Creating 5G Test Scenarios, the Constrained-Random way

Keshav Kannan and Eric P. Kim
Intel Corporation
2200 Mission College Blvd
Santa Clara, CA 95054

Abstract

Constrained random verification (CRV) at system level is often considered difficult due to the size of the solution-space due to the number of parameters and their inter-dependencies dictated by the requirements. The overwhelming complexity lies not only in understanding the system level requirements, but also in creating valid constraints and generating a valid scenario via a constraint-solver. In this paper, we present a multi-staged methodology to generate practical constraints that can be easily solved. Our methodology has been applied to a 5G-NR (5th Generation - New Radio) communication link by effectively mapping the 5G-NR specification to the SystemVerilog constraints and delivering a constraint-random (CR) generated test scenario for effective verification of any uplink/downlink system. The constraint-random engine focuses on creating a 5G-NR resource-grid. The resource-grid is a representation of the time and frequency domain. The CR infrastructure maps the resource allocation requirements present in the 5G-NR specification into SystemVerilog constraints and builds the slot-map with the desired channel-types. The key challenge this paper addresses is to identify the intersection region for all the requirements per channel type abstracted as parameters along with inter-dependency between channels. The solution focuses on breaking down the randomization load effectively to make randomization at this grand scale a possibility. In the end, we successfully demonstrate a practical 5G NR system level CR test scenario generation framework that can randomize a full scenario with approximately 1% of verification execution time overhead.

I. Introduction

Constrained random verification (CRV) is widely used for block level verification and is considered industry standard due to its verification efficiency, increasing coverage while reducing verification time and effort [1]. However, writing good constraints is an involved task that requires deep understanding of the underlying solver [2], [3]. This complexity often limits the use of CRV to block or module level [1]. In this paper we provide a methodology that enables system level CRV of a 5G-NR communication system by breaking down the constraints for over 400 inter-dependent random variables into multiple hierarchies while maintaining a common constraint that applies to all levels. In addition, focus of CRV was shifted to constrained-random system-level test scenario generation from a traditional constrained-random stimulus generation with stimulus generated via a reference model. To achieve this, the effort has been channelized into structuring a randomization infrastructure and putting the use of powerful randomization engines provided by tools that support SystemVerilog. The remainder of the paper is as follows. Section II gives a brief overview on the 3GPP (3rd Generation Partnership Project) 5G-NR specification. Section III and IV describes the challenges of CRV applied to a 5G-NR communication system and our proposed solution. Section V outlines the results achieved by the proposed solution, while Section VI concludes this paper.

II. Brief introduction to 5G specification

Like its predecessors, 5G networks at the physical layer have channel mapping divided into three broad sections as shown in Fig. 1. The communication channels are divided into synchronization channels, control channels, and data channels in both the uplink and downlink directions. The 5G NR's physical channels of both uplink and downlink for multiple users are mapped to specific resource elements (REs) in the frequency/time domain represented on a resource element grid (Fig. 2). Each of the channel-types' resource allocation depends on various sections of the 5G-NR specification represented by different parameters, functions, and tables. Each of the channel-types have different functionalities as part of the medium between the gNodeB (5G base station) and the user equipment (UE) such as a smartphone. The uplink and downlink channel-types between the gNodeB and multiple UEs share the same space in the frequency/time domain.

The primary synchronization signal (P-SS) and the secondary synchronization signal (S-SS) are the two downlink synchronization signals used for downlink frames. In order to broadcast master information block (MIB), a physical broadcast channel (PBCH) is used. The channel state information reference signal (CSI-RS) is a reference signal (RS) that is used in the downlink direction for channel sounding. Channel sounding refers to measurement and estimation of channel characteristics such as power delay profile, path loss, path delay, power angular spectrum, correlation matrix, doppler spectrum, etc. It is used to measure the characteristics of a radio channel so that the correct modulation, code rate, beam shaping, etc., can be employed. Physical downlink control channel (PDCCH) and physical downlink shared channel (PDSCH) are used to provide control and data, respectively.

