

SPLASH: Single-chip Planetary Low-power ASIC Spectrometer with High-resolution

Rachel Hochman



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-230

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-230.html>

December 19, 2014

Copyright © 2014, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

SPLASH: Single-chip Planetary Low-power ASIC Spectrometer with High-resolution

by Rachel Hochman

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Borivoje Nikolic

Research Advisor

Date

* * * * *

Vladimir Stojanovic

Second Reader

Date

SPLASH Report

December 19, 2014

Contents

1	Introduction	7
1.1	Motivation	7
1.2	Prior Work	7
1.3	Goals of Project	8
1.4	Scientific Significance	9
1.5	Outline of Document	10
2	Spectrometer Design	12
2.1	ADC	12
2.1.1	TISAR Architecture	12
2.1.2	SPLASH ADC Block	13
2.2	DSP Design	15
2.2.1	PFB and FFT	15
2.2.2	Power, Bypass MUX, and Vector Accumulation	17
2.2.3	Test Controller	18
2.2.4	Verification in Simulink	19
2.2.5	Verification on FPGA	20
3	Design Flow	21
3.1	ASIC Design Flow	21
3.2	Simulink Design Process	21
3.3	Chip-In-A-Day Design Flow	22

4	Implementation and Testing	24
4.1	Die Photo	24
4.2	Test Setup	25
4.3	PCB Design	25
4.4	Test Software	25
5	Conclusion and Future Work	27
6	Appendix 1- Full 'SPLASH-in-a-day' Guide	28
6.1	Getting Started	28
6.2	IP Generation	29
6.2.1	Setup	29
6.2.2	Usage	29
6.3	RTL Generation	30
6.3.1	Setup	30
6.3.2	Usage	30
6.3.3	Memory Replacement	30
6.4	Synthesis	31
6.4.1	Inputs	31
6.4.2	Usage	31
6.4.3	Outputs	31
6.5	Place and Route (IC Compiler)	31
6.5.1	Setup	31
6.5.2	Constraints and Floorplanning	32
6.5.3	Place and Route	32
6.6	Finishing	33
6.6.1	Export to Cadence	33
6.6.2	DRC	34
6.6.3	LVS	34
7	Appendix 2 - Data Sheet	35

List of Figures

1	Die Photo of Mars Spec. 2.8x2.8mm	8
2	Measurements Made by NASA's Upper Atmosphere Research Satellite	11
3	Block Diagram of TISAR ADC	13
4	Die Photo of TISAR ADC Analog core area: 0.4 x 0.45mm ²	13
5	SPLASH ADC Block Diagram	14
6	Digital Clock Generation	14
7	Block Diagram of DSP Core	15
8	Example of DFT Leakage	16
9	Single Bin Response of PFB vs FFT	16
10	PFB FIR Implementation	17
11	Bit Widths Through DSP	18
12	FFT Block Diagram	18
13	Scan/Chain Test Controller	19
14	DSP Functional Simulation Pre-Synthesis	20
15	Single Bin Output	20
16	PFB + FFT Output	20
17	Design Flow	22
18	Die Photo of SPLASH	24
19	Block Diagram of Shuttle LX1 [4]	25
20	Layout of SPLASH Carrier Board	26

List of Tables

1	Key Specs	35
2	Test Controller Modes	35
3	Test Controller Instruction Descriptions	36
4	Test Controller Registers	36
5	Force Block Control Programming	36
6	Bottom Side Pinout	37
7	Right Side Pinout	38
8	Top Side Pinout	39
9	Left Side Pinout	40

Acknowledgements

First of all, I would like to thank my advisor Bora Nikolic and my co-advisor Dan Werthimer; they have guided and mentored me throughout this entire project. Furthermore, I must thank Luis Esteban-Hernandez and Brian Richards, without whom SPLASH would not have been made. Juan Antonio Lopez Martin also contributed to the design. I am grateful to all of the CASPER collaborators who have created the DSP libraries in Simulink (especially Mark Wagner), the BWRC staff and students who worked on the Raven project (in particular Stevo Bailey and Brian Zimmer), the Radio Astronomy Lab engineers who designed the SPLASH PCB (Matt Dexter and Calvin Cheng), and our JPL collaborators (especially Bob Jarnot). In addition, thanks to my other reader, Vladimir Stojanovic. Finally, none of this work would be possible without the financial support of NASA, the EECS department, ST Microelectronics, and the Marie Curie Fellowship Program.

As for the people who are always there for me, thanks to Sue and Roger, Adrienne, and Chris.

1 Introduction

1.1 Motivation

Spectrometers are one of the most useful instruments available to space scientists. Over the past several years NASA's Outer Planets Assessment Group (OPAG) has detailed several desired science goals and objectives to be met by a future flagship mission to Jupiter or its moons, as well as investigations and measurements that will fulfill these objectives. Many of these required measurements can be made with a wideband submillimeter spectrometer. Two examples of investigations that can be conducted on Jupiter by such a spectrometer are the study of the stratosphere, which links the troposphere to the upper atmosphere, where effects of external supplies of energy through radiation and particle exchange are dominant, and the characterization of the physical and chemical properties of Jupiter's clouds and the processes that maintain them. Further applications might be the study of other planets such as Saturn or Venus, exospheres around satellites and asteroids, and cometary comae. Therefore a NASA Planetary Instrument Definition and Development (NPIDD) project was defined, for which UC Berkeley and the Jet Propulsion Lab (JPL) have developed an ASIC spectrometer designed specifically to provide an essential building block for a space instrument that can address fundamental issues about the chemistry, evolution, and dynamics of planetary atmospheres. This spectrometer chip has been named SPLASH, for Single-chip Planetary Low-power ASIC Spectrometer with High-resolution.

1.2 Prior Work

At it's most basic, a spectrometer instrument consists of a receiver, an analog to digital converter(ADC), and digital signal processing (DSP) that performs a fast fourier transform (FFT). The FFT takes in time domain data and returns frequency domain data, or a spectrum. Previous spectrometer instruments have been made with separate receivers, ADCs, and DSP engines. In the past FPGAs have been used for the DSP block rather than ASICs. An FPGA is a field programmable gate array, which is an integrated circuit that contains programmable logic blocks and reconfigurable interconnect. This programmability makes the development time much less than that of an ASIC (application specific integrated circuit). However, for high bandwidth applications, FPGAs are less suitable, because they are power hungry and extremely sensitive to errors caused by high-energy particles. In addition, currently available radiation tolerant FPGAs lack the resources for a large, high-bandwidth spectrometer. Before 2008, state-of-the-art NASA spectrometers included an acousto-optical spectrometer with a bandwidth of 1.5GHz but only 1000 channels and 5.5W of power consumption. A chirp transform spectrometer had 4096 channels but only 800MHz of bandwidth

and consumed 10W. Therefore, in 2008, students and staff at UC Berkeley's Space Sciences Lab (SSL) and the Berkeley Wireless Research Center (BWRC) designed the 'Mars Spec' ASIC, a precursor to SPLASH. This chip was fabricated in ST 90nm technology, is 7.84mm^2 and operates with clock rates up to 390MHz, delivering a throughput of up to 1.56GS/s with 710 mW of power. The Mars spectrometer receives data from an off-chip 1.5GS/s ADC that produces 4 parallel 8-bit streams at 375MS/s. An 8K FFT was performed to produce a spectral resolution of $1.5\text{GHz}/8192 = 183\text{KHz}$ per bin, which is sufficient for studying mm- and sub mm-thermal emissions from the atmospheres surrounding Earth, Venus, and Mars [2]. Figure 1 shows a die photo of the Mars Spectrometer ASIC. This ASIC requires an external power hungry ADC (2 W) and has half the bandwidth of the SPLASH chip.

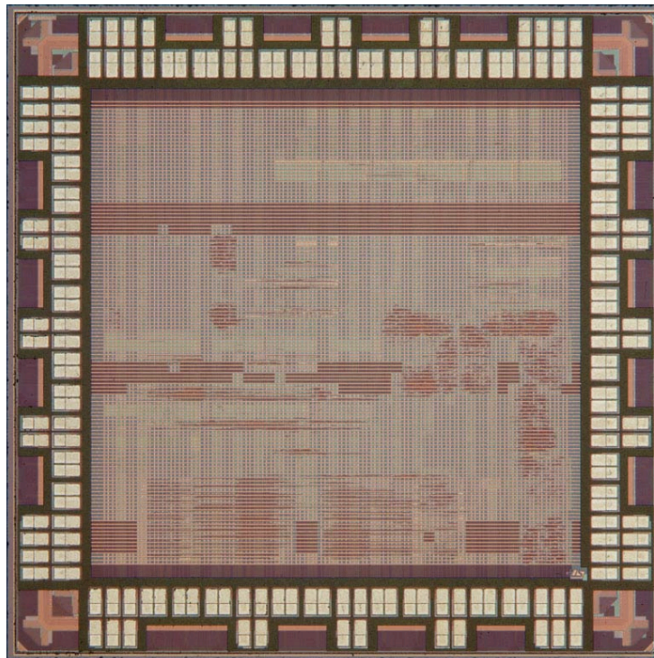


Figure 1: **Die Photo of Mars Spec. 2.8x2.8mm**

1.3 Goals of Project

The stated goals of the project were to design and test a high bandwidth (1.5 GHz) and high resolution (183 kHz) single chip (mixed signal ASIC) digital spectrometer. One of the chief innovation goals of this rad-hard spectrometer was to combine a 3GS/s ADC and high bandwidth DSP functions of previous implementations, with unprecedented low power consumption, and excellent channel shapes (polyphase FFT implementation). Its low mass, size and power footprint will allow the spectrometers of future microwave sounders to be implemented within the IF subsystems, with significant savings in cost, mass, power and volume. One

of the proposed benefits of this ASIC is that it is appropriate for immediate application to several flight missions; for a typical planetary mission, this technology can yield up to 10 watts of savings on a 30 watt instrument. The key to this power savings is the utilization of the mixed signal ASIC, since several watts are normally required for line driver circuitry to interconnect high speed chips. The power savings, is, in turn, highly desirable on a spacecraft which has extremely strict weight and power budgets. In addition to this new spectrometer technology, recent advances in submm-wave receivers such as: compact light weight mixers, efficient multiplied THz local oscillators, low noise MMIC amplifiers and THz TACIT mixers, will make future microwave radiometer instruments more competitive due to the savings in weight and power. The improved precision of these radiometers, combined with the high bandwidth and excellent resolution of the digital spectrometers, will also result in improved science data quality, strengthening proposals for such instruments. In addition, the chip was designed to be resilient to soft-upsets, which happens when a high energy particle strikes the chip and flips a stored bit. This is an especially important feature due to the high radiation environment surrounding Jupiter.

