractions (but it does naturally follow from the 'as-if' rule).