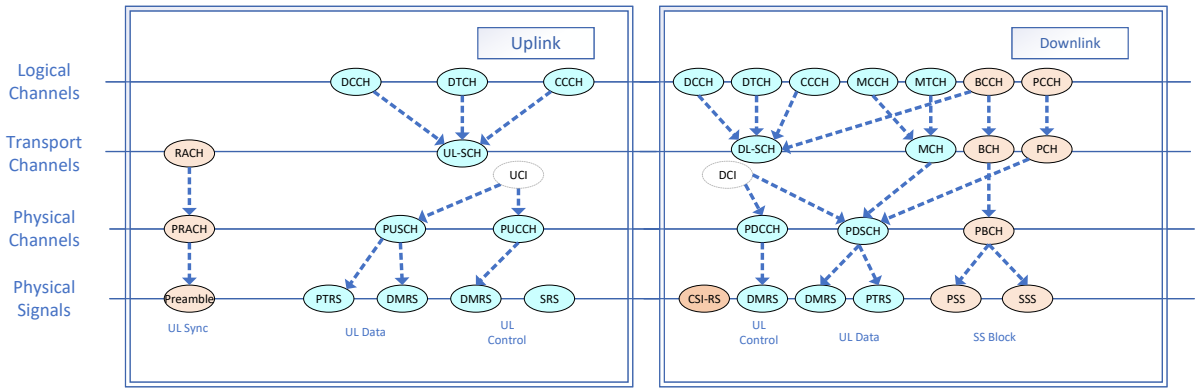
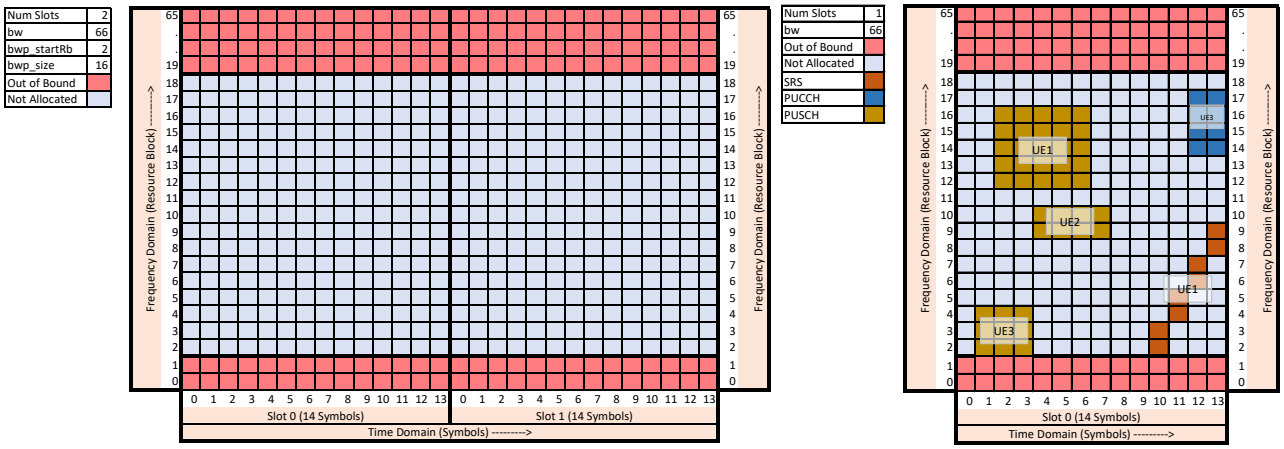


Fig. 1: 5G NR channel mapping. Courtesy of https://info-nrlte.com/.



(a) (b)

Fig. 2: 5G NR uplink RE grid: (a) empty RE grid for two slots, and (b) RE grid with multiple channel/UEs allocated.

Uplink frames are synchronized based on downlink signals. Physical uplink control channel (PUCCH) is the control channel transmitted by uplink, which contains information including channel quality info, acknowledgments, and scheduling requests. Physical uplink shared channel (PUSCH) is used by uplink users to transmit data to the base station. Physical random access channel (PRACH) is used by an uplink user to initiate contact with a base station. The base station broadcasts some basic cell information, including where random-access requests can be transmitted. A UE then makes a PRACH transmission asking for PUSCH allocations, and the base station uses the physical downlink control channel (PDCCH) to provide downlink control information (DCI) that indicates where the UE shall transmit PUSCH.

Configuring an end-to-end 5G link is accomplished by setting up a multitude of parameters from system level, user level and channel level, both static and dynamic. First, the capability of the gNodeB and each UE need to be set. These include system level static configurations such as number of antennas, radio frequency, sub-carrier spacing, maximum bandwidth to support, maximum number of users that can be supported etc. Based on these capabilities, higher layer radio resource control (RRC) parameters [4] are communicated from gNodeB to each UE. Once a link is established, a corresponding RE grid is created as specified in [5] and depicted in Fig. 2, which is a representation of resources available to be used for communication. The RRC parameters specify bulk of the configurations each channel can take, including partial slot/frequency/time allocation of resources. RRC parameters are considered static as they are only updated via reconfiguration update messages. Finally, through DCI, dynamic parameters are communicated to the UE, which provide the per slot configuration information such as uplink grants and the corresponding frequency/time domain allocation of PUSCH. Each uplink grant is given by its scheduling DCI and has its own PUSCH configuration parameters, allowing these parameters to differ for every PUSCH.