The ASIC based, radiation-tolerant spectrometer described here will advance the state of the art for digital, high resolution spectrometers designed for space applications. These spectrometers can be expected to provide significant scientific advances to studies of planets, asteroids, and comets, when used in conjunction with a heterodyne mixer receiver designed for planetary spacecraft. In summary, the new technology we have developed for this program provides the following characteristics for digital space spectrometers: 1) first radiation tolerant design needed for studies of the giant planets (e. g. Jupiter) and satellites, 2) low mass and power needed for outer planet and space applications in general, 3) broad bandwidth, very high spectral resolution with excellent band pass shape design, 4) ease of testing, calibration and use in space and 5) flexible programming to suit specific applications. These attributes will all contribute to the wide use of these devices.

1.4 Scientific Significance

Spectrometers measure the properties of light over a portion of the electromagnetic spectrum, specifically intensity, which can be used in studying the chemical makeup of atmospheres. This is possible because of the nature of polar atmospheric molecules, which exist in specific states. Quantized energy is stored in three modes: rotational, vibrational, and electric energy. When the molecule changes from a higher energy state to a lower energy state it releases electromagnetic radiation. Changes in rotational energy lead to microwave emissions, rotational and vibrational together give off infrared light, and changes in all three modes emit ultraviolet light. Quantum mechanical models of molecules predict the specific frequency it

emits, and therefore, the intensity of light in certain frequency ranges corresponds with radiation emitted by certain polar molecules. These spectral 'lines' are also influenced by the temperature and pressure of the molecule's environment. Planetary scientists, in particular, benefit greatly from data gathered by spectrometers, because they can determine atmospheric composition by examining the emission spectra produced. Which lines match up with which molecules and environments has been well cataloged and documented in various 'Spectral Line Catalogs' including the one found at <http://spec.jpl.nasa.gov>, which is a computer-accessible catalog of submillimeter, millimeter, and microwave spectral lines in the frequency range between 0 and 10,000 GHz.

The study of planetary atmospheres can tell us many things. One current satellite project is studying the role that loss of volatiles from the Mars atmosphere to space has played through time, giving insight into the history of Mars' atmosphere and climate, liquid water, and planetary habitability [7]. Learning about the evolution of atmospheres on other planets can provide insight into how our own might evolve. One of the intended uses of SPLASH is to study the atmosphere of Europa, a moon orbiting Jupiter, specifically looking for signs of water or life. Clearly, a large and diverse amount of information can be discovered through spectroscopy. One of the most important experiments concerning our own planet resulted in the image in Figure 2, made from data gathered by a Microwave Limb Sounder (a spectrometer that "looks" tangentially at the edge of the atmosphere, rather than a nadir sounder, which looks directly down through the atmosphere at the Earth's surface).

These results gave the definitive proof that industrial chemicals in our atmosphere are perfectly correlated to the hole in the ozone layer. Such concrete information has led to international actions to slow depletion of the ozone layer, and similarly convincing data may be able to spur future enactment of environmental policy.

1.5 Outline of Document

The rest of this document describes the project as follows: The first section is a brief discussion about the ADC used on this chip, how it works, and how it was integrated into the spectrometer. This is followed by a more in-depth discussion about the DSP core of the chip including to what specifications it was designed, the main blocks that make it up, and the verification done in Simulink and on an FPGA. Next there is a description the tool flow of putting the entire chip together, along with the associated benefits and challenges. An appendix contains full instructions on how to make a 'chip-in-a-day' starting from the Simulink and verilog descriptions. Also included is a die photo of the fabricated chip. The testing plan for the chip is then described with information on the chip carrier board, the testing environment, and test

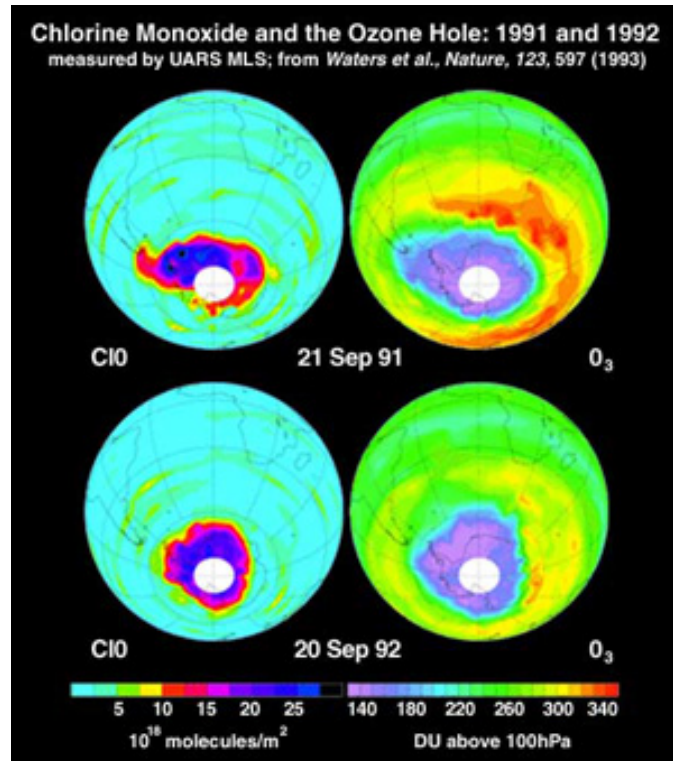


Figure 2: Measurements Made by NASA's Upper Atmosphere Research Satellite

software. Finally, in another appendix, the data sheet is shown as it will be given to the engineers putting together future radiometer instruments with this chip as a key component.

2 Spectrometer Design

2.1 ADC

The ADC implemented on the SPLASH chip was ported from a stand alone design [1]. It is a time-interleaved successive approximation register (TISAR) ADC. This ADC was chosen in part because it was designed in ST 65nm, meaning we could reuse large portions of the analog layout. In addition, the ADC has 8.16 effective bits when running at 2.8GS/s and consumes only 44.6mW of power. For the purposes of this chip, the design was improved for $f_s > 3$ GS/s. This speed, precision, and energy efficiency were ideal to meet the data conversion needs for SPLASH. Below is an explanation of the ADC architecture and information on how the ADC portion of SPLASH was put together. Figure 3 and Figure 4 are taken from the VLSI paper about this ADC [1].

2.1.1 TISAR Architecture

A high-level block diagram of the ADC is shown in Figure 3. It consists of $M=24$ time-interleaved channels, with two additional channels used for calibration. Each channel is split in two parts: analog (SARx_A, which also includes SAR logic), and digital (SARx_D). The analog part consists of the capacitive DAC and the comparator. The digital part finds the final output by summing the weighted output bits from the analog part with the respective channel offsets. The values of the digital weight coefficients and offsets are adaptive and are iteratively calculated by a least-mean-squares (LMS) algorithm. This calculation takes place in the linearity LMS block. There is also a 'timing LMS' block, which calculates timing mismatches and tunes the delay elements, Δt . The raw output of each channel is 11 bits with a radix of 1.85. A reduced radix was used to enable mismatch correction in the digital domain. The subsequent filter block converts to the data to radix 2.

The ADC has a scan chain, into which you can input initial guesses for the various weight coefficients and control the calibration procedure. Calibration type and timing calibration are set by bits written into the scan chain first and then to a register at the edge of an external signal. Radix and offset calibration are always enabled. The timing calibration is also always running although the tuning in the analog domain can be disabled with the timing calibration enable bit.

Figure 4 shows the die photo of the TISAR chip. The analog core area occupies $.18\text{mm}^2$ and the other half of the core area is made up of the digital calibration logic and the memory that was used to store and read out values. We only re-used the analog core roughly as is; the memory block was not re-used at all in SPLASH and the digital logic was redesigned.

