2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Creating 5G Test Scenarios, the Constrained-Random way

Keshav Kannan and Eric P. Kim

Intel Corporation

intel

accellera SYSTEMS INITIATIVE

Agenda

- **Introduction**
- System Level Test Scenarios
- Brief Introduction to 5G Specification
- Breaking it Down
- Conclusion

Introduction

- Constrained random (CR) verification at system level
  - Difficult due to the size of the solution-space
  - Vast number of parameters and their inter-dependencies
- Traditional CR limited to stimulus generation
- Shifting CR effort to test-scenario generation

Agenda

- Introduction
- **System Level Test Scenarios**
- Brief Introduction to 5G Specification
- Breaking it Down
- Conclusion

Traditional Vs CR System Level Test Scenarios

- Traditional System Level Test Scenarios:
  - Directed and feature centric
  - Rarely measures coverage
  - Difficult to create test scenarios
- CR System Level Test Scenarios:
  - CR Test scenarios
  - Easy to create scenarios for:
    - Corner case
    - Performance measurement
    - Throughput measurement
  - Downside: Complex constraints

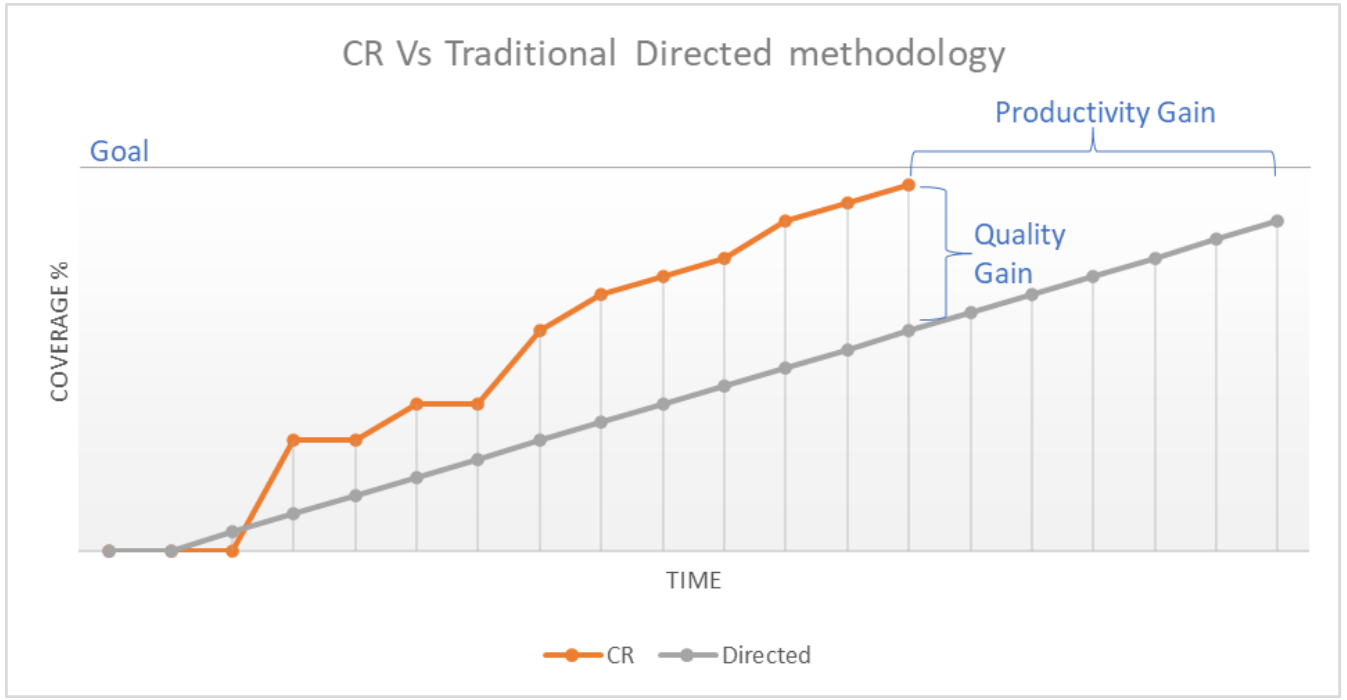


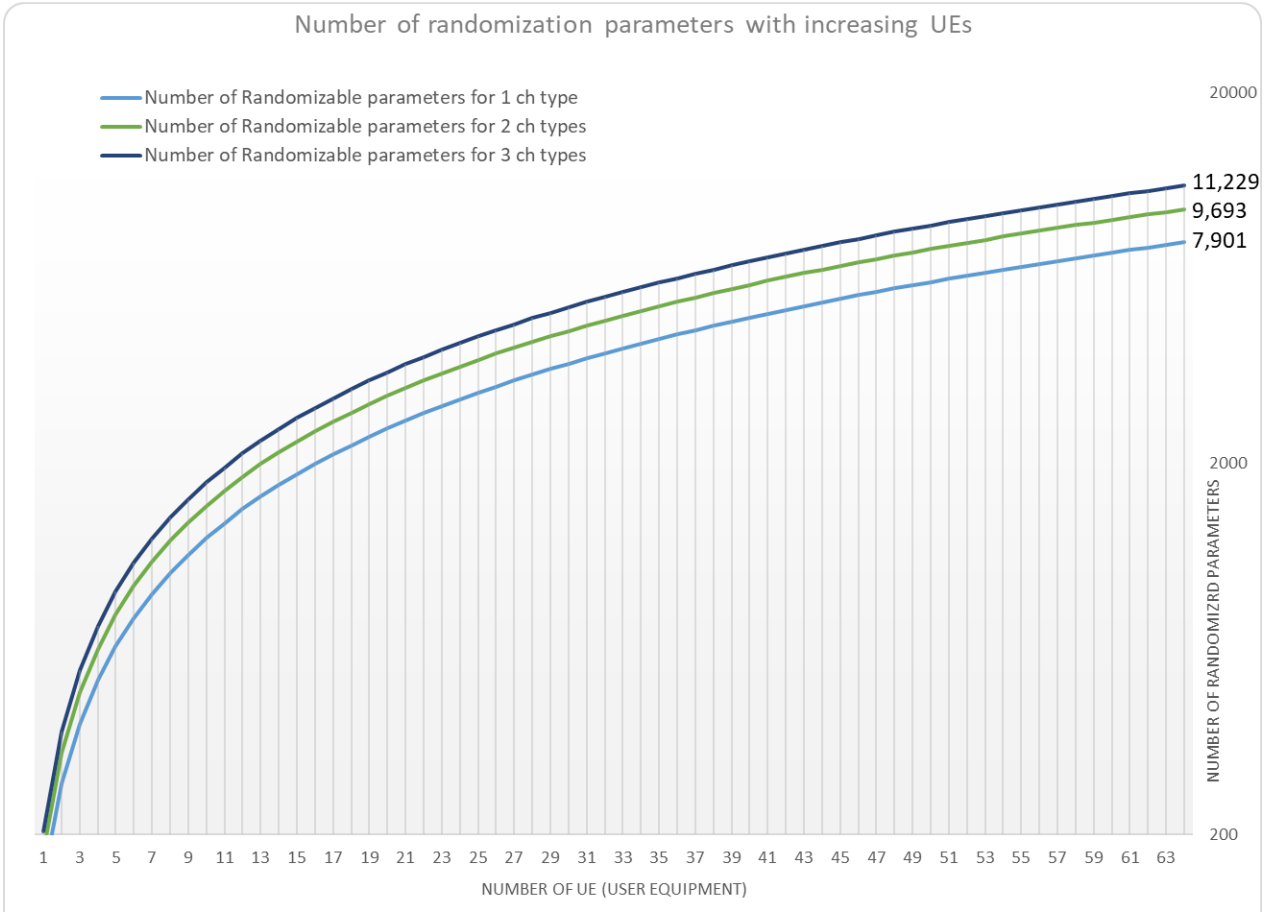
Image Courtesy of “Constrained Random Verification (CRV)” | SpringerLink

Impact on Results

- Verified 5G-NR communication system
- Over 400 high level CR test scenarios defined
- Scenarios’ coverage measurement
- Over 80 critical RTL bugs caught
- Multiple performance, stress, and through-put scenarios created
- CR infrastructure reused at top and unit levels
- Replay of failing tests

Magnitude of the Problem

- To insert channel types PUCCH, PUSCH and SRS into one slot for 64 instances of User Equipment:
  - Constrained-randomization of over 11,000 inter-dependent parameters
  - Randomization load on the tool (simulator)
  - Constraints' management (code)
  - Randomization failure debug/analysis