As an example, we show how PUSCH allocation, as specified in [6], is determined. Time and frequency are independently allocated and combined to form a rectangular allocation. Time domain allocation is determined via start length indication value

TABLE I: Nominal resource block group (RBG) size P. P is determined based on BWP size and configuration.

Bandwidth Part Size	Configuration 1	Configuration 2
1-36	2	4
37-72	4	8
73-144	8	16
145-275	16	16

(SLIV) as shown in (1).

$$\text{SLIV} = \begin{cases} 14 \cdot (L-1) + S & \text{if } (L-1) \leq 7 \\ 14 \cdot (14 - L + 1) + (14 - 1 - S) & \text{else} \end{cases} \tag{1}$$

where $0 < L \leq 14 - S$, S is start symbol index, and L is number of consecutive symbols. SLIV is determined by a two-step process, involving RRC and DCI. First, via RRC, a PUSCH resource allocation table is configured. Each row of this table contains SLIV value and which mapping type (A or B) to use. Then via DCI, a row index to be used is communicated for each uplink grant. To complicate matters, based on mapping type, there are further constraints on which time domain allocation are valid.

Two types of frequency domain allocation exist, type 0 and type 1. A complex method (beyond scope of this paper) is employed in determining which type to use, which depends on both RRC and its scheduling DCI. Once the type is determined, the DCI provides either the RBG bitmap, in case of type 0, or the resource indication value (RIV), in case of type 1. For type 0 allocation, RBG bitmap indicates which RBG are allocated for PUSCH, with number and size of RBG determined via RRC parameters, as shown in Table I and (2), with bandwidth part (BWP) size and configuration 0 or 1 given as RRC parameters.

$$N_{\text{RBG}} = \left\lceil \left(N^{\text{size}}_{\text{BWP},i} + \left(N^{\text{start}}_{\text{BWP},i} \mod P \right) \right) / P \right\rceil \tag{2}$$
$$\text{where} \begin{cases} \text{size of first RBG is} & \text{RBG}^{\text{size}}_0 = P - N^{\text{start}}_{\text{BWP},i} \mod P \\ \text{size of last RBG is} & \text{RBG}^{\text{size}}_{\text{last}} = \left(\left(N^{\text{start}}_{\text{BWP},i} + N^{\text{size}}_{\text{BWP},i} - 1 \right) \mod P \right) + 1 \\ \text{size of all other RBG is} & \text{RBG}^{\text{size}}_i = P \text{ for } 0 < i < \text{last} \end{cases}$$

where RBG is resource block group, N_{RBG} is the total number of RBGs available within a bandwidth part i of size $N^{\text{size}}_{\text{BWP},i}$, and $N^{\text{start}}_{\text{BWP},i}$ is the starting resource block number for the allocation.

For type 1 allocation, similar to SLIV, RIV provides the start RB and length of the frequency allocation, again where the final allocation is determined by combining RRC and DCI parameters, as shown in (3).

$$\text{RIV} = \begin{cases} N^{\text{size}}_{\text{BWP}}(L_{\text{RBs}} - 1) + \text{RB}_{\text{start}} & \text{if } (L_{\text{RBs}} - 1) \leq \left\lfloor N^{\text{size}}_{\text{BWP}}/2 \right\rfloor \\ N^{\text{size}}_{\text{BWP}} \left(N^{\text{size}}_{\text{BWP}} - L_{\text{RBs}} + 1 \right) + \left(N^{\text{size}}_{\text{BWP}} - 1 - \text{RB}_{\text{start}} \right) & \text{else} \end{cases} \tag{3}$$

where $N^{\text{size}}_{\text{BWP}} - \text{RB}_{\text{start}} \geq L_{\text{RBs}} \geq 1$, RB_{start} is start RB index, and L_{RBs} is number of consecutive RBs.

III. Problem statement and analysis

Verification at 5G NR physical system level is challenging as can be seen by the vast possible scenarios covering the RE grid and the underlying number of permutations and combinations, evidenced by the simple PUSCH allocation example given in Section II. To generate a valid 3GPP 5G NR compliant scenario, combining different channel types from multiple users, first we must ensure all user/channels are allocated within the RE grid randomly without any overlap. Taking all the parameters across all users and all channel types, then randomizing according to the constraints in the specification, then finding the allocation on the RE grid is an extremely large load onto the randomization engine and a seemingly impossible task. As an example, inserting just two channel types (PUCCH and PUSCH) into the RE grid for a single user involves around 400 randomized parameters and with more users and channel types sharing a single RE grid, the complexity of solving the constraint increases exponentially.