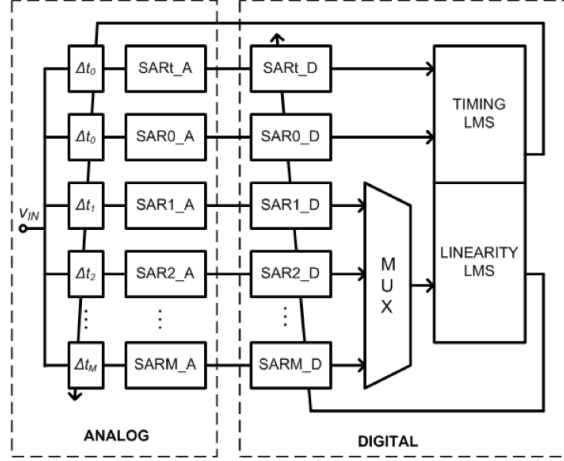


Figure 3: **Block Diagram of TISAR ADC**

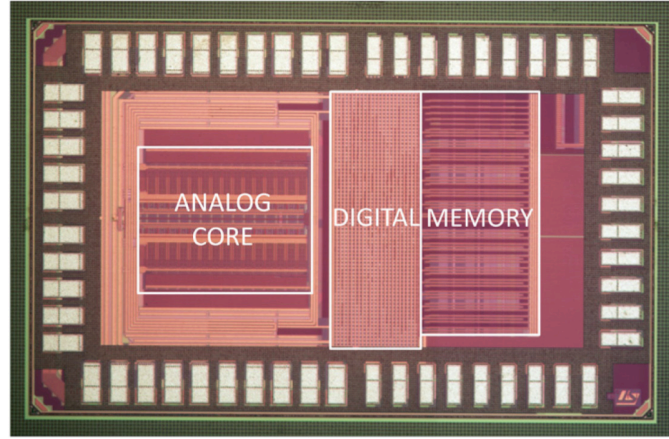


Figure 4: **Die Photo of TISAR ADC**
Analog core area: $0.4 \times 0.45\text{mm}^2$

2.1.2 SPLASH ADC Block

In order to integrate the TISAR with the SPLASH digital section, we made many additions, specifically to the digital logic portion of the ADC. Figure 5 shows a block diagram of the ADC within the SPLASH chip. The analog core and the filter block remained mostly unchanged. However, a filter select block was added in order to output either the 'filtered' (calibrated) outputs or the raw ADC outputs. This feature was added to aid in debugging. The 'MUX' block performs two important functions. The first is actually a demuxing of the data. Since the data comes out of the ADC in 24 streams, but the DSP operates on 8 streams of data, the 24 streams had to be demuxed into 8. The other function of the MUX block is to create the digital clock.

A 3GHz clock is input into the ADC, and the clock circuitry divides this down so that each channel

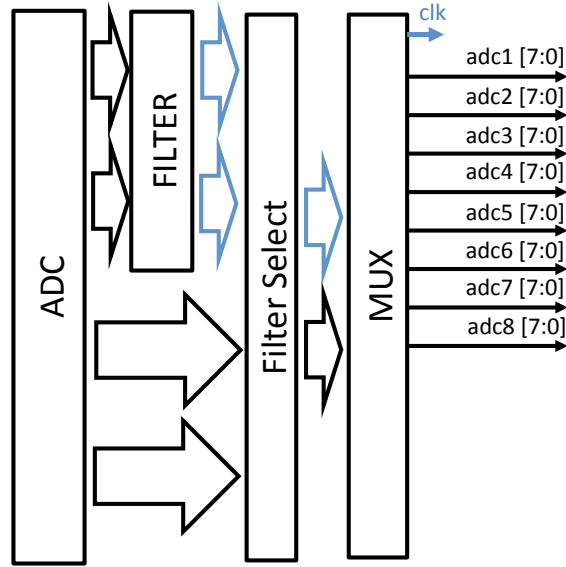


Figure 5: **SPLASH ADC Block Diagram**

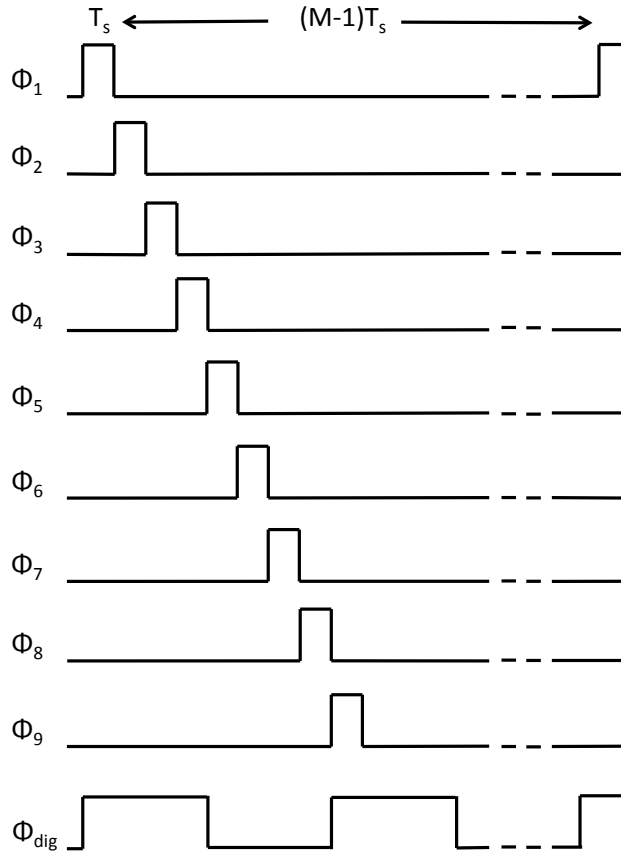


Figure 6: **Digital Clock Generation**

samples with a frequency of f_s/M and a duty cycle of $1/M$. The clocks for each channel are equally phase shifted. The clock needed for the DSP, however, is equivalent to $f_s/8$. The MUX block therefore generates this clock based on the rising edge of every 4th channel clock (1, 5, 9, etc.) , as shown in Figure 6. The outputs of the entire ADC block are the digital clock and the 8 streams of ADC data (either calibrated or uncalibrated).

2.2 DSP Design

The complete DSP core was described in Simulink using PFB filter, FFT, detection and accumulator blocks designed by the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER). A block diagram in Figure 7 outlines the main blocks as well as the data flow. The blocks are each explained in detail below.

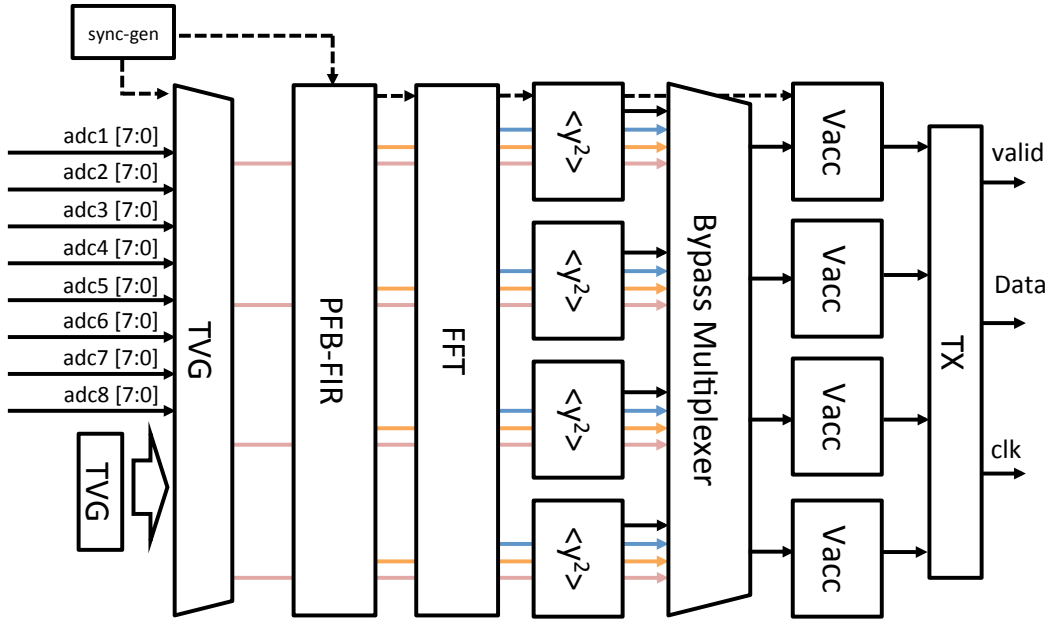


Figure 7: Block Diagram of DSP Core

2.2.1 PFB and FFT

The primary function of any spectrometer is Fourier analysis, converting time-domain data into frequency-domain data with a Discrete Fourier Transform (DFT). However, there is one main drawback to a straightforward application of the DFT: leakage.

Because a DFT operates on a *finite* length of samples, the frequency domain response of a complex sinusoidal waveform using the DFT is the convolution of the Fourier transform of the sinusoid and that of

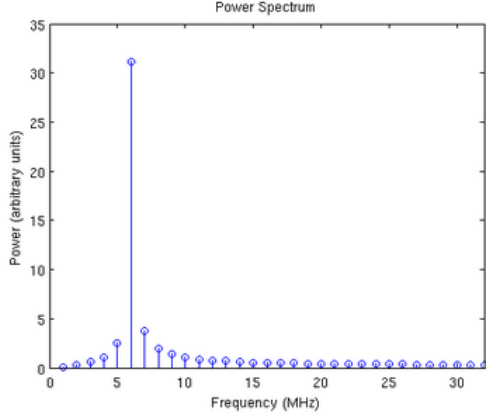


Figure 8: **Example of DFT Leakage**

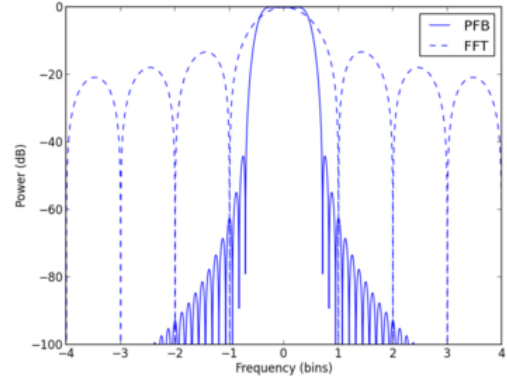


Figure 9: **Single Bin Response of PFB vs FFT**

a rectangular window. Since the Fourier transform of the complex sinusoid is a shifted delta function, the result of the convolution is the Fourier transform of the window - a sinc function - centered at the location of the delta function. Although they may line up perfectly for specific inputs and sample lengths, in general, the frequency domain bin centers fall at non-zero locations on the sinc function. Therefore a single tone appears with some level in all the frequency bins of the DFT output. Since the energy contained in the input frequency bin 'leaks' into other frequency bins, this effect is called DFT leakage. In certain use cases this effect may be insignificant, but in astronomy, for example, the leakage from a strong radio frequency interference (RFI) signal can drown out astronomical signals of interest in the nearby bins. An example of this is shown in Figure 8; a tone at 5.1MHz, sampled at 128MHz, and Fourier-transformed with 64 points, appears to varying levels in all the output frequency bins [3] .

