Digital Payload Modeling for Space Applications

LOCKHEED MARTIN

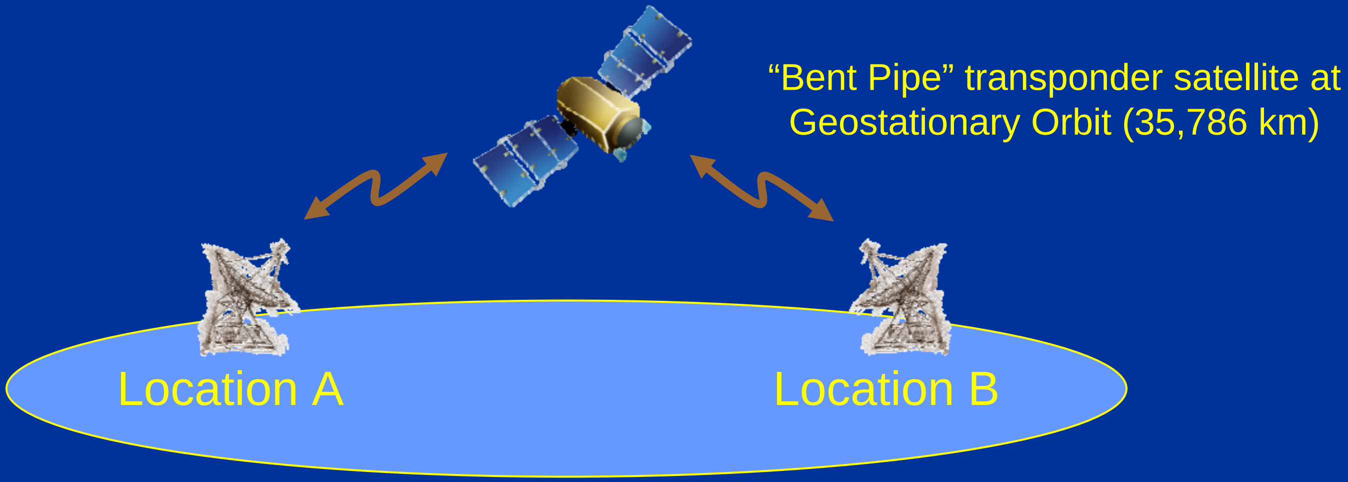
Bradford S. Watson
Staff Engineer
Advanced Algorithm Development Group

Overview

• In recent years in the commercial or government space businesses, there has been a movement away from RF analog payloads for pure “bent-pipe,” or “transponder” applications.

- • Up until even the recent past, these payloads have been fixed band plan, implemented with all analog electronics.

• Today’s customers are looking towards all digital payloads that utilize digital signal processing (DSP) to accomplish the same goals while providing enhanced mission capability and flexibility.


“Bent Pipe” transponder satellite at Geostationary Orbit (35,786 km)
Location A
Location B

The Challenge

- Design a **Modular Agile digital Payload (MAP)** to take the place of the traditional analog communications satellite payload.
- The payload must have the following characteristics:
    - Be *spaceworthy*; i.e., highly reliable, and highly tolerant of radiation effects.
    - Be *reconfigurable*; that is, able to be altered on-orbit for different functions as business conditions change.
    - Be *highly flexible* and *agile* with respect to bandplan and channelization, so that bandwidth for users can be assigned "on-demand," and spectrum can be arbitrarily allocated anywhere within the full bandwidth of a beam.
    - Be *modular*, and *re-useable* for many different programs, all with different requirements.