Agenda

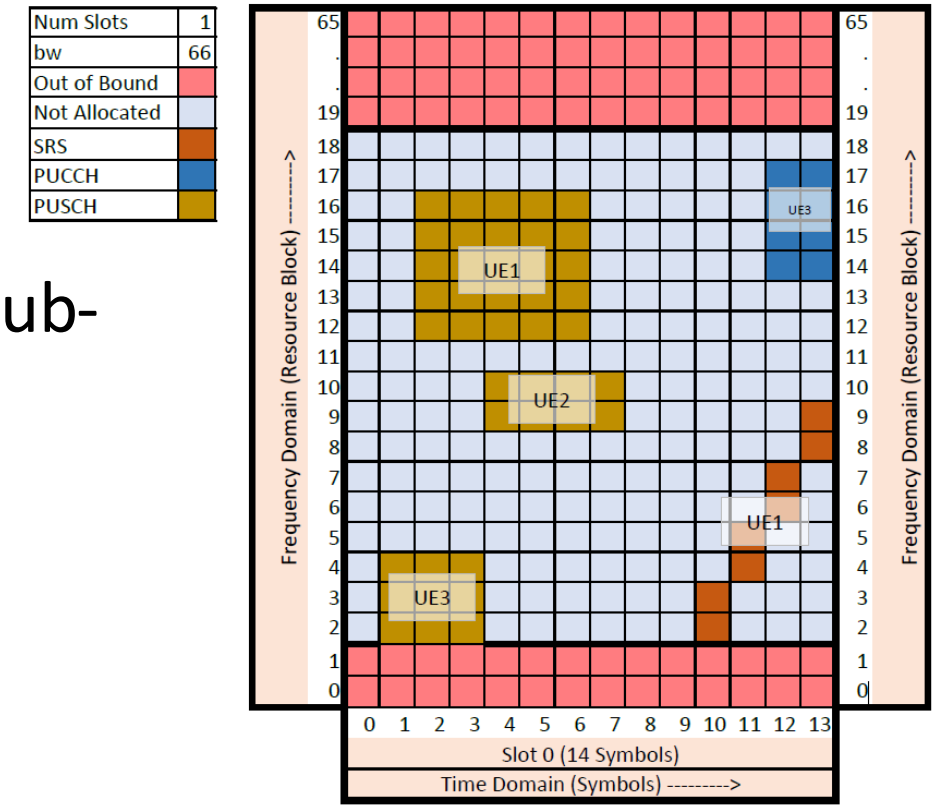
- Introduction
- System Level Test Scenarios
- **Brief Introduction to 5G Specification**
- Breaking it Down
- Conclusion

Brief Introduction to 5G Specification

- 5G network's physical layer communication channels in UL and DL:
  - Synchronization/Reference-signals channels (E.g.: PSS, SSS, PBCH, CSI-RS, PTRS)
  - Control channels (E.g.: PUCCH, PDCCH)
  - Data channels (E.g.: PUSCH, PDSCH)
- Multiple users are mapped to specific resource element (RE) in frequency/time domain in shared resource grid

Resource Grid

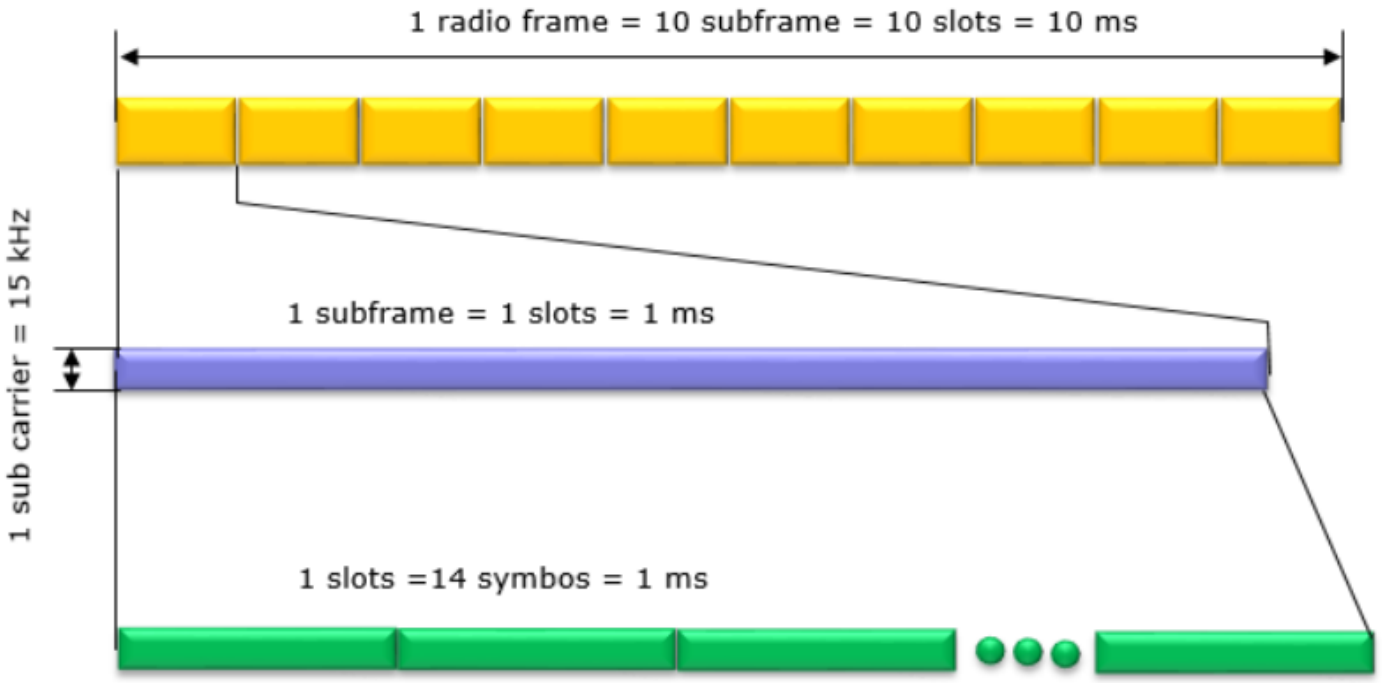
- Resource grid is formed based on system-level (RRC Parameters) and user-level parameters
- Number of antennas, radio frequency, sub-carrier spacing, maximum bandwidth
- Resource grid represents time (x-axis) domain and frequency (y-axis) domain
- Channel-types for various users are allocated resources as shown