The polyphase filter bank (PFB) technique is a way to mitigate the negative effects of the straightforward DFT. A comparison of the single bin response of both techniques is shown in Figure 9 [3]. The PFB results in a flat rather than rounded response across the channel. It also lessens the leakage effect by providing excellent suppression of the side-lobes of the the sinc function by changing the single-bin frequency response of the DFT to approximate a rectangular function. Instead of taking a direct N-point DFT, a block of data of size $N \times P = M$ is first multiplied point-by-point with a window function. This windowing is effectively a filtering process; the elements of the window function act as the filter coefficients in a finite impulse response filter (FIR). By the nature of the DFT, the shape of the single-bin frequency response is determined by the shape of the window function. Therefore, to force the single-bin frequency response to resemble a rectangular function as much as possible, we use its Fourier transform pair, the sinc function, as the window function. After the windowing/weighting, the block of data is split into P subsets (each of length N) and added point-by-point. Figure 10 shows the polyphase filter bank implementation with $P = 4$ taps and N sub-filters. The

commutator at the left rotates in the clockwise direction, and makes one complete rotation in the duration of one unit delay. The resulting array, here denoted by $y(n)$, then goes through an N -point DFT implemented as an FFT. The 8 bit input data grows to 18 bits in the FIR. All bit widths though the design can be seen in Figure 11.

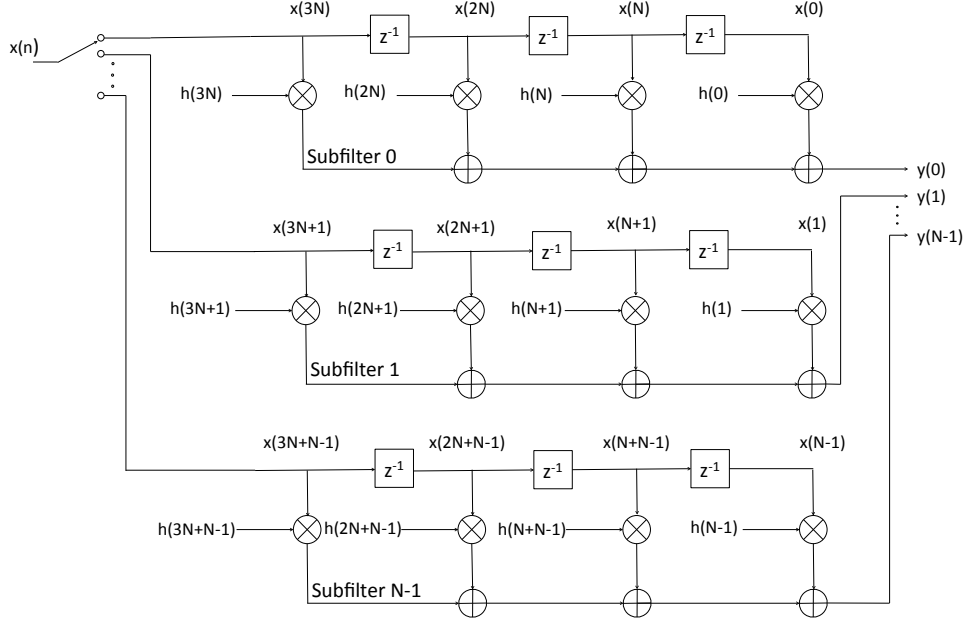


Figure 10: **PFB FIR Implementation**

The N -point FFT (for SPLASH, $N = 16k$) is computed with a biplex FFT followed by a direct FFT. A block diagram is shown in Figure 12. The biplex FFT is a pipelined FFT implementation that reduces the number of complex multipliers and adders needed for a given FFT size, fully utilizes the hardware, and has distributed memory. It achieves this by realizing that only $N/2$ points need to be stored before processing begins since the first butterfly operates on sample 0 and $N/2$. Each subsequent stage needs only to store half the data of the previous stage. There are two 8K biplex FFT blocks. The direct-form FFT produces the 16K FFT result as eight streams of 2K packets coming from the biplex cores. The outputs of this decimation-in-frequency FFT are scrambled and must be reordered in software.

2.2.2 Power, Bypass MUX, and Vector Accumulation

The power of the complex streams is computed after the FFT by squaring the real and imaginary parts and adding them together. Next in the flow is a bypass multiplexer that chooses between the outputs of the ADC, PFB, FFT, and power blocks. This will be used for debugging and can be set via the scan chain as described in the next section.

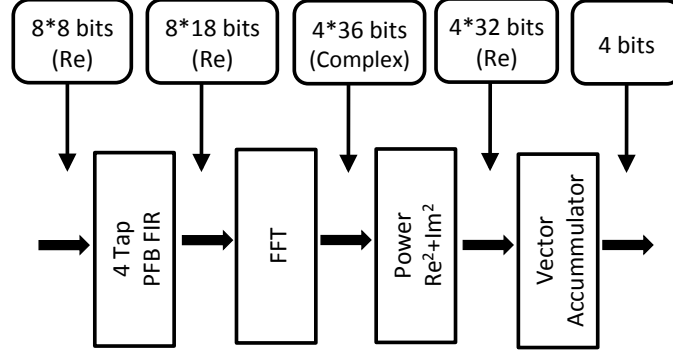


Figure 11: **Bit Widths Through DSP**

Finally, the results are accumulated in the vector-accumulator (vacc) block for a programmable number of packets. The vector-accumulator is an adder with feedback from a synchronous delay line, which sums corresponding frequency bins from adjacent packets. Therefore, the frequency-domain data can be averaged over a period from 10ms to several seconds to trade-off the spectrum update time for improved SNR. This accumulation length is programmable via the test controller described below. The user can choose any length in multiples of 2048 spectra, up to 1.048 million spectra. A transmitter block sends the data out in 4bits, with a clock and valid signal.

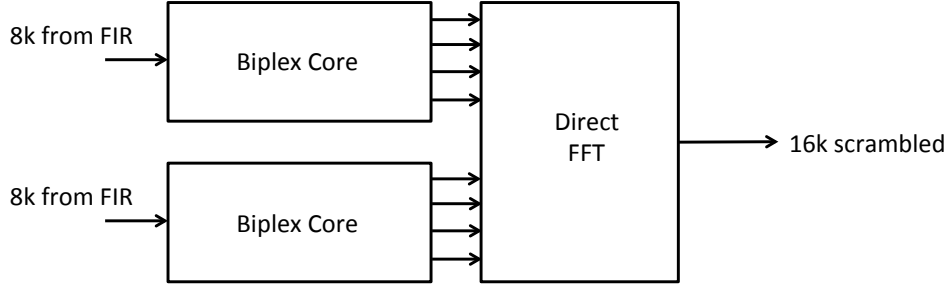


Figure 12: **FFT Block Diagram**

2.2.3 Test Controller

SPLASH includes dedicated datapath elements for ease of testing and to support run-time soft-upset detection. The chip can run in two modes, hard and soft. In the hard mode, operation does not depend on any state registers; this prevents failures caused by soft upsets from high-energy particles. In the soft mode, some control inputs are re-defined to support internal testing modes. A block diagram of the scan chain and test controller is seen in Figure 13. There are three 'force' registers, which are protected by triple-voting logic and therefore against soft-upsets. The settings in these 'force-registers' are the accumulation length, FFT

shift schedule, and the control of the bypass multiplexer. This MUX chooses which data goes into the vector accumulator, thereby allowing the user to view intermediate datapath results. A snapshot can be uploaded from the accumulator through the serial interface.

Additionally, multiple-input shift registers (MISRs) are included to monitor streams of data at predetermined points between key function blocks. These registers can generate unique signatures from streams of data during a preprogrammed interval. When combined with the internal test-vector generator, the same signature should recur unless a soft-upset changes one or more of the data values, effectively turning the entire spectrometer core area into a soft-upset detector. The MISR can also capture single-cycle data values to assist with debugging. The scan chain total has 266 bits to be programmed and 501 bits in the MISRs. To program the 266 configuration bits, the user must turn the serial_program bit high while inputting the 266 bits. Once this is done, the user must set the serial_force bit high. This will force the bits into the correct registers (OFFLOAD), and the test controller will push the data into the force blocks. Specific bit locations are described in Appendix 2.