The Early Days

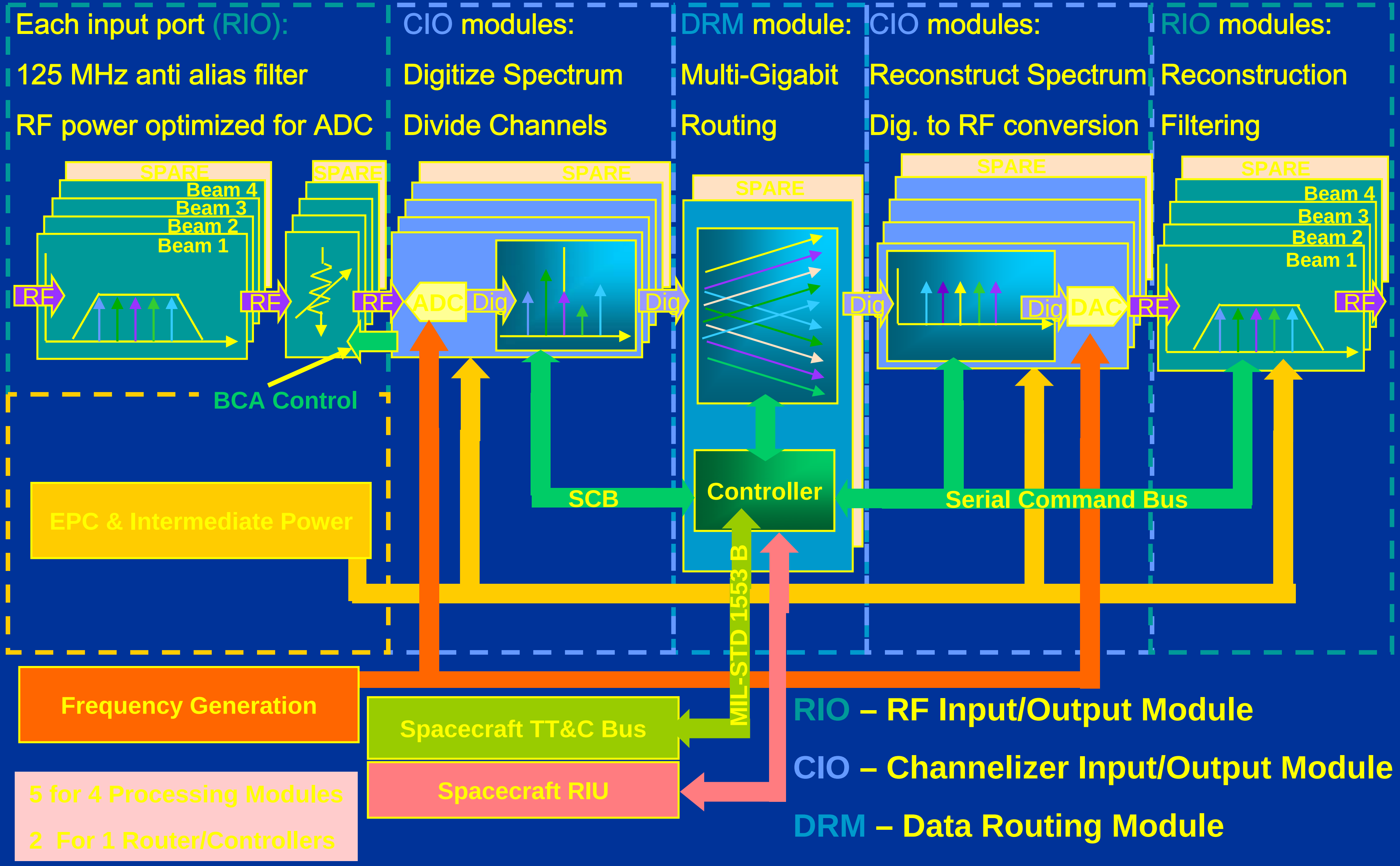
- Early on in the program, it was a complete unknown what this system would look like.
- Requirements from numerous programs were flying around, and they were often incompatible with each other, particular their bandplans:
    - AMC: 4, 8, 36, and 72 MHz wide channels.
    - MMSI: 24 and 32 MHz wide channels.
    - JSAT: 27 MHz wide channels.
    - AsiaSAT: 8 and 36 MHZ wide channels.
    - and many others…
- These unknown requirements not only made it very difficult to design the hardware, but put a tremendous amount of work on the DSP engineer designing algorithms.
    - As the lead DSP engineer, I found myself re-writing the algorithms almost daily, in order to demonstrate the various bandplans.

