**Dan Grissom – July 23, 2015**

This document contains high-level implementation needs to add static single assignment (SSA) form to the current static framework. I see the following needs to get this to work:

1. Add control-flow-graph (CFG) capabilities to the DMFB SSS which will include:
   1. CFG processing – Can largely **port** from dynamic framework
   2. Transfer-in and – out nodes – Can largely **port** from dynamic framework
   3. Language constructs – Can largely **port** from BioCoder extensions (again, currently living in dynamic framework)
2. Pre-compile all CFG nodes (i.e., DAGs, assay) into pin-activation sequences (PAS), which can also include droplet locations and other meta-data for visualization purposes.
   1. **Ability to compile to this level already exists in DMFB SSS (i.e., Schedule, Place & Route), so we basically get this for free**
   2. HOWEVER, we should leave **out** of the pre-compiled PAS/simulation-meta-data, since it can change depending on control flow:
      1. The assay intro routes (initial (time-step 0) dispenses and transfer ins)
      2. The assay outro routes (final (last time-step) outputs and transfer outs)
3. Generate routes for every possible CFG edge:
   1. E.g., For Edge1 which connects DAG1 to DAG2, generate routing results to concurrently route DAG1’s transfer outs to DAG2’s transfer ins simultaneously with DAG2’s initial (i.e., time-step 0) dispense nodes
   2. These should be pre-compiled, just like the DAGs
   3. **We obviously have the basic building blocks to do this type of routing with our Maze Router code**
4. Glue the results of #2 and #3 (which are done “statically” before the assay is “executed”) into the CFG code from #1, which dynamically selects the next PAS to run based on currently-executing DAG’s/assay’s sensor feedback
   1. **This will be where most of the original work comes**, but again, I think most of the harder algorithmic work is already done (e.g., synthesis, point-to-point droplet routing, CFG processing), albeit in the dynamic framework (which is in C++)
5. Initial flow should support:
   1. List scheduling – This may already be done in dynamic framework w/ transfer ins/outs
      1. Path Scheduler might be already done too
      2. With these, we could potentially get the genetic schedulers for free as I believe they just call list/path…but I’m unsure about that
   2. Grissom Path-Binding – Not sure if this exists in the dynamic framework
      1. Left-Edge Binding probably exists already w/ transfer ins/outs…not sure though
   3. Roy Maze-Routing – May need minor modifications to support transfer ins/outs
   4. Individual Pin-mapping – Shouldn’t need modifications
   5. Path-finder Wire Router – Just to have a complete flow, shouldn’t need modifications
6. Initially we’ll just do the pre-compilation at the beginning of our run and then jump into the dynamic run while the pre-compiled DAGs are still in memory.
   1. Eventually, we can support outputting of intermediate files so that you can read in from a library of pre-compiled DAGs/assays to practically take advantage of long-running schedulers, etc.