One approach to solve this problem is to come up with an elaborate functional verification list which will have a complete set of features that need to be verified and to create directed test scenarios to achieve the functional verification list. This would involve identifying all combinations of channels and users that can be allocated in a single RE grid, and provide all the parameters that will create a valid test scenario to cover the directed test. The problem with this is that it is time consuming and error prone and majority of time will be spent in debugging the test scenario rather than the design.

Without a good verification randomization infrastructure, it is nearly impossible to create scenarios where we can test performance and throughput with maximum utilization and with maximum user allocations in the RE grid. For example, it

```
typedef enum {
      PSS = 'h0,  // 5G-NR PSS (Primiary Synchronization Signal)
      SSS = 'h1,  // 5G-NR SSS (Secondary Synchronization Signal)
     PBCH = 'h2,  // 5G-NR PBCH (Physical Broadcast Channel)
    CSIRS = 'h3,  // 5G-NR CSI-RS (Channel Status Information - Reference Signal)
      SRS = 'h4,  // 5G-NR SRS (Sounding Reference Signal)
    PRACH = 'h5   // 5G-NR PRACH (Physical Random Access Channel)
    PDCCH = 'h6,  // 5G-NR PDCCH (Physical Downlink Control Channel)
    PDSCH = 'h7,  // 5G-NR PDSCH (Physical Data Shared Channel)
      TRS = 'h8,  // 5G-NR TRS (Tracking Reference Signal)
     DMRS = 'h9,  // 5G-NR DMRS (Demodulation Reference Signal)
     PTRS = 'hA,  // 5G-NR PTRS (Phase Tracking Reference Signal)
    PUCF0 = 'hB,  // 5G-NR PUCCH Format-0 (Physical Uplink Control Channel Format-0)
    PUCF1 = 'hC,  // 5G-NR PUCCH Format-1 (Physical Uplink Control Channel Format-1)
    PUCF2 = 'hD,  // 5G-NR PUCCH Format-2 (Physical Uplink Control Channel Format-2)
    PUCF3 = 'hE,  // 5G-NR PUCCH Format-3 (Physical Uplink Control Channel Format-3)
    PUSCH = 'hF,  // 5G-NR PUSCH (Physical Uplink Shared Channel)
      OFB = 'h10, // Out of Bound. Outside the BW region.
    BLANK = 'h11, // Forced blank RB
       NA = 'h12  // Not Allocated
  } re_grid_enum;

  static re_grid_enum re_grid_map[MAX_CC][MAX_SLOTS][NUM_SYM][MAX_RB][]; // Channel-Type occupying the RB.
  static int unsigned re_grid_ue[SM_MAX_CC][SM_MAX_SLOTS][NUM_SYM][MAX_RB][]; // UE occupying the RB.

  // Function to initialize/reset the RE grid to Not-Allocated //
  // Described later are user sequences, that shall be used for actual //
  // randomization and insersion of various channel-types for different UEs. //
  static function void reset_grid();
    // Loop through[All CCs][All Slots][All Symbols][All Resource-Blocks][All Layers] //
    foreach(re_grid_map[cc,sl,sy,rb,la])
      re_grid_map[cc][sl][sy][rb][la] = NA; // Not-Allocated //
  endfunction
```

Program 1: Enumuration of channel types in a RE grid and the RE grid (`re_grid_map`) implemented as a static array.

would be a daunting task to manually come up with a valid allocation for 64 users into a slot with channel types of PUCCH, PUSCH and SRS all together in the same slot.

IV. Proposed solution

The constraint-random infrastructure translates resource allocation requirements briefly presented in Section II into SystemVerilog constraints and builds the slot-map with the desired channel-types. This paper will focus on uplink (UL) channel-types and its extension to downlink (DL) is straightforward. The task of inserting a large number of channel types into a RE grid can be achieved by breaking down the randomization task into multiple stages. At each stage, additional randomization is performed while respecting the dependency of prior stages. RE grid is maintained as a static array of enumerated fields, which is shared among all the randomization stages. The enumeration contains all the channel-types and additional state indicators for a resource-block in the RE grid. The RE grid and enumeration are shown in Program 1.

The first stage of randomization deals with determining cell/link level parameters that are unrelated to UE configuration, and define the size and shape of the RE grid, such as numerology, bandwidth, bandwidth-part (BWP), subcarrier spacing, FFT (fast Fourier transform) size and number of slots. At this stage, our intended number of UEs is passed on to the randomization engine. A list of test modes we have defined is given in Table II.