The Hardware Solution

Digital Channelizer Unit (DCU):

- A modular, space-worthy payload composed of the following building blocks:
- **CIO – Channelizer I/O Module**
  - Composed of 5 Field Programmable Gate Arrays (FPGAs), three of which are voted for extremely high reliability.
  - Handles all of the signal processing, real-time.
- **DRM – Data Routing Module**
  - Multi-gigabit, non-blocking data routing in rad tolerant FPGAs.
- **RIO – RF Input/Output Modules**
- **Power Converters**

System Level DCU Architecture

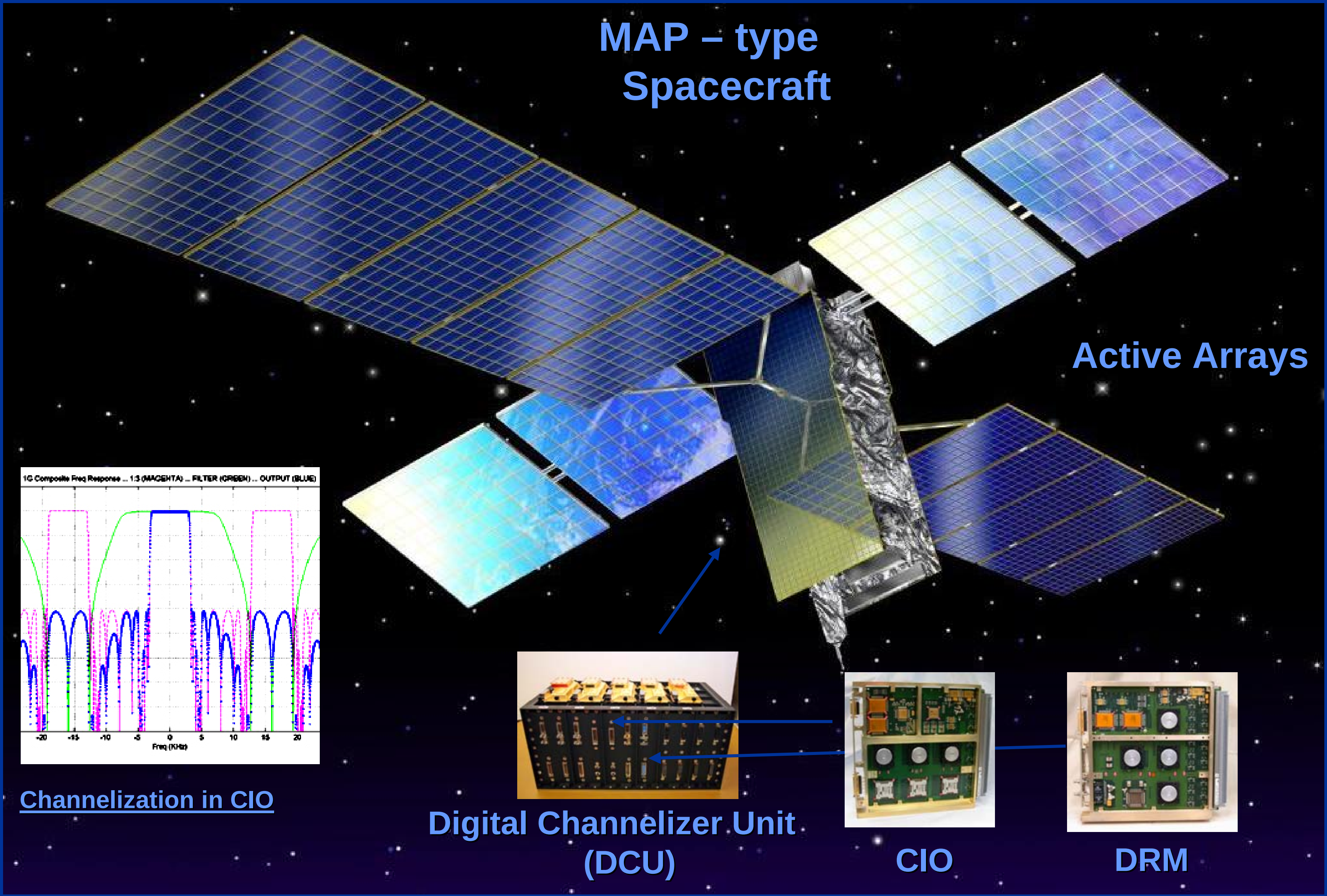


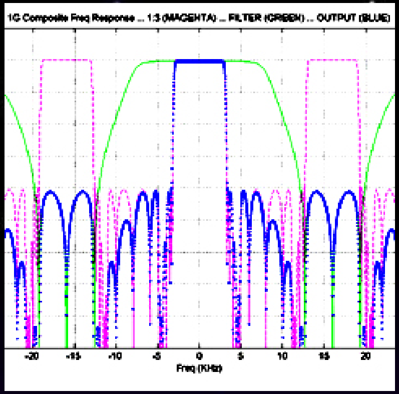
RIO – RF Input/Output Module

CIO – Channelizer Input/Output Module

DRM – Data Routing Module

DCU on MAP Spacecraft





Channelization in CIO

Digital Channelizer Unit (DCU)

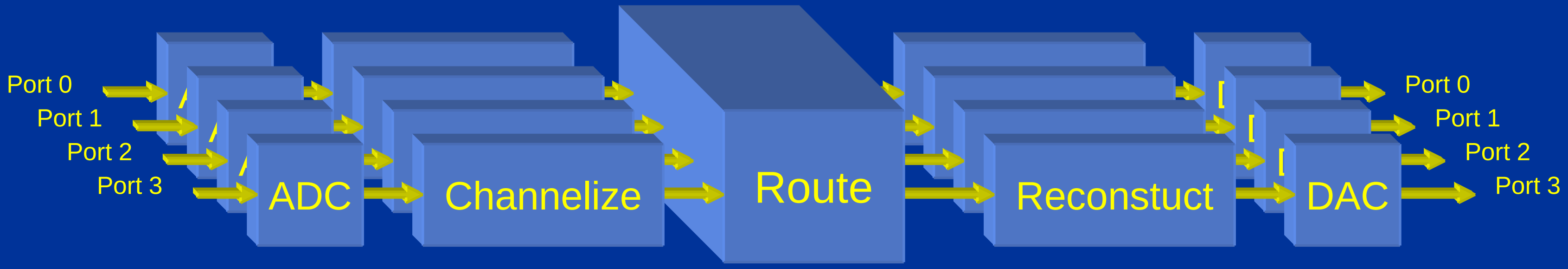
CIO

DRM

DCU Algorithm

Algorithm Overview

- The *DCU algorithm* is defined as the mathematical manipulation (via fixed point arithmetic implemented in the DCU FPGAs) of data samples to perform the following tasks:
  - Process a total digitized analog frequency bandwidth of 500 MHz (which may or may not be contiguous).
  - Break up this bandwidth into narrowband “subchannels” (i.e., “channelize”)
  - Have the ability to re-arrange the subchannels in any arbitrary order.
  - Have the ability to alter the subchannel gains arbitrarily, in multiple modes.
  - Have the ability to measure and report the average and peak signal power level at the input and output of the DCU, as well as at the subchannel level.
  - Recombine the subchannels (i.e., "reconstruct" them) into a composite bandwidth of the same size as the input.
  - Transform the recombined spectrum so that it can be converted back into the analog domain for further processing.
  - Be arbitrarily reconfigurable in signal processing function.
- The functional data flow for a 4 port DCU is depicted here:



DCU Algorithm

Matlab Algorithmic Model

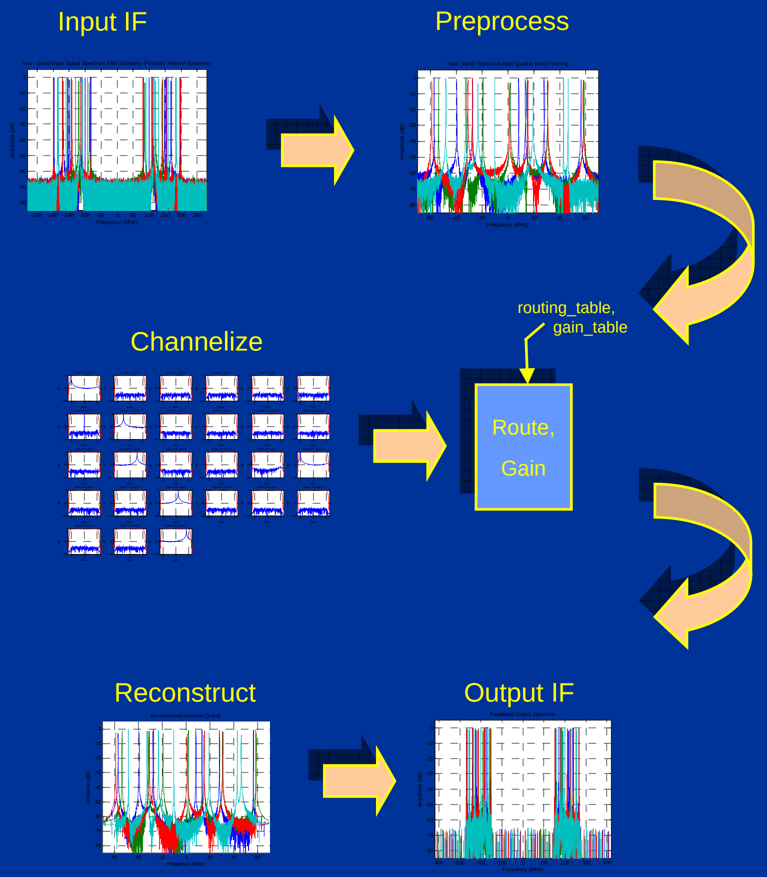
- A *floating point* model of the DCU algorithm has been developed in the *Matlab* environment.

- This model is designed to be *very* flexible, allowing the user to specify a wide range of arguments.

    - This makes the Matlab model useful for functionally verifying various channelization schemes quickly and efficiently. The user simply needs to pick the right parameters (filter coefficients, FFT length, decimation/interpolation factors), etc.

- The DCU algorithm function can be called from the Matlab command line, or as part of a script file.

- This function is composed of lower level functions that do arbitrary channelization and reconstruction.



DCU Algorithm

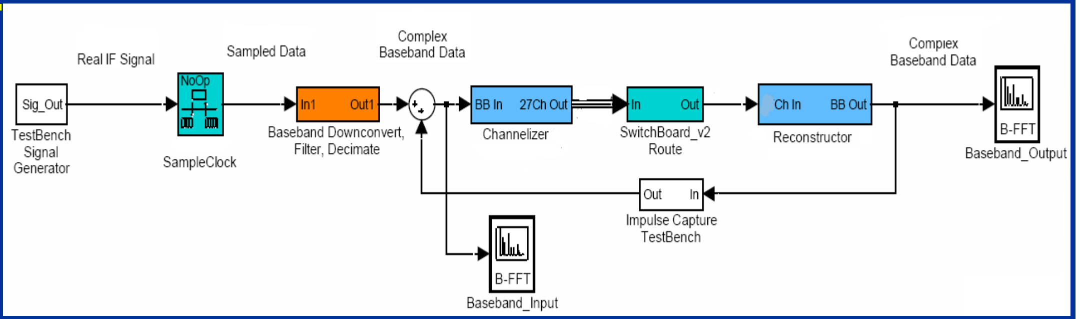
Simulink Structural Model

The next step in the development was to build a *structural* model of the DCU algorithm in the *Simulink* environment.

- This model is designed to test implementation assumptions about the DCU Algorithm and provide a model that is very similar to what the real gateware will look like.

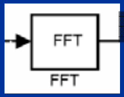
Once the Simulink model works properly, another Simulink model is developed by replacing the blocks with components from another vendor's blockset in order to generate VHDL.