Time Domain: Frame, Slots & Symbols

- μ indicates numerology (subcarrier spacing configuration) in NR

μ	N^{slot}_{symb}	$N^{frame,\mu}_{slot}$	$N^{subframe,\mu}_{slot}$
0	14	10	1
1	14	20	2
2	14	40	4
3	14	80	8
4	14	160	16



Frequency Domain: Resource-Blocks

- Resource Block
  - 12 consecutive subcarriers
- Bandwidth part
  - Subset of contiguous common resource blocks

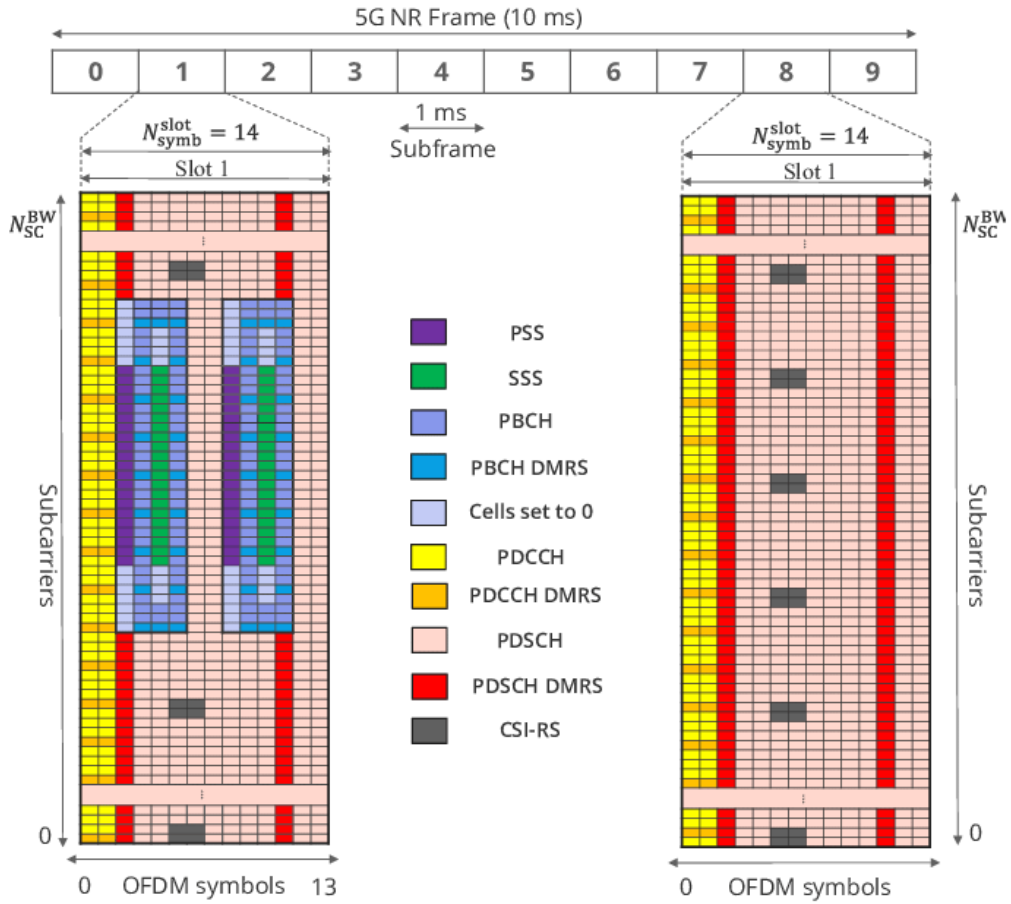


Image Courtesy of "IMT-2020 Key Performance Indicators: Evaluation and Extension Towards 5G New Radio Point-to-Multipoint | IEEE Conference Publication | IEEE Xplore"