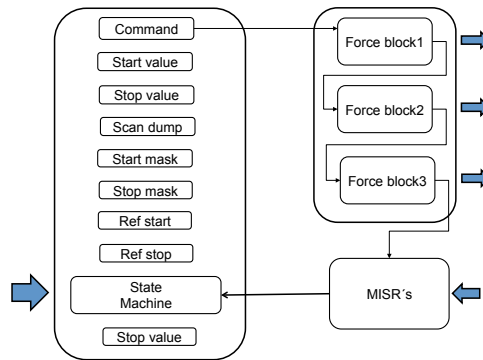


Figure 13: Scan/Chain Test Controller

2.2.4 Verification in Simulink

The design flow optimizes for differences between FPGA and ASIC implementations by substituting datapath blocks with cycle-accurate and bit-accurate subsystems that may have different micro-architectures. For example, a memory-based synchronous delay line uses dual-port memories that have the same resource cost on an FPGA as a single-port memory. The same function would be inefficient on an ASIC, and is replaced by a single-port SRAM-based circuit that has the same I/O behavior. A Simulink testbench verifies equivalence by co-simulating the FPGA and ASIC implementations side-by-side. The below simulation shows a sinusoidal input, with the tone going into the vector accumulator, and below that the output of the vector accumulator growing until the valid signal goes high.

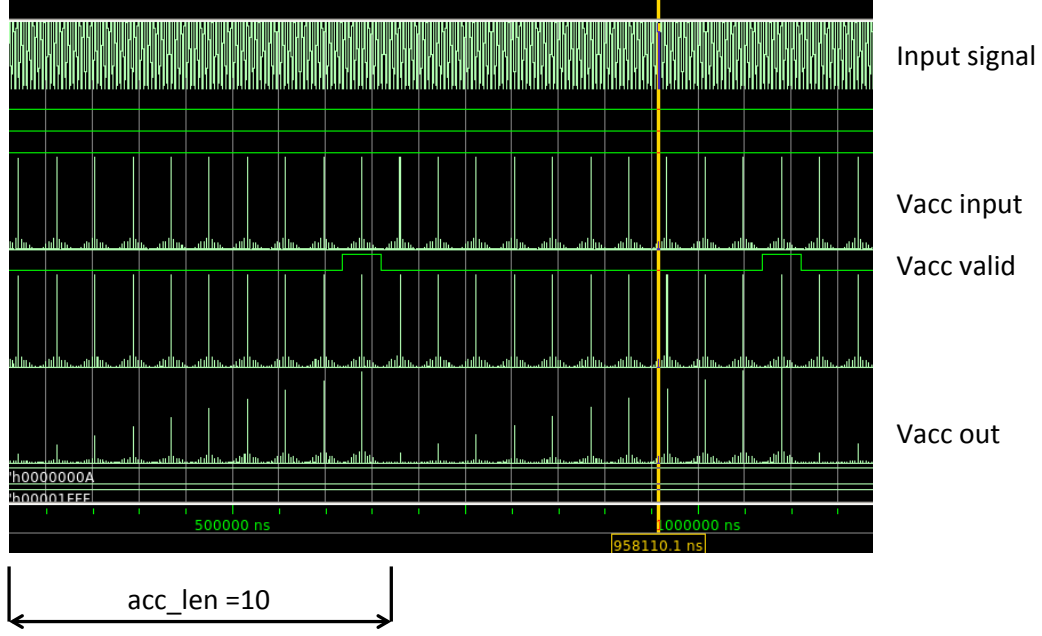


Figure 14: **DSP Functional Simulation Pre-Synthesis**

2.2.5 Verification on FPGA

The system was mapped to a ROACH2 FPGA board [8] and field tested (at a lower clock rate) using a 3 GS/s ADC board that plugs into the ROACH2 board. The plot in Figure 15 shows the results of a single tone test, measuring the output power from a single tone in 700 bins (350 on each side). The plot in Figure 16 shows the results of a sweep of frequencies, with the output powers summed. Because this test was run on the FPGA, these are the outputs of an 8K FFT, where each bin is 48.828KHz wide.

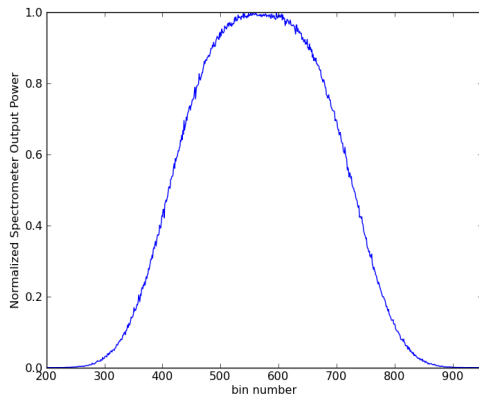


Figure 15: **Single Bin Output**

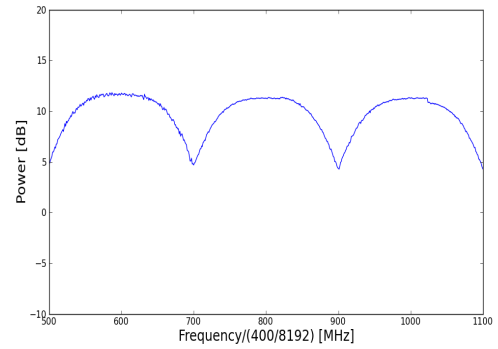


Figure 16: **PFB + FFT Output**

3 Design Flow

3.1 ASIC Design Flow

Traditional approaches to designing and implementing digital signal processing on an ASIC can be complicated, involving multiple phases overseen by multiple designers. Each design description must be translated into the next, and during each translation an opportunity for error arises. Therefore, between each step, additional verification is required to avoid logic errors. Generally, a design begins with an abstract algorithm, the operation of which is verified in a floating-point simulation in a tool like Matlab. This simulation is used to derive system specifications. Next, a system designer will map the system specifications into a system architecture. This is done with a behavioral or structural description. At this point, the floating-point data types must be converted to fixed-point, and the designer must verify that finite word lengths do not alter the functionality of the algorithm. The third step involves implementing the architecture in Verilog or another register-transfer level (RTL code), and again verifying functionality. The Verilog can then be synthesized into standard cell netlists by a tool such as SYNOPSIS Design Compiler. These standard cell net lists are then given to a physical designer, who will use place-and-route tools (such as SYNOPSIS IC Compiler) to create a layout to provide to a fabrication house. During this step the designer needs to verify that all timing constraints are met and of course this translation, like the previous two, necessitates another verification. Throughout the process, opportunities to reduce power consumption and area are often unknown or ignored, and any issues discovered during the physical design phase are unknown to the algorithm designer. This can lead to the need for multiple iterations, which can greatly extend the design time.

Another issue slowing design time and complicating design procedures is the fact that simulation of designs can be extremely lengthy and resource intensive. One way to speed up the simulation time is to map the design onto an FPGA and actually perform hardware emulation. Taking advantage of the extensive performance capacity and software support of FPGAs can greatly speed up design iterations but leads to another issue of FPGA/ASIC microarchitecture incompatibility. Certain primitive components and memory types can be implemented in an FPGA fabric, but not in an ASIC. To get around this issue, an 'in-house' tool was developed, which is further discussed in the following section. Figure 17 contains a diagram of the entire flow, which has been modified from [2].

3.2 Simulink Design Process

The most challenging part of the Simulink design process involves the transition from FPGA oriented Verilog to ASIC specific Verilog. In order to emulate the DSP design in hardware, System Generator, a Xilinx tool,

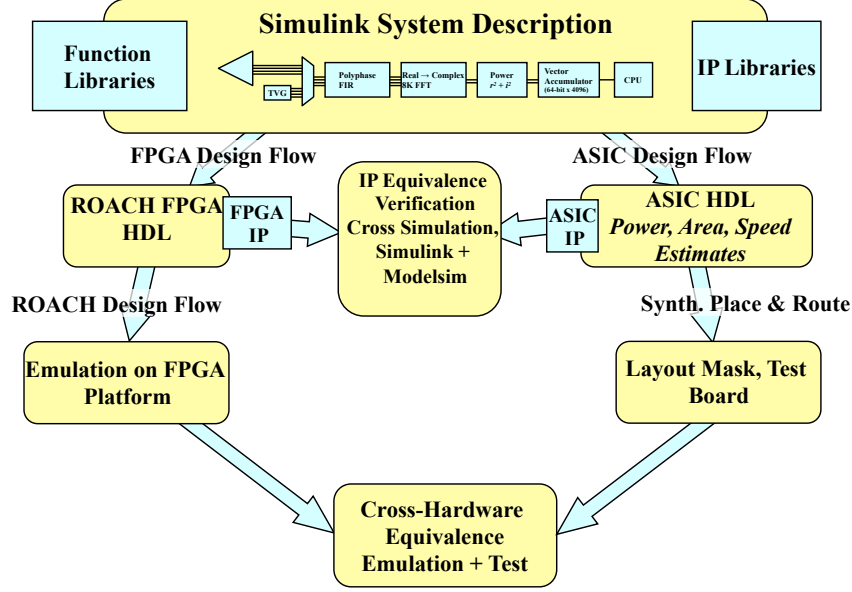


Figure 17: **Design Flow**

was used to convert the Simulink design into VHDL, which could then be implemented on the ROACH2 board. Several iterations of the design were tested on the FPGA before settling on the final design with the desired output bin shape and scalloping. Insecta is the in-house tool used to synthesize basic primitives based on the behavioral RTL, and perform initial top-level synthesis for an ASIC based on a Simulink description. The tool also performs HDL simulation to confirm functional equivalency between the two hardware descriptions [5]. However, Insecta is not equipped to deal with memory blocks, which are implemented as dual-port memories for the FPGA, but must be single-port for the final chip. Therefore, once the design is completed, custom memories must be inserted as blackboxes in the Simulink design, and the entire flow of System Generator and Insecta must be run again. Finally, the results must be simulated in Modelsim against the design with dual-port memories, in order to check that the memory behavior is bit and cycle-accurate. Once this behavior has been proven the results can be passed into the place and route tools.