Simulink Block Diagram

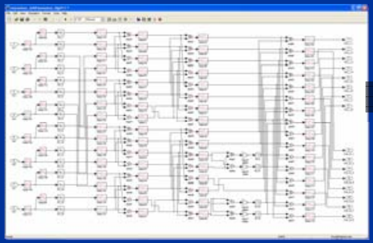


Simulink to Hardware Development

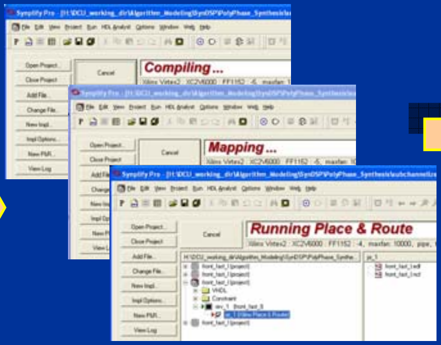
Simulink Model



FPGA SynDSP Blockset Model



HDL Synthesis



DCU Algorithm

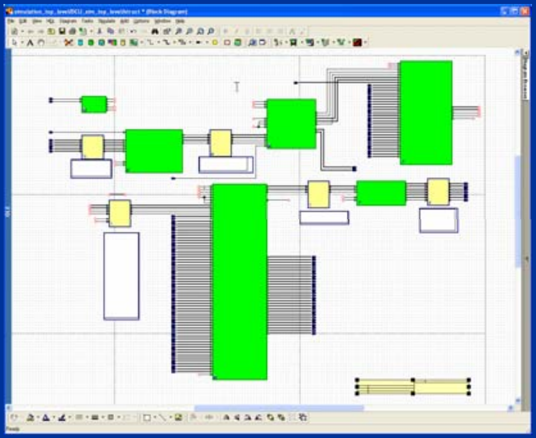
Model Verification

Once VHDL for the various pieces of the DCU algorithm have been developed, tested in the Simulink environment, and stitched together in the Mentor HDL environment, the next step is functional simulation and verification.

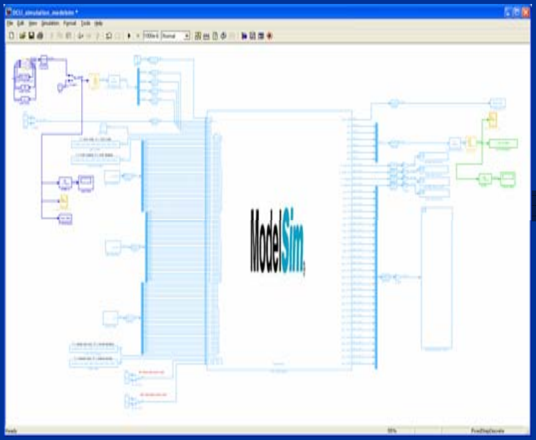
For this task, we choose the EDA Simulator Link MQ tool (formerly known as "Link for Modelsim").

- This tool allows us to make use of all of the signal generation and visualization tools available to us in the Simulink environment, while still simulating the VHDL at the bit level.
- In fact, we can use the exact same test-benches for VHDL test and verification that we used during the development phase.
- This tool was immensely helpful in finding bugs in the code that were not apparent in the Simulink environment alone.

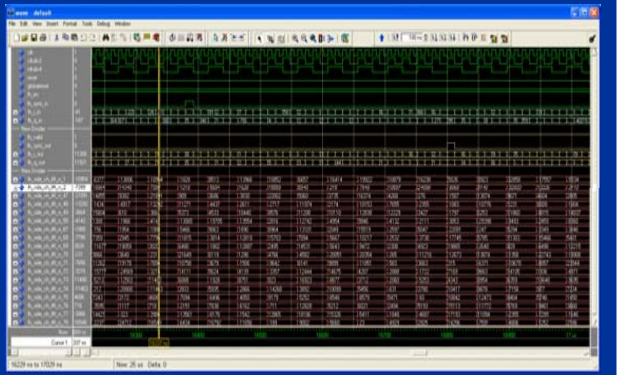
Pull the generated code together with HDL Designer



Bring into Simulink Environment



Simulate with ModelSim



Conclusion

- Using advanced tools and methods, we took a program from concept through functional, space-worthy hardware in about 1½ years.
    - Estimated reduction in development time 8 months over traditional (hand coding) methods.
- Process flow and tools:
    - **Concept** – Matlab (Signal Processing Toolbox)
    - **Structural Model** – Simulink (Signal Processing and Communications Blockset)
    - **VHDL Model** – SynDSP Blockset in Simulink Environment
    - **VHDL Code Integration** – HDL Designer
    - **Functional Verification and Degugging** - EDA Simulator Link MQ tool in Simulink Environment