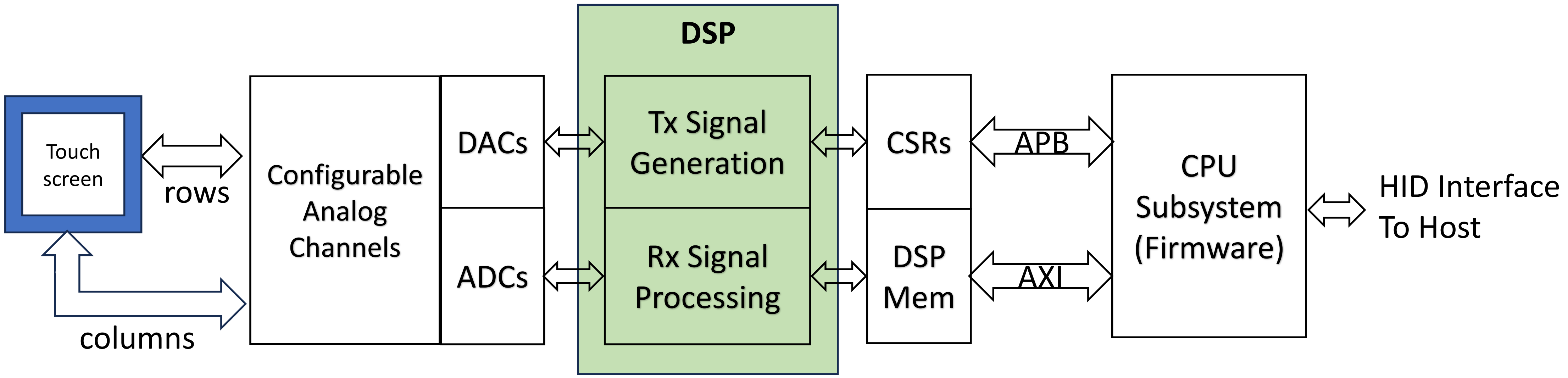
SigmaSense technology - Fast, continuous, low voltage data capture with intelligent digital signal processing moves analog challenges to the digital domain where software defined sensing delivers orders of magnitude improvements.

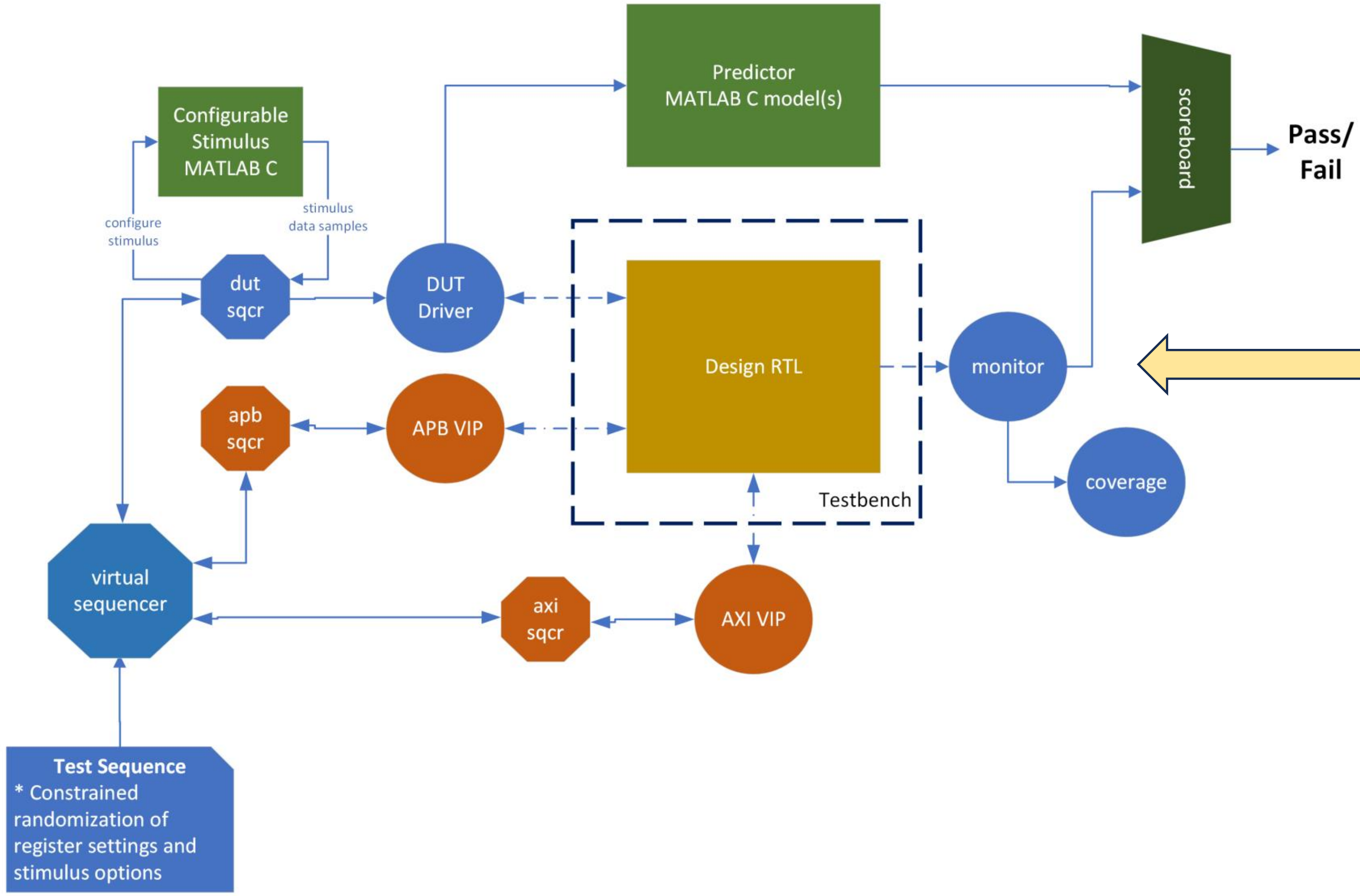


SigmaSense Touchscreen SOC Block Diagram

SigmaSense DSP Methodology:

- Use MATLAB modeling to refine system architecture (digital + analog) to achieve desired performance at minimum gate count.
- Model defines DSP data path bit width at each stage of the pipeline to understand quantization effects.
- DSP RTL design requirements are extracted from the detailed MATLAB models.
- Design Verification – MATLAB models provide exact bit-accurate prediction for DUT
  - Design engineer can do initial debug with testvector data files exported from MATLAB sims and used as stimulus/checking in a simple SV testbench
  - DPI-C models from MATHWORKS DPIGEN are used by DV team for stimulus and predictors
    - In a given simulation the CSR settings will be set via constrained randomization and applied to DUT, Stimulus and Predictor models (num of channels, frequencies, amplitude, phase)
    - Seed from SV can be passed as an input arg to stimulus models for additional randomization such as addition of noise

MATLAB DPI-C models in UVM testbench



Monitor/Predict internal signals as well as DUT output.

DSP Modeling with MATHWORKS SIMULINK

- GUI based
- Cycle based (clock cycles are intuitive for RTL and DV engineers)
- ASIC flow support
  - DPI-C model export with option for SV testbench generation
  - uvmbuild – generate a UVM testbench from a Simulink model (creates DUT, sequence and scoreboard components with options for driver, monitor and predictor subcomponents)
- Simulink was not used in our project as our expertise is with MATLAB

DSP Modeling with MATHWORKS MATLAB

- SigmaSense has extensive experience and codebase with MATLAB for architectural and implementation modeling
- MATHWORKS DPIGEN utility supports export of DPI-C models that can be integrated into an SV TB
  - DPIGEN can also export an SV testbench
- Sample (vs cycle) based - All data samples are generated or processed in one DPI-C function call.
  - A sample is data associated with a particular clock cycle in the DUT.
  - Memory utilization to complete the function call is proportional to the number of data samples needed for the simulation.
  - Touchscreen Rx data is processed in "frames" with a specific number of samples per frame so our sample count is quantized by number of frames to include in a simulation run.

MATLAB DPIGEN – key issues

- DPIGEN enforces syntax restrictions on the MATLAB code.
  - Legacy code may require updates to support running DPIGEN.
  - Add %#codegen comment to matlab function, CODEGEN will highlight any coding issues

- Variable size multi-dimension arrays that are MATLAB function i/o arguments must be flattened to variable length vectors before running DPIGEN.
  3D array example: array A(128 channels, 64 frequencies, N samples)
  - This adds a burden on the coding of MATLAB functions
    MATLAB *reshape* utility is generic solution
    A(:) reshapes A into a single column vector

  - UVM testbench has to deal with the flattened arrays
    - Access array elements via complex indexing OR
    - Convert a flattened array back to n-dimensional SV array for use in TB
    - DV engineer must have knowledge of the MATLAB internal array dimensions and of how the flattening was done.

MATLAB DPIGEN – key issues

- MATLAB indexing is 1:N, SV is 0:N-1
- MATLAB array flattening by default is column major order, SV is row major order
  Default behavior can be changed via coder.rowMajor
- “double” datatype matlab args translate to “real” datatype in SV
  - In/out args of C function are “real” so conversion between “logic” and “real” types is required.
  - Recommend hiding these DPI-C specifics in “predictor” so “scoreboard” can be more generic.
- Predictor C model will produce an array of expected data for entire simulation
  - Scoreboard will check DUT output cycle by cycle, updating index into predictor array on each cycle.
  - Similarly, a driver will index through a Stimulus array cycle by cycle.

MATLAB C model example:

- MATLAB DPIGEN (runs in Windows OS)
  - produces zip file with .c, .h and .sv files
  - Unzip in linux sim environment and run gcc
    MATHWORKS provides example porting_DPIC.mk script
  - .so library file is generated

- Testbench filelist.f – include .sv file with DPI-C function(s)
  - *firFilter_dpi_pkg.sv*

- Simulation command line
  - *xrun –dpi –sv_root ../cpred/lib_firFilter –sv_lib lib_firfilter.so*

MATLAB C model example:

- firFilter_predictor.sv
  - Declare handle

    chandle objhandle_DPI_firFilter;
    objhandle_DPI_NG1_firFilter = DPI_firFilter_initialize(objhandle_DPI_firFilter);

  - Call using function(s) provided in firFilter_dpi_pkg.sv

```
DPI_firFilter_output1( objhandle_DPI_firFilter,  bitsSDM,  sdmOut,  decRatioFIR,
                                          firOut_size, bitsFIR);

firOut = new[firOut_size];

DPI_firFilter_output2(firOut);
```

MATLAB DPI-C: Summary of key issues for DV

- Up front agreement between RTL, DV and MATLAB authors on module partitioning, interfaces and important internal signals to be compared is important

- Keeping track of indexing into flattened multidimensional variable size arrays can be confusing.
  DV engineer needs to understand the matlab internal array structure before flattening and how flattening was done (column major vs row major ordering)