3.3 Chip-In-A-Day Design Flow

Our tool flow has been designed so that, in theory, one can begin with the ADC IP block and the Simulink DSP design and end up with a physical layout design in one day. This greatly facilitates the revision process by making it possible to change the source RTL and produce a new version of the chip in a day. This process begins with manipulating the ADC to be used in the flow. Three files are needed describing the ADC block in various ways: a Verilog module, a `.lib` file to describe the timing related to its pins, and a milkyway file to describe the block physically for place and route. This milkyway must include the appropriate metal

layer blockages to avoid any routing over the design and pin cutouts to allow the routing tool to connect its pins. The Simulink design must then be run through the Insecta flow to generate an RTL description of the DSP core. At this point, a top-level Verilog description is needed which includes the I/O pads, the ADC, the ADC's digital circuits, and the DSP core. These are input into Design Compiler in order to map the design to actual 65nm gates. Before importing the resulting `.ddc` file into IC Compiler, there are several scripts that have been written to prepare the floorplan, pads, and power mesh for the chip. Once the floor plan is laid out, the design is ready to have the gates placed and routed. These steps are also scripted, and written into a makefile. The steps include placement, clock tree synthesis, clock routing, and signal routing, and may be done incrementally or all at once. Once the design is completed and meets timing requirements. The design must be 'streamed out' to Cadence for finishing. Design rule checking (DRC) and layout vs schematic checking (LVS) are done in virtuoso, and both must be error free before sending the `.gds` files to be fabricated. A thorough, step-by-step guide to all of these actions can be found in Appendix 1.

4 Implementation and Testing

4.1 Die Photo

The chip was fabricated by ST Microelectronics with 65nm features; it was taped out in early July 2014 and came back in October 2014. Figure 12 shows a die photo of the SPLASH chip, which measures 3.16mm per side. Overlaid are indications of the largest blocks on the chip. Certain memories are clearly visible, the largest of which are in the transmitter (TX) block. The analog I/O pads are concentrated in the bottom left corner, near the ADC, and the digital I/O pads surround the rest of the chip. A full pinout is in Appendix 2.

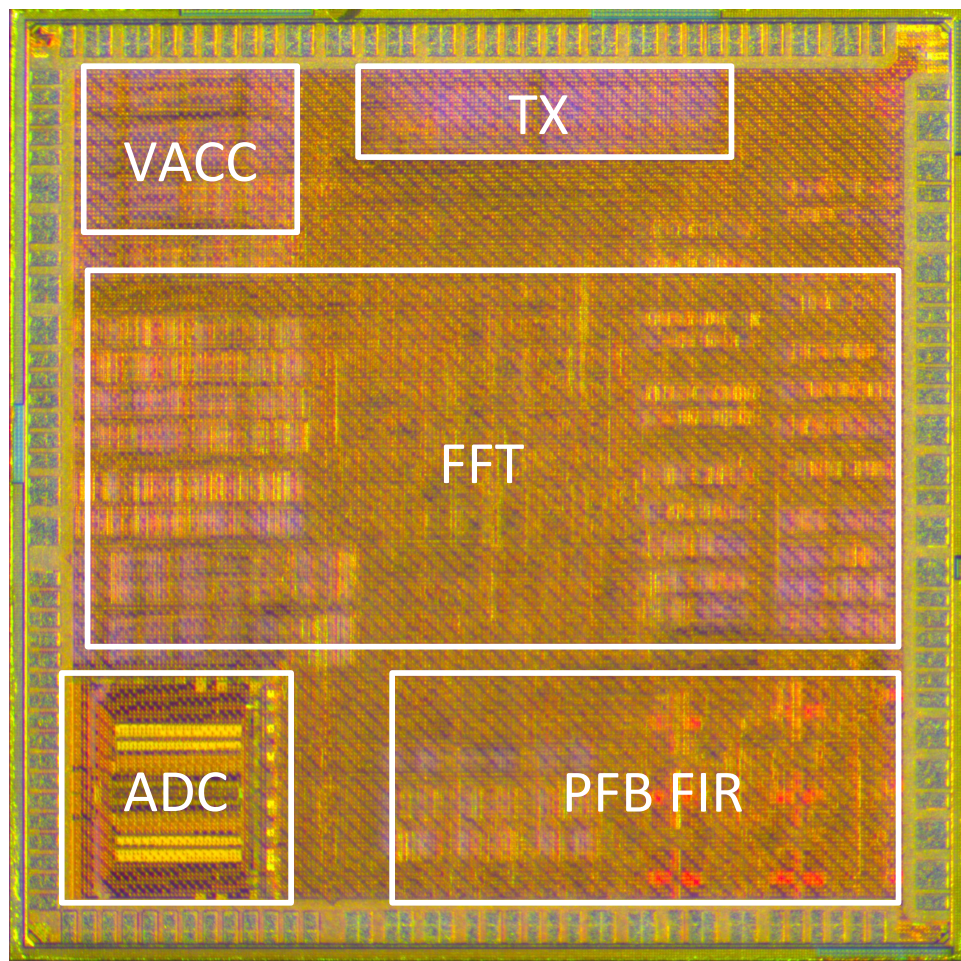


Figure 18: Die Photo of SPLASH

4.2 Test Setup

The test setup used for SPLASH is based on the RAVEN setup. There are 3 boards used for testing- an Opal Kelly Shuttle LX1, the RAVEN voltage supply board, and the SPLASH carrier board, described in more detail in the next section. The Shuttle LX1 is a general-purpose FMC carrier the combines a Xilinx Spartan-6 FPGA, high-speed USB 2.0, and the FrontPanel SDK, as seen in Figure 19. The RAVEN supply board has an FMC connector and 4 Linear Technology VLDO linear regulators. The output voltages are controlled by the FPGA via an I²C protocol.

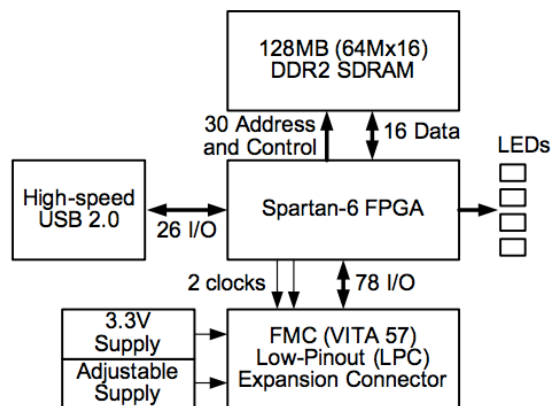


Figure 19: **Block Diagram of Shuttle LX1** [4]

4.3 PCB Design

This PCB was designed by Matt Dexter (Radio Astronomy Lab). It connects to the Raven supply board with an FMC connector. Special considerations were taken when designing the high speed analog inputs on the board to match wire lengths. Additionally, the power and ground rings and the I/O pads were very carefully placed to ensure that the wires are roughly parallel leaving the die. Figure 20 shows the layout and silkscreen of the carrier board.

The chip will be wire-bonded directly to the board- there is a gold plated ground pad on which the chip can be glued, and then wire bonded by Corwil Technology Corporation. A 'dam and fill' process will secure the chip and wires to the board. All other components on the board are being assembled by Digicom.

4.4 Test Software

Verilog is used to program the FPGA interface and python scripting is used to control the FPGA. There is specific Verilog that must be added for the Opal Kelly Host Interface, by which a computer can 'talk to'

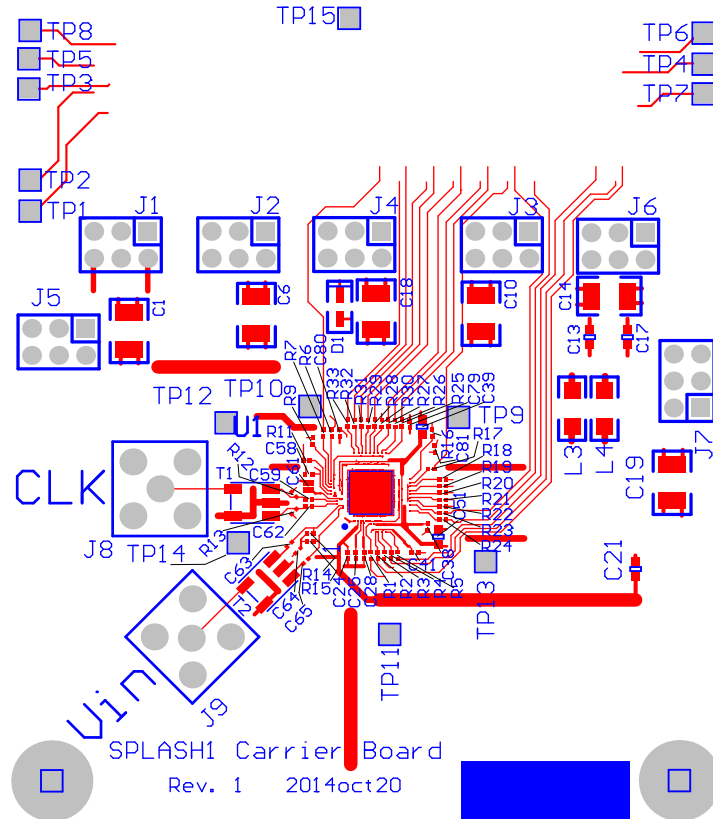


Figure 20: **Layout of SPLASH Carrier Board**

the FPGA through the USB. The python script then works by sending bits to specific wires into Opal Kelly. Additionally an I²C controller must be programmed into the FPGA, to set the voltages on the supply board.