Channel-Types

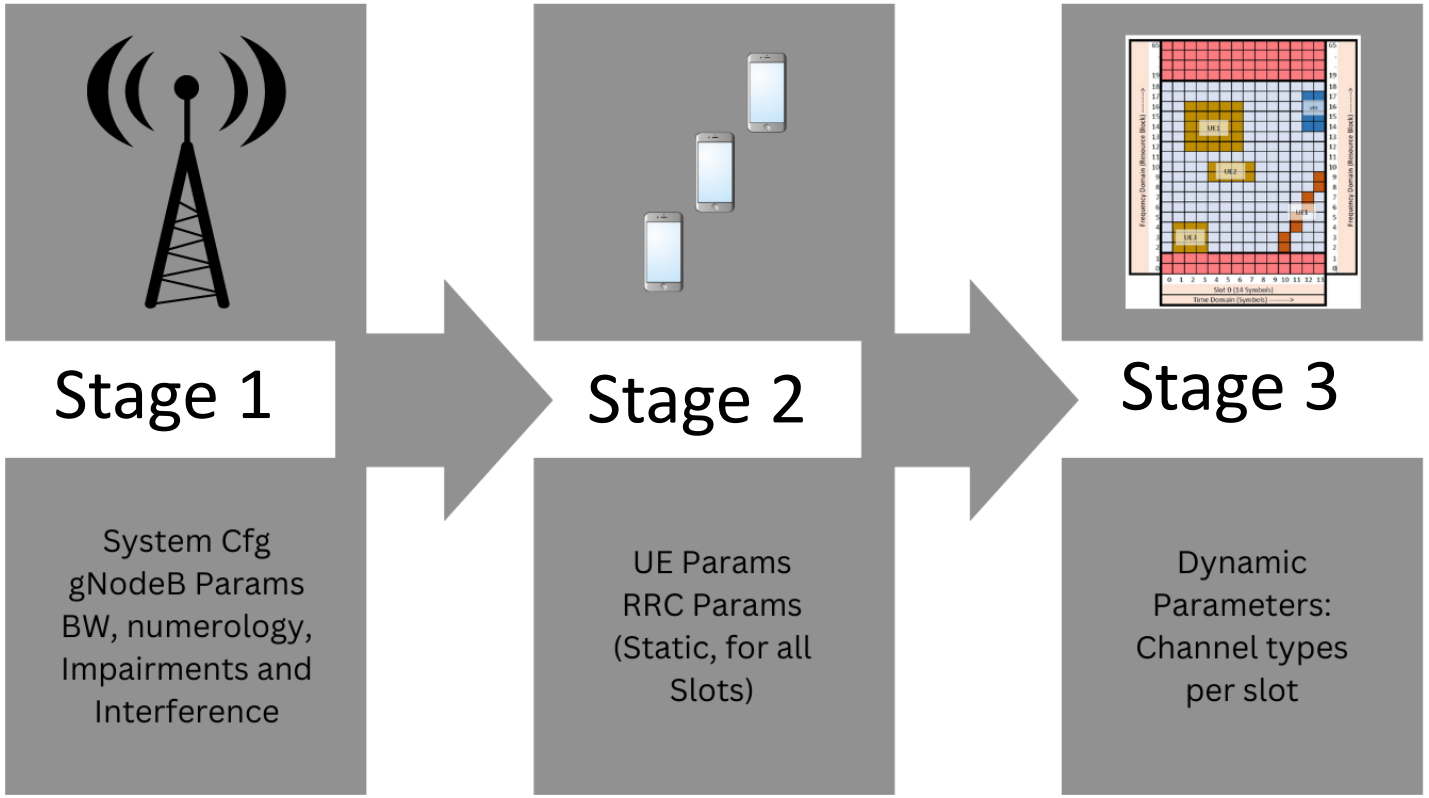
- Users can have different channel types in a resource grid
- Each channel type has specific parameters
- Some parameters determine placement in the resource grid
- No overlap of channel types in resource area
- Constrained-random generation must ensure 5G-NR specification is not violated while allocating channel types.