For the second stage of randomization, RRC parameters are generated via constrained-randomization as per the RRC protocol specification [4]. This includes both gNodeB (3GPP-compliant implementation of the 5G-NR base station) and UE specific RRC parameters. Some UE specific RRC parameters will influence the resource allocation of channels. Bandwidth part size, resource block group (RBG) size, and puschRbgConfig are few examples, where description of determining RBG size was given in Table I, and its corresponding SystemVerilog constraint is given in Program 2 . All the UE specific parameters are randomized in the second stage for the number of UEs that the user intends to simulate. The UE parameters randomized shall be applicable for all the slots.

RRC parameters pertaining to PUCCH that are randomized are:

- `DataScramblingSequence`
- `DmrsScrambling-Sequence`
- `F2MaxCodeRate`
- `F0F1initialCs`

TABLE II: 5G NR randomization test modes.

TEST Mode	nChains	numCC	numerology	SCS* (band†)	Mode	FFT Size	bandwidth (number of RBs)
0	2	1	3	120 kHz (FR2)	1cc_2x2	1K	{32, 66}
1	2	2	3	120 kHz (FR2)	2cc_2x2	1K	{32, 66}
2	2	4	3	120 kHz (FR2)	4cc_2x2	1K	{32, 66}
3	4	1	1	30 kHz (FR1)	1cc_4X4	1K/2K/4K	{11, 24, 38, 51, 65, 66}/{106, 133}/{189, 217, 245, 273}
4	4	2	1	30 kHz (FR1)	2cc_4X4	1K/2K/4K	{11, 24, 38, 51, 65, 66}/{106, 133}/{189, 217, 245, 273}
5	4	1	0	15 kHz (FR1)	1cc_4X4	1K/2K/4K	{25, 66}/{106, 133}/{216, 270}
6	4	2	0	15 kHz (FR1)	2cc_4X4	1K/2K/4K	{25, 66}/{106, 133}/{216, 270}
7	4	1	2	60 kHz (FR1)	1cc_4X4	1K/2K	{11, 18, 24, 31, 38, 51, 65, 66}/{93, 107, 121, 132, 135}
8	4	2	2	60 kHz (FR1)	2cc_4X4	1K/2K	{11, 18, 24, 31, 38, 51, 65, 66}/{93, 107, 121, 132, 135}
9	2	3	3	120 kHz (FR2)	3cc_2x2	1K	{32, 66}
10	2	1	1	30 kHz (FR1)	1cc_2X2	1K	{11, 24, 38, 51, 65, 66}
11	2	2	1	30 kHz (FR1)	2cc_2X2	1K	{11, 24, 38, 51, 65, 66}
12	2	4	1	30 kHz (FR1)	4cc_2X2	1K	{11, 24, 38, 51, 65, 66}
13	2	1	0	15 kHz (FR1)	1cc_2X2	1K	{25, 52}
14	2	2	0	15 kHz (FR1)	2cc_2X2	1K	{25, 52}
15	2	4	0	15 kHz (FR1)	4cc_2X2	1K	{25, 52}

* sub-carrier spacing
† FR1 denotes frequency range 1, sub-6 GHz, and FR2 denotes frequency range 2, mmWave

```
// Generates random bwp_size per UE/CC, corresponding rbg_size and bwp0_startRb
// num_rb is the Bandwidth value randomized based on different constraint in the same class
// rbg_size is the resource block group size randomized based on different constraint in the same class
constraint c_bwp_rbg_size{
  bwp_size inside {[4:num_rb]};
  puschRbgConfig inside {1, 2};
  if(puschRbgConfig == 1){ // Configuration 1 in the table
    if(bwp_size inside {[1:36]})   {rbg_size == 2}; // Refer to table in 38214-f20's 5.1.2.2.1
    if(bwp_size inside {[37:72]})   {rbg_size == 4};
    if(bwp_size inside {[73:144]})  {rbg_size == 8};
    if(bwp_size inside {[145:275]}) {rbg_size == 16};
  } else { // Configuration 2 in the table
    if(bwp_size inside {[1:36]})    {rbg_size == 4}; // Refer to table in 38214-f20's 5.1.2.2.1
    if(bwp_size inside {[37:72]})   {rbg_size == 8};
    if(bwp_size inside {[73:144]})  {rbg_size == 16};
    if(bwp_size inside {[145:275]}) {rbg_size == 16};
  }
  bwp0_startRb inside {[0:(num_rb-bwp_size)]};
}
```

Program 2: SystemVerilog constraint for determining bandwidth part (start RB and size), puschRbgConfig, and corresponding RBG size, P.