Examples and tutorials on Opal Kelly programming, a start-up script for SPLASH, and the I²C controller Verilog module can be found in:

`/tools/scratch/lilrayray/opal-kelly-testing`

5 Conclusion and Future Work

In this report, I have explained the design and testing of an ASIC spectrometer with a bandwidth of 1.5GS/s and a spectral resolution of 183kHz, hence offering substantially enhanced functionality over previous state-of-the-art spectrometers. In addition, this spectrometer has a footprint of less than 10mm², and consumes less than 1W of power, making it an ideal component for use on a satellite mission, on which space and power are heavily constrained. I re-purposed a previous stand-alone ADC for use in this project and created a design flow procedure for making the physical design. The chip was fabricated in 65nm technology by ST Microelectronics, a carrier board has been designed on which the chip will be tested, and I have written the test software. Future work will involve testing the chip once it has been bonded onto the test board. If successful, the chip is slated to fly to Europa as part of a microwave spectrometer.

The next step in designing ASIC spectrometers will rely heavily on the development of a faster ADC. The goal for the successor of SPLASH is to have a 20GS/s ADC on chip. This will involve much work in designing an ADC that can achieve such a high sampling rate with a moderate effective number of bits (5-7). Possible candidates for this ADC design include another time-interleaved SAR ADC, which will require precise digital calibration to correct for timing and bandwidth mismatch, or a time-interleaved flash ADC. Another important change will be the vast improvement of the design flow. Rather than using Insecta to obtain Verilog from a Simulink description, the DSP will be described in CHISEL, a hardware construction language which has been developed at UC Berkeley [9]. CHISEL generates low-level Verilog designed to pass on to standard ASIC or FPGA tools, so in this way, the design process will be even faster and easier than before.

6 Appendix 1- Full 'SPLASH-in-a-day' Guide

This section describes in some detail the steps required to build SPLASH from a combinations of sources: Simulink, Verilog, and custom analog layout. To begin from scratch, check out the SPLASH git repo, preferably into a `/scratch` directory on one of the BWRC servers:

```
git clone /tools/designs/CASPER/projects/lilrayray/gitsplash/splash-icc-par/.git
```

6.1 Getting Started

All of the build steps require that VLSI software (e.g., Synopsys Design Compiler) as well as ST's libraries and scripts (e.g, FrontendKit) be set up correctly and in your path. The easiest way to accomplish this is to source the top-level script written for this purpose:

```
cd splash-icc-par
source splash.bashrc
```

This script calls the bash script to set up the environment and it may clobber existing environment variables that you've defined. It will work as intended for running all of the tools in this flow, but may break other setups.

The script mostly sources Synopsys and Cadence tools and is largely generic to any use of those tools on the BWRC servers. To work in cadence you will need to source another ST 65nm specific environment, which will be described later in this section. These scripts are roughly modeled after the reference version maintained by Brian Richards in `/tools/stalpha/local/setup_working_dir`.

There may come a time when ST releases a new design kit and we need to update our tool versions. The approximate way to do this is to run the command `uk-conf`, which will launch a graphical view that will allow you to select appropriate versions of each tool and generate a new `.ucdprod` file. (You can also generate the `.ucdprod` file by hand if you know the version numbers exactly.) Once you've done this, source the new tools as described above. To link in any new libraries, you need to do the following:

```
uk-lib-link -workspace 'pwd'
```

Unfortunately, this will replace the entire `LIBRARIES` directory. As we have many of our own custom libraries linked into this directory (as well as hacked ST libraries for which we've replaced the simlinks with our own) you'll need to recreate our custom build structure after running this command. Git can help you with this.

6.2 IP Generation

All custom IP needs to be packaged correctly such that the digital flow integrates the IP correctly. This includes:

- `.lib/.db` to describe the block for synthesis/timing. The `.lib` is a textual description of the timing related to every pin, and the `.db` is a compiled binary version of the `.lib` file. There is generally a different `.lib` and `.db` file for every process corner used in timing analysis, as setup/hold/transition will all change.
- `.lef/FRAM/CEL` to describe the block physically for place and route. P&R needs to know where it is allowed to connect each pin to. For example, you might have a signal routed in M2, M3, and M4, but you only want the tool to be able to connect to this signal from the edge—you need to describe this in the `.lef/FRAM/CEL` view. The `.lef` file is a textual description of pin locations, that is generated from an abstract view inside Virtuoso. The FRAM and CEL are compiled binary Milkyway views of the `.lef` file.
- `.v` to describe the block behaviorally. Even if there is no digital-like behavior, you need an empty Verilog (`.v`) file.

6.2.1 Setup

For this task, you must run the following commands:

```
cd cmos065
source .cshrc_cmos065
```

6.2.2 Usage

In order to generate the `.db` file, You must first open the attract tool and generate an attract with pins in metal layer 7 with pin cutouts in the M7 blockage and full blockages in all other layers. Once you have generated the abstract, you must run `make` in the `lib/` directory, which generates the `.lib` and `.db` files that contain information about the pins. To generate the `.lef/FRAM/CEL` files which contain the physical information for place and route, run `make` in the `lef/` directory, which will create the milky way view. You must make sure these are pointed to in the `mw.dir` and `db.dir` with all of the standard cells and memories so that the tools can access them. You can set up these pointers in the common settings script.

6.3 RTL Generation

6.3.1 Setup

Make sure you are back in the bash environment and have sourced the `splash.bashrc` script.

6.3.2 Usage

The top-level Verilog description of the chip includes the pads (and some other ST IP), the ADC block, and the DSP core. Because the DSP core is described in Simulink, extra steps are required to generate the Verilog needed for synthesis.

Insecta is used to generate a `.ddc` file. To set up the Insecta tool, follow the commands in:

```
tools/designs/luis/ficherosfinales/scripts/insectastart
```

Once you have started Insecta, make sure the settings are as follows before clicking 'Run Insecta':

```
Tech: 5.3.4 (ST)
Corner: WC_0.9
Effort: MED
Optimize: Target Clock
Period: 2ns
Hier: Boundary Opt
Select: Check Input through Export Schematics
```

Once you have the Verilog and ddc files for the DSP core, you must make sure that the name of the module in the top level Verilog matches the name of what you just generated. The tool adds a random hash to the name, so each time you re-run this part of the flow you must change the reference.

6.3.3 Memory Replacement

The memories are more pieces of ST IP that must be inserted into the design. In order to do so, run the following commands:

```
write_file -hier -format ddc {spec_entity_xxxx} -0 -splash_core.ddc
exit
dc_shell
start
open splash_core.ddc
source ../scripts/mem_replace.tcl
link
uniquify
current_design spec*
compile_ultra
```


6.4 Synthesis

6.4.1 Inputs

- `splash_toplevel.v` and `splash_core_luis.v`

These are the top level chip Verilog file and top level spectrometer Verilog.

- `crc04081120.ddc`

This is the compiled spectrometer entity core, the name will change every time you recompile.

- `st_verilog/*.v`

These files (`tisar_mux`, `filter`, `cal_slice`, etc.) make up the digital calibration section of the ADC.

- `clocks.tcl`

This is the script that sets up the clocks.

6.4.2 Usage

Now you must synthesize the entire design. Before doing so, make sure the specific name of the spectrometer matches the one listed in the script you will run below and inside of the `splash_core_luis.v` file. Once you have saved the pre-compiled version of the entire thing run the following commands:

```
dc_shell -64bit
source scripts/splash_top_level_make.tcl
```

6.4.3 Outputs

Now the results are mapped to actual gates, and the file should be saved as `splash_pads_mmdd.mapped.ddc`, where `mmdd` is the month and date. Now you must copy this into the 'ddc' folder within the ICC folder.

```
mv splash_pads_mmdd.mapped.ddc splash-icc-par/ddc/splash_pads.mapped.ddc
```

6.5 Place and Route (IC Compiler)

6.5.1 Setup

There are no extra actions needed to setup the environment for this step if you are continuing straight from the previous step.

6.5.2 Constraints and Floorplanning

There are a number of constraint and floorplanning files that you must have in order to begin the place and route process.

- `floorplan/splash_phy_constraints3_11.con`

This file sets up the pad ring. First it creates the corner cells, the power and ground pads, the filler cut cells, and the esd protection cells. It then tells the tools in which order to place the pads and specifies the spacing between each pad.

- `create_top_pins.tcl`

This script creates IO Ports for assorted power IO pads defined in the constraint file.

- `basic_ring.tpl` and `pg_mesh.tpl`

These files are the templates for creating the power and ground rings and mesh for power distribution throughout the chip.

- `floorplan.bounds.tcl`

This is the script that sets up the bounds that certain groups of cells have to fit within.

- `memory_placement.tcl`

This is the script that places the memories in specific locations. These specific placements, in combination with the floor plan bounds, were chosen after many iterations of placing the memories, having the tools automatically place the standard cells, and then analyzing density and congestion maps.