Agenda

- Introduction
- System Level Test Scenarios
- Brief Introduction to 5G Specification
- **Breaking it Down**
- Conclusion

Breaking it Down

- “Divide and Rule”
- Identifying logical stages to break the randomization
- Three high level stages in 5G test scenario randomization



Breaking it Down – Stage 1: Randomize RE Grid

- BW, fftSize, scSpacing
- bwMHz per CC, carrierFreq, TO/FO (Timing/Frequency offset)
- Builds resource grid based on requested num_slots and num_cc

Breaking it Down – Stage 2: Randomize UE Configurations

- BWP_SIZE, RBG_SIZE, bwp0_startRb, RNTI, SNR, Channels (AWGN/Fading), ChannelType, correlationType.
- PUSCH/PUCCH/SRS Specific Params, HARQ Params
- Slot-repeat interval settings
- All the above are parameters of the UserCfg class' objects for number of UEs for each CC

Breaking it Down – Stage 3: Channel Types

- Randomizing and inserting the desired UE channel-types into the resource grid
- Randomization complexity is highest in this stage
- This stage is further broken down into multiple randomization steps, one channel-type for one user at a time
- User sequence defines the insertion needs

Common Resource Grid

- Randomized channel-types are stored in static resource grid (re_grid_map)
- Resource Grid represents the entire time-domain and frequency-domain of the current test scenario

```
static re_grid_enum re_grid_map[MAX_CC][MAX_SLOTS][NUM_SYM][MAX_RB][]; // Channel-Type occupying the RB.
```

User Sequence (Stage 3 Randomization)

- In this sequence, every slot in every CC is inserted with PUCCH, PUSCH and SRS for every UE
- Each channel type is inserted one at a time

```
virtual task body();
  bit status;
  ··· /* omitted */
  for(int ue=0;ue<sm_num_ue;ue++) begin     // Loop through all the UEs //
    for(int cc=0;cc<re_grid.num_cc;cc++) begin      // Loop through all the CCs //
      for(int sl=0;sl<sm_num_slot;sl=sl++) begin         // Loop for all slots //
        // Select UE to be inserted in desired CC and Slot //
        // The set_ucl() function is described later in the paper. //
        pucch_f2_o.set_ucl(ue, cc, sl);
        // Insert PUCCH Format-2 in all slots. User parameters will be //
        // picked based on the selected UE number provided in set_ucl //
        status = pucch_f2_o.randomize();
        ··· /* omitted: prints based on randomization status */

        // Select UE to be inserted in desired CC and Slot //
        pusch_type1_o.set_ucl(ue, cc, sl);
        // Insert PUSCH Type-1 in all slots. User parameters will be //
        // picked based on the selected UE number provided in set_ucl //
        status = pusch_type1_o.randomize();
        ··· /* omitted: prints based on randomization status */

        // Select UE to be inserted in desired CC and Slot //
        srs_ch_o.set_ucl(ue, cc, sl);
        // Insert SRS in all slots. User parameters will be picked //
        // based on the selected UE number provided in set_ucl //
        status = srs_ch_o.randomize();
        ··· /* omitted: prints based on randomization status */
      end
    end
  end
endtask
```

Features Covered

- Lists the features covered from an Uplink perspective

Channel-Type	Features
PUSCH	Type-0 (non-contiguous) and Type-1 (Contiguous) allocations
	Single-User/Multi-user, multiple-input, multiple-output (SU/MU-MIMO
	Frequency Hopping
	UCI Over PUSCH (HARQ, CSIP1 and CSIP2)
	All possible DMRS allocation
	PTRS insertion
	HARQ
	MCS Selection
PUCCH	Format 0 and Format 2, can be extended to Formats 1 and 3
	PUCCH Frequency Hopping
	PUCCH FMT0 multi-user Multiplexing
SRS	Support for KTC (Transmission Comp) 2, 4
	Support for numAntPorts of 1, 2 and 4
	All possible cyclicShift combinations supported
	Frequency Hopping is supported
	BandwidthCfg (mSRS and cSRS)
	Supports Multi UE Overlap in allocation

Randomization Load Analysis

- Randomization load analysis for 64 UE insertion of PUCCH Fmt0, PUSCH Type1 and SRS into one slot

Randomization Stage	Randomization parameter count	Number of randomization calls	Total Randomization Parameter Count (Load)
First: RE Grid, gNodeB	29	1	29
Second: UE Configs	69	64 (Num UE)	4416
Third: PUCCH Fmt0	28	64 (Num UE)	1792
Third: PUSCH Type1	51	64 (Num UE)	3264
Third: SRS	24	64 (Num UE)	1536
Total		257	11037

Agenda

- Introduction
- System Level Test Scenarios
- Brief Introduction to 5G Specification
- Breaking it Down
- **Conclusion**