- `HoppingId`
- `GroupSequenceHopping`
- `F2numOfPrbs`

RRC parameters pertaining to PUSCH that are randomized are:

- `DataScrambling`
- `xOhPusch`
- `ulPtrsTimeDensity`
- `ulPtrs-FreqDensity`
- `FreqHopingEnable`
- `frequencyHoppingOffset`
- `ulPtrsMaxNrofPorts`
- `ulPtrsReOffset`
- `ulPtrsPresent`
- `ulPtrsScramblingSequence`
- HARQ (Hybrid automatic repeat request, which is a combination of high-rate forward error correction and automatic repeat request) related parameters

RRC parameters pertaining to SRS that are randomized are:

- `ScramblingIdentity`
- `srsLambda`
- `SymRange`

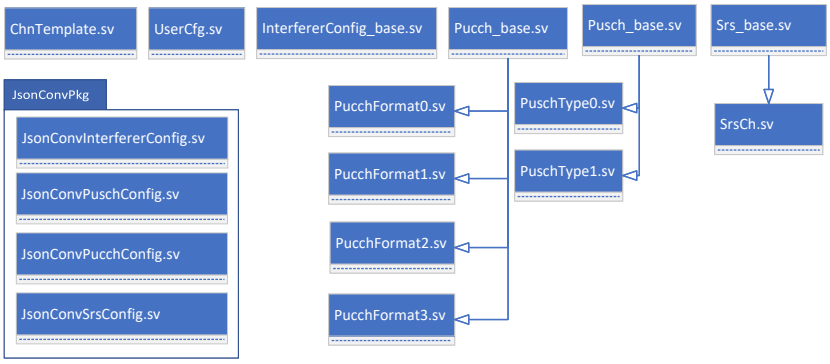


Fig. 3: Hierarchy of constraint classes.

- `PowOffsetRef`

At this stage, we have the slot-map set up, with a randomized, but empty, RE grid as shown in Fig. 2(a). Based on the randomized value of BWP size and BWP start RB, the region that can be used for allocation can be determined, shown by the "Not Allocated" region. BWP is the active region where all the channel types shall be allocated in.

The final stage of randomization can now be performed. With the resource-grid being ready and with the availability of randomized UE RRC parameters, we can start randomizing and inserting the desired UE channel-types into the resource-grid as per a scenario's needs. Randomization complexity is highest in the final stage, which is a series of randomizations, taking a single channel-type of a selected user and randomizing and inserting into the resource-grid, one at a time. The various channel-types' parameters decide the placement of the channel-type on the resource-grid, and the randomization engine will need to refer to the static slot-map each time to constrain the placement of the channel-type and ensure there is no overlap or violation of any of the specifications' requirements. The object-oriented structure of the infrastructure divides the randomization effort into various constraint classes as shown in Fig. 3.

The user can develop scenarios/use-cases using SystemVerilog universal verification methodology (UVM) to tailor different needs. An example sequence to achieve this is shown in Program 3. This sequence is an example and the order of insertion is not restricted in any way. Every time, a channel-type is inserted, the resources on the resource-grid will be blocked. The sequence can be kept as simple as possible and all the complexity is limited to the channel-type classes, whose handle can simply be randomized to insert into the resource-grid, one channel-type at a time.

The entire randomization stages are summarized in a flow chart as shown in Fig. 4.

That brings us to the complexity involved in the root classes to see how the specification is mapped to SystemVerilog constraints. The various channel-type parameter configurations are derived out of different regions of the specifications, and the one configuration all the channel-types depend upon is the time-frequency domain allocation within the RE grid. The scenario generation infrastructure has been designed to tackle this key region of intersection. This is done by breaking down the channel-types' insertion process into the RE grid. The static slot map maintained by the infrastructure as shown earlier in Fig. 2(a) is updated every time a channel-type is inserted into it, thereby restricting the next item to the remaining available resources only. If we consider X0 and X1 as the time-domain parameters and Y0 and Y1 as the frequency-domain parameters, then the region that needs to be allocated for a particular UE would be the rectangle denoted by the diagonals formed by (X0, Y0) and (X1, Y1) on the resource-grid. A SystemVerilog constraint for allocation of PUSCH is shown in Program 4.

