Pseudocode 1 BUILD_OUT_WITH_LSQS_CIRCUIT

Input: dependency_graph_edges, edges_with_respected_dep, CFG
Output: CIRC

```
1: procedure BUILD_OUT_WITH_LSQS_CIRCUIT
2:     // Initialize data structures
3:     delay_generators ← {}                                   // Delay generator for each predecessor
4:     sync_signals ← {}                                       // Synchronization signals for each successor
5:     CIRC ← ∅                                                // The output circuit
6:     ...                                                     // Other data structures

7:     for all edge ∈ dependency_graph_edges do
8:         if edge ∈ edges_with_respected_dep then
9:             continue
10:        end if
11:        pred ← edge.src                                     // Predecessor memory operation
12:        succ ← edge.dst                                     // Successor memory operation
13:        N ← edge.N                                          // Window size (i.e., number of comparators) used for this edge

14:        // Builds Delay Generator for addresses (top left of Fig. 8)
15:        unless delay_generators[pred] ≠ null ∧ delay_generators[pred].size ≥ N then
16:            if delay_generators[pred] = null then
17:                del_gen ← INSTANTIATE(DelayGenerator, N)
18:            else if delay_generators[pred].size < N then
19:                old_del_gen← delay_generators[pred]
20:                del_gen ← EXTEND(old_del_gen, N)
21:                REMOVE(CIRC, old_del_gen)
22:            end if
23:            delay_generators[pred] ← del_gen
24:            APPEND(CIRC, del_gen)
25:            CONNECT(CIRC, pred.addr, delay_generators[pred].input[0])
26:        end unless

27:        // Builds Delay Generator for done signals (top right of Fig. 8)
28:        ...
29:        // Builds logic to enforce correct window advancement (top center of Fig. 8)
30:        ...

31:        // Builds Address Comparators (bottom left of Fig. 8)
32:        ...
33:        CONNECT_FTD(CIRC, CFG, delay_generators[pred].output[0 .. N-1], comparators[0 .. N-1])
34:        // When Fast Token Delivery is required, connection is done by CONNECT_FTD instead of CONNECT.
35:        ...

36:        // Builds Skip logic (bottom right of Fig. 8)
37:        ...
38:        join_op ← INSTANTIATE(JoinOp, N)
39:        APPEND(CIRC, join_op)
40:        CONNECT(CIRC, skip_comp[0 .. N-1], join_op[0 .. N-1])
41:        sync_signals[succ] ← sync_signals[succ] ∪ join_op.output[0]
42:    end for

43:    // For all successors of all edges in the dependency graph, gate (i.e., Join) the address with all predecessor's sync signals
44:    // Note that there is only one predecessor in Fig. 8 for simplicity
45:    for all succ_op ∈ KEYS(sync_signals) do
46:        succ_sync_signals ← sync_signals[succ_op]
47:        W ← succ_sync_signals.size
48:        succ_join_op ← INSTANTIATE(JoinOp, W + 1)
49:        APPEND(CIRC, succ_join_op)
50:        CONNECT(CIRC, succ_op.addr, succ_join_op.input[0])
51:        CONNECT(CIRC, succ_sync_signals[0 .. W-1], succ_join_op.input[1 .. W])
52:        CONNECT(CIRC, succ_join_op.output[0], succ_op.addr)
53:    end for
54:    return CIRC
55: end procedure
```