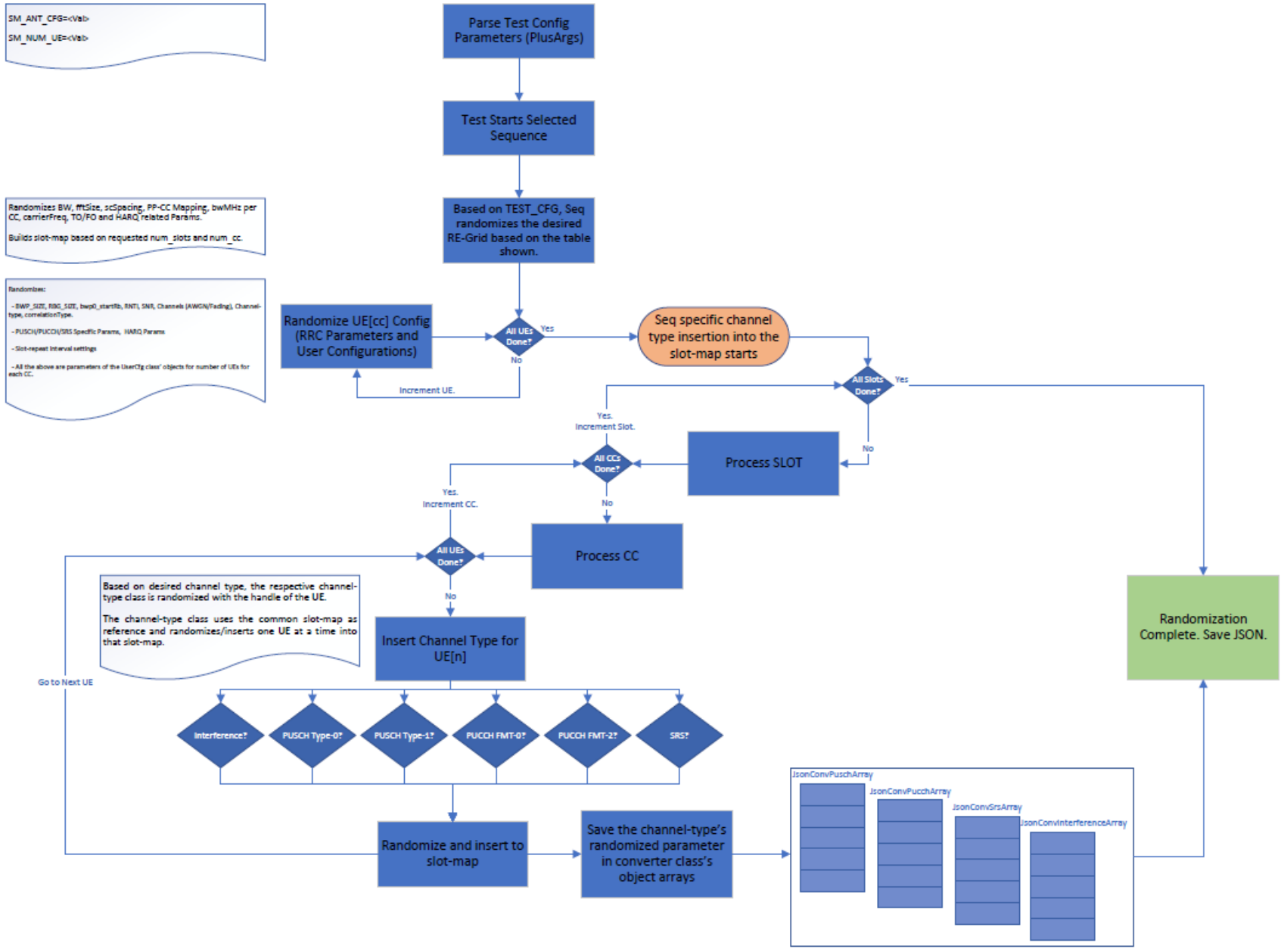
Conclusion

- Constraints to be divided hierarchically
- “Divide and Rule” on the randomization calls
- Use of common RE Grid in the 5G realm made this possible
- User sequence drives the required test scenario
- Constrained-Random system level test scenario generation made possible

Questions?

Thank You!