PUSCH resource allocation is done as part of third-stage randomization, meaning the resource-grid has already been setup and initialized followed by randomization of RRC parameters and UE specific parameters, which are ready for use to perform resource allocation. The time-domain constraints extracted from the specification are orthogonal to frequency-domain constraints, and the constraint is added to the PUSCH base class. The frequency domain constraints differ between type-0 and type-1 allocation, and due to the large difference between the two types, sub-classes of PUSCH have been made as seen in Fig. 3. Prior to the third-stage randomization call, the `set_ucl(userNum, slotNum, ccNum)` function sets the targeted user, slot and CC where the channel-type is intended to be inserted. The RE grid has the resource blocks that are marked as Not-Allocated and these blocks are the ones that will be targeted for insertions. The constraint loops through the selected slot, selected CC and looks for the Not-Allocated resource blocks and automatically uses any available block randomly to allocate the channel-type for the given user. At the end of the randomization, the allocated resources are committed to the channel-type in the resource-grid by marking the allocated resource blocks as PUSCH and to the intended UE, making it unavailable for future channel-types' insertion. The code snippet uses a temporary RE grid (`re_grid_act`) to map the current allocation, which is then saved back into the static `re_grid_map` at the end of randomization. The corresponding

```
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
```

Program 3: Example randomization sequence.

TABLE III: Number of parameters randomized at various stages.

Feature	Randomization Stage	Randomization parameter count
RE Grid	1st	16
gNodeB	1st	13
RRC Parameters and User Equipment Parameters	2nd	69
PUSCH Type 0	3rd	54
PUSCH Type 1	3rd	51
PUCCH Format 0	3rd	28
PUCCH Format 2	3rd	27
SRS	3rd	24

RBs in the static variable `re_grid_ue` are marked with the desired UE number. Upon successful insertion into the slot-map, all the randomized parameters of every channel-type are stored in a conversion class' object list, which is used to build a json file at the end of all the user-sequence. The json file is generated at time-0 during simulation and passed onto the Python based reference model, which uses the json file as the configuration to generate the bit-true input stimulus (used to inject at top level or at block level) and expected data (used as golden reference) for all desired nodes in the design.

V. Results

The key achievement of the multi-staged randomization is the ability to break down the constraints and isolate the constraints as much as possible in the respective classes, reducing the potential of constraints conflicting among different channel-types. Constraints were contained with the respective parameters and the only link among channels were the resource availability in the RE grid. Resource availability was a joint constraint utilized while randomizing the channel-type's parameters at the time of its insertion.

Table III lists the number of parameters that were randomized for uplink. The various features supported by the randomization infrastructure are mentioned in Table IV. Table V provides analytical perspective of the randomization parameter load that a randomization engine would need to handle without the structural break-up of the overall randomization task.

Despite the complexity, the resulting CR had very low overhead in verification time. Table VI depicts the total test execution time of randomizing and generating a verification scenario based on the 5G CR infrastructure. As seen in the table, verification time overhead of CR was around 1% of total verification time.