6.5.3 Place and Route

Place and route is controlled by a Makefile. It is possible to simply navigate to `splash-icc-par` and type `make`. Each step is run sequentially, so running `make` will run each step along the way. However, it is highly recommended to first type `make init.design.icc` and then view the results of floorplanning before continuing. This step is relatively quick and allows you to double check that the padding, memories, and power mesh all look correct. After this first step, the design is saved as `build-icc-yyyy-mm-dd.hh-mm` and can be viewed by opening IC Compiler and opening the specific design as follows:

```
icc_shell -64bit
start
open_mw_lib splash-icc-par/build-icc-xxxx-xx-xx_xx-xx/splash_pads_LIB
open_mw_cel splash_pads
```

If the floor plan is satisfactory, then you can exit IC Compiler and run **make** again. The tool will begin with the already floorplanned design and run the sequential steps of placement optimization, clock tree synthesis, clock optimization, clock routing, signal routing, route optimization, and chip finishing. The design is now saved as **build-iccdp-yyyy-mm-dd_hh-mm**. Again, if necessary, you can run up to a certain step and open and examine the design at that point. It is often a good idea to run the design all the way through clock tree synthesis and then open the design to verify the existence of the clock tree.

Note that each time you make a new version the most recent results of **init_design.icc** are pointed to by the symbolic link **current_icc@** and the final results are pointed to by **current_iccdp@**.

6.6 Finishing

6.6.1 Export to Cadence

In order to view the final design in Cadence, you must first export a GDS file from IC Compiler. To do so run the following (where the name at the end of the command is the name you want to give the GDS file) :

```
source ../icc_script/filler.tcl #fixes notching errors in the filler cells
source ../floorplan/create_top_pins.tcl to add supply pin terminals to the layout.
source ../icc_scripts/streamout.tcl to generate splash_pads.gds
```

Once you have exported the GDS file, you can stream it into Cadence by making sure you are back in the cshell cmos065 environment and opening Cadence Virtuoso. Open the stream in dialog under File >Import >Stream. For 'Stream File' choose the GDS file you just created. Then choose the library in which you want to place it, the top level cell name (probably splash_pads) and attach it to the existing technology library **cmos065**. Finally, click 'Translate.' You should now be able to open the design in the Library Manager.

One step that needs to be done on this specific design is to make sure all of the power and ground pads have the view of **layout_50u** rather than **abstract** or **layout_40u**. For some reason certain pads in the ST libraries have all three views available and some of them get imported with the wrong view. Do this by:

```
Press Shift-S to start the search and replace dialog
Search for inst
Add Criteria, view_name == abstract, Apply. Figure count should be >0
Replace view_name ->layout_50u
Press Replace All
Change criteria view_name == layout_40u, Apply. Figure count should be >0
Press Replace All again
Save the design
```

6.6.2 DRC

To run DRC, open the DRC dialog under Calibre >Run DRC and click 'Run DK DRC.' Click 'OK' for the Customization Settings.

Under the Setup Menu, make sure the 'DRC Options' box is checked, and click on DRC Options in the sidebar menu. Click on the tab called 'Connect' and check the box that says 'Connect nets named.' This makes sure that all nets called GND are connected to those named gnd, etc. Under the tab 'Include,' make sure to check the box 'Include Rule Files After Main DRC Rules File.' Then in the text box below that type in `bbox_nofiller.rul`. Now click 'Run DRC.' The results will appear in a new window.

If there are DRC errors, the errors should be fixed within IC compiler, and the GDS needs to be recreated. Inside IC Compiler, go Verification — Read 3rd-party DRC Error file to open the `.results.db` from Calibre. Then, delete vias, move wires, etc to fix DRC errors. One useful command is `route_zrt_eco`, which will fix DRC errors on wires and route all open nets.

One main cause for errors is at the boundary of IP. You can modify the IP to prevent the error from occurring (update the GDS and FRAM/CEL view), delete all offending routes, then reopen IC Compiler. The new IP should be visible with the changes, and you can run a `route_zrt_eco`.

Once you are done, export a new GDS, then rerun DRC.

6.6.3 LVS

The goal of LVS is to compare: the Verilog: `splash-icc-par/current-icc/results/splash_pads.output.pg.lvs.v` (or another verilog in the same directory) with the GDS: `splash-icc-par/current-icc/splash_pads.gds`

The above file was generated by:

```
Load the change-names-icc design in icc_shell
Run the command: source ../floorplan/create_top_pins.tcl
Run the command: source ../icc_scripts/streamout.tcl
```

7 Appendix 2 - Data Sheet

Sampling Rate	3GHz
Bandwidth	1.5GHz
Resolution	183kHz
Filter Type	4-tap Polyphase FIR filter
FFT Size	16384 point
Input	$1.8 V_{pp-diff}$
Technology	ST 65nm
Supply Voltage	1.2 V
Chip Area	9.98mm ²
ADC	0.18mm ²
DSP	8mm ²
Chip Power	950mW (predicted)
ADC	44.6mW
DSP	900mW (predicted)

Table 1: **Key Specs**

Table 1 outlines the key specs for the chip. On the next pages are tables describing the various modes and register values as well as the pinout for the chip with pin name, I/O direction, and description. The pin numbering begins with 1 on the bottom side of the chip in the left hand corner and counts up counter clockwise to 168 on the bottom of the left side of the chip.

State	enable	reset	shift
HOLD	0	x	x
INIT	1	1	x
SHIFT	1	0	1
CAPTURE	1	0	0

Table 2: **Test Controller Modes**

Code	State	Function
2'b00	WAIT_START_ACQUISITION	wait to start acquisition
2'b01	DATA_ACQUISITION	acquire data in the signature registers
2'b10	WAIT_OUTPUT_AVAILABLE	wait for the output to be available to dump the scan data on the output pin
2'b11	SCAN_DATA_OFFLOAD	wait for the scan dump to be complete and return to the wait state

Table 3: **Test Controller Instruction Descriptions**

Register	Length	Functionality
scan_dump_duration	10	total length of chain
command	4	triggering options
start value	8	trigger start
stop value	8	trigger stop
start mask	4	trigger mask
stop mask	4	trigger masks
ref count	44	acquisition count
ref start	44	start acquisition
ref stop	44	stop acquisition
forceblock1	32	accumulation length
forceblock2	32	shift schedule
forceblock3	32	control bits

Table 4: **Test Controller Registers**

Register	Bits	Description
Register 3	31:23	unused
	22	single mode
	21:9	unused
	8	ADC diff_en
	7	ADC ref_en
	6	ADC rs_ds_en
	5	ADC cal_type
	4	ADC t_cal_en
	3	ADC rs_ds_en
	2	filter select. 0= raw ADC output, 1=weighted, radix 2 output
	1:0	bypass mux. 00=ADC, 01=PFB, 10=FFT, 11=power

Table 5: **Force Block Control Programming**

Pin Number	Name	Direction	Description
1	esdcell	in	ESD protection cell, tie to analog ground
2	vrp2	in	ref voltage for CAPDAC in even channels
3	vrp2	in	ref voltage for CAPDAC in even channels
4	tisarvss	in	analog ground
5	vcmrst	out	analog even common mode
6	tisarvss	in	analog ground
7	tisarvd1	in	analog power: even channels, comparators, switches
8	tisarvd1	in	analog power: even channels, comparators, switches
9	tisarvss	in	analog ground
10	tidarvd3	in	power for digital circuitry in ADC (buffers, etc.)
11	tisarvd3	in	power for digital circuitry in ADC (buffers, etc.)
12	tisarvss	in	analog ground
13	tisarvss	in	analog ground
14	vddcore1.2	in	digital power
15	gndcore1.2	in	digital ground
16	vddcore1.2	in	digital power
17	gndcore1.2	in	digital ground
18	vddcore1.2	in	digital power
19	gndcore1.2	in	digital ground
20	io_sclko	out	analog scan clock out
21	vdd	in	power
22	gnd	in	ground
23	vdde	in	pad supply power
24	gnde	in	pad supply ground
25	io_sclki	in	clock in
26	vddcore1.2	in	digital power
27	gndcore1.2	in	digital ground
28	vddcore1.2	in	digital power
29	gndcore1.2	in	digital ground
30	io_scno	out	scan out
31	vddcore1.2	in	digital power
32	gndcore1.2	in	digital ground
33	vddcore1.2	in	digital power
34	gndcore1.2	in	digital ground
35	io_ser_clk_in	in	determines sample time for TC serial input data
36	vdd	in	power
37	gnd	in	ground
38	vdde	in	pad supply power
39	gnde	in	pad supply ground
40	io_rst	in	reset transmitter
41	NC	-	-

Table 6: **Bottom Side Pinout**

Pin Number	Name	Direction	Description
42	io_scan	in	scan
43	vddcore1_2	in	digital power
44	gndcore1_2	in	digital ground
45	vddcore1_2	in	digital power
46	gndcore1_2	in	digital ground
47	io_schni	in	scan in
48	vddcore1_2	in	digital power
49	gndcore1_2	in	digital ground
50	vddcore1_2	in	digital power
51	gndcore1_2	in	digital ground
52	serial_data_out_0	out	serial data out bit0
53	vdde	in	pad supply power
54	gnde	in	pad supply ground
55	vdd	in	power
56	gnd	in	ground
57	serial_data_out_1	out	serial data out bit1
58	vddcore1_2	in	digital power
59	gndcore1_2	in	digital ground
60	vddcore1_2	in	digital power
61	gndcore1_2	in	digital ground
62	serial_data_out_2	out	serial data out bit2
63	vddcore1_2	in	digital power
64	gndcore1_2	in	digital ground
65	vddcore1_2	in	digital power
66	gndcore1_2	in	digital ground
67	serial_data_out_3	out	serial data out bit3
68	vdde	in	pad supply power
69	gnde	in	pad supplyl ground
70	vdd	in	power
71	gnd	in	ground
72	io_serial_sync_out	out	indicates the beginning of a scan dump
73	vddcore1_2	in	digital power
74	gndcore1_2	in	digital ground
75	vddcore1_2	in	digital power
76	gndcore1_2	in