Backup


SM_ANT_CFG=<Val>
SM_NUM_UE=<Val>
Parse Test Config Parameters (PlusArgs)
Test Starts Selected Sequence
Randomizes BW, fftSize, scSpacing, PP-CC Mapping, bwMHz per CC, carrierFreq, TO/FO and HARQ related Params.
Builds slot-map based on requested num_slots and num_cc.
Based on TEST_CFG, Seq randomizes the desired RE-Grid based on the table shown.
Randomizes:
- BWP_SIZE, RBG_SIZE, bwp0_startRb, RNTI, SNR, Channels (AWGN/Fading), Channel-type, correlationType.
- PUSCH/PUCCH/SRS Specific Params, HARQ Params
- Slot-repeat Interval settings
- All the above are parameters of the UserCfg class' objects for number of UEs for each CC.
Randomize UE[cc] Config (RRC Parameters and User Configurations)
All UEs Done?
Yes
No
Increment UE.
Seq specific channel type insertion into the slot-map starts
All Slots Done?
Yes
Yes. Increment Slot.
No
All CCs Done?
Process SLOT
Yes. Increment CC.
No
All UEs Done?
Process CC
Based on desired channel type, the respective channel-type class is randomized with the handle of the UE.
The channel-type class uses the common slot-map as reference and randomizes/inserts one UE at a time into that slot-map.
No
Insert Channel Type for UE[n]
Go to Next UE
Interference?
PUSCH Type-0?
PUSCH Type-1?
PUCCH FMT-0?
PUCCH FMT-2?
SRS?
Randomize and insert to slot-map
Save the channel-type's randomized parameter in converter class's object arrays
JsonConvPuschArray
JsonConvPucchArray
JsonConvSrsArray
JsonConvInterferenceArray
Randomization Complete. Save JSON.

```
// ue_inst[ue][cc] is the array of UE configurations that are randomized in the //
// 2nd stage of randomization. These are readily available to use here in the   //
// 3rd stage of randomization.                                                  //
// Below constraint determines the allocation of RE blocks with x0 and x1       //
// indicating the time-domain bounds in the form of starting and ending symbol  //
// numbers respectively. The parameters y0 and y1 indicate the frequency-domain //
// bounds in the form of starting and ending RE Block numbers respectively.     //

//--------------------------------Pusch_base.sv--------------------------------//
// Time-Domain constraint for PUSCH Allocation //
 constraint c_symLen {
   if(mapType[ue][cc][sl] == 0) { // Type A
     x0[ue][cc][sl] == 0;                    // Starting symbol is always 0 for type A//
     if(ue_inst[ue][cc].ulDmrsTypeApos == 3) {
       x1[ue][cc][sl] inside {[4:NUM_SYM-1]}; // PUSCH Length can be 5~14 in case of normal CP
     } else {
       x1[ue][cc][sl] inside {[3:NUM_SYM-1]}; // PUSCH Length can be 4~14 in case of normal CP
     }
     if(ue_inst[ue][cc].puschFreqHoping == 1) {
       (x1[ue][cc][sl]-x0[ue][cc][sl]) > 7; // Table 6.4.1.1.3-6 of 38.211
     }
   } else { // Type B
     x0[ue][cc][sl] inside {[0:NUM_SYM-1]}; // PUSCH Starting Symbol can be 0~13 in case of Normal CP
     x1[ue][cc][sl] inside {[0:NUM_SYM-1]}; // PUSCH Length can only be 1~14 in case of Normal CP
     x0[ue][cc][sl] <= x1[ue][cc][sl];
   }
 }

 //--------------------------------PuschType1.sv--------------------------------//
 // Frequency-Domain constraint for PUSCH Type-1 Allocation //
 constraint c_reAlloc {
   y1[ue][cc][sl] >= y0[ue][cc][sl];
   y0[ue][cc][sl] inside
       {[ue_inst[ue][cc].bwp0_startRb : ((ue_inst[ue][cc].bwp_size+ue_inst[ue][cc].bwp0_startRb)-4)]};
   y1[ue][cc][sl] inside
       {[ue_inst[ue][cc].bwp0_startRb : ((ue_inst[ue][cc].bwp_size+ue_inst[ue][cc].bwp0_startRb)-1)]};

   foreach(re_grid_act[cc][sl][l]) { // SYM //
     foreach(re_grid_act[cc][sl][l][m]) { // RB //
       foreach(re_grid_act[cc][sl][l][m][n]) { // LAYER //
         if((l inside {[x0[ue][cc][sl]:x1[ue][cc][sl]]}) &&
         (m inside {[y0[ue][cc][sl]:y1[ue][cc][sl]]})) {
           if(re_grid_c::re_grid_map[cc][sl][l][m][n] == NA) { re_grid_act[cc][sl][l][m][n] == PUSCH;}
           else { re_grid_act[cc][sl][l][m][n] == NA;}
         } else { re_grid_act[cc][sl][l][m][n] == NA;}
 } } } }
```

Program 4: Time and frequency domain constraints for PUSCH allocation. Time domain constraints covered entire PUSCH space and was part of PUSCH base class, while frequency domain constraints were split into separate classes based on type. Constraints for type1 are shown.

TABLE VI: Randomization time compared to total verification execution time.

Feature	Randomization time	Simulation runtime
1 slot, 1 UE, 1 Channel-Type	< 30 sec	1 hour per slot
1 slot ,4 UEs, 1 Channel-Type	< 1 Min	1 hour per slot
1 slot, 64 UEs, 1 Channel-Type	~ 30 mins	1 hour per slot
1 slot, 64 UEs, 3 Channel-Types	~1:30 mins	2 hours per slot