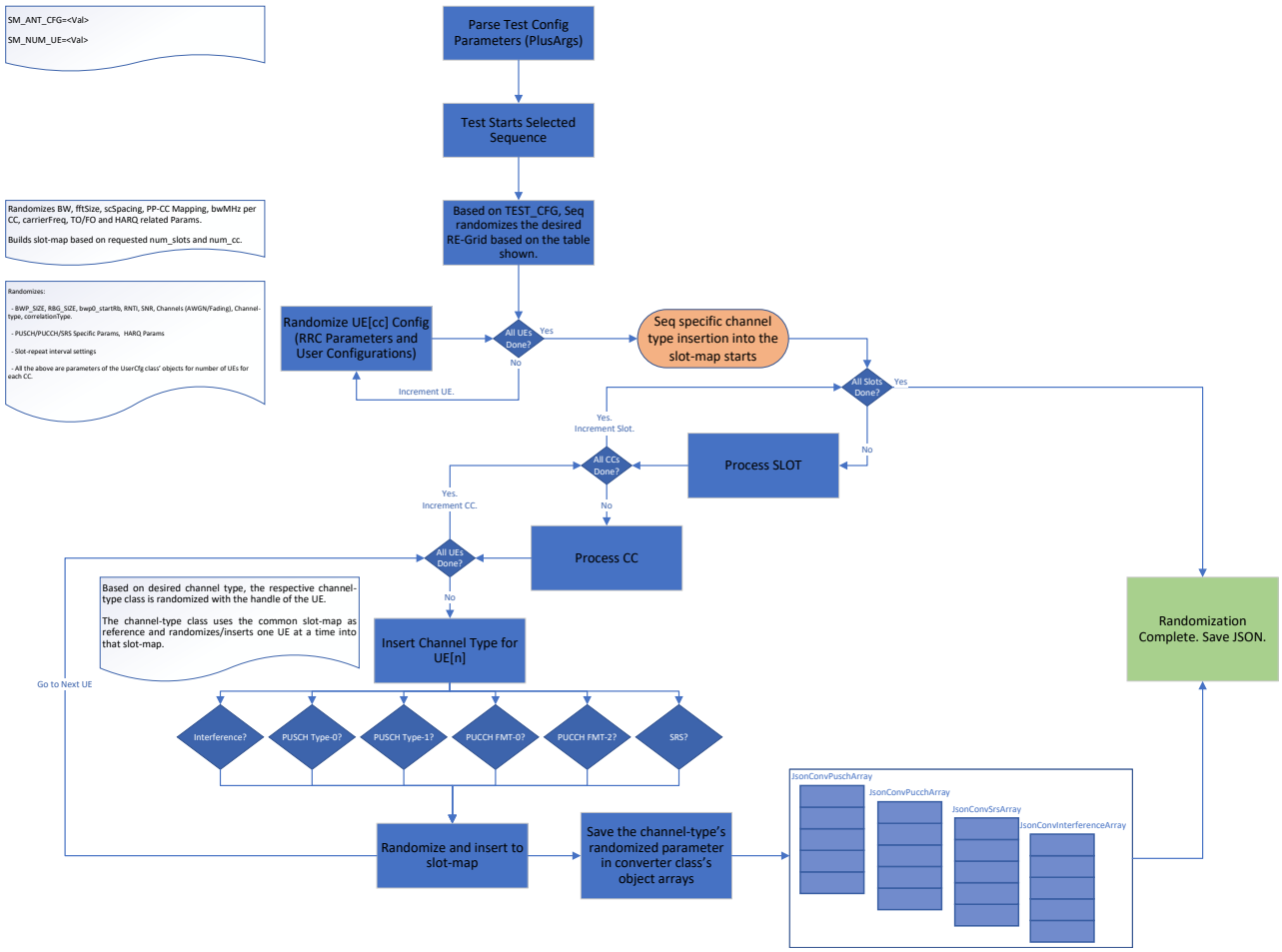


Fig. 4: Flow chart indicating one of the possible ways a sequence can generate a scenario.

TABLE IV: Uplink features covered by the constraint-randomization infrastructure.

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

Furthermore, the constraint-random infrastructure was used to come up with over 400 high level test scenarios that were randomized in every regression during the course of the project. The verification infrastructure was complimented with an elaborate functional coverage targeted to measure the quality of the stimulus. The functional coverage infrastructure came with over 80 coverpoints primarily targeting the channel-types' parameters.

```
// 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 V: 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

Multiple test scenarios could be created using the CR infrastructure, and combined with random seeds, verification goals were effectively achieved. Total constraint-random scenarios derived were 428; and after having run multiple seeds every week, over 80 RTL bugs post design-freeze were caught. Most of the bugs that were caught were critical, resulting in a hang, albeit in the specific scenario. The CR infrastructure was instrumental in generating thousands of random scenarios in combining all the channel-types for multiple UEs within a slot, exercising and stressing the design in ways that directed tests wouldn't do justice. Multiple performance sequences were added such as maximum throughput (full allocation, max CC), stress sequences

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

with 64UE allocations over multiple slots and PUCCH format 2 large RB allocations with high payload sizes.

The CR infrastructure was easily reused at various levels of the testbench, as a simple plug-n-play, and one of the most used features was to replay a failing test without the need to re-randomize. Most importantly, we could take a test failing at one level (unit) and replay it at another level (top) and vice-versa. This was useful to replay many top-level failures at the respective block for fast debug turnaround.

VI. Conclusions

In this paper, we have successfully created constraints for a full 5G uplink. Constraints were hierarchically divided into multiple stages, and utilized a common RE grid to tie all channels together. The resulting constraints were easy to manage and sequences could be written to direct verification towards less covered configurations. Overall, overhead of generating a valid 5G uplink scenario was approximately 1% of total verification execution time. This shows that, even with a system as complex as 5G NR, with a hierarchical multi-stage approach in writing constraints, system level CRV is possible with low overhead.

References

[1] M. Rathi and A. Chandran, "System-level random verification: How it should be done," in *DVCon US*, 2019.

[2] D. Rich, "System verilog constraints: Appreciating what you forgot in school to get better results," in *DVCon US*, 2020.

[3] A. Yehia, "The top most common system verilog constrained random gotchas," in *DVCon-Europe*, 2014.

[4] 3GPP, "NR; Radio Resource Control (RRC); Protocol specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.331, 2022, version 17.1.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3197

[5] ——, "NR; Physical channels and modulation," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211, 2022, version 17.2.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[6] ——, " NR; Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.214, 2022, version 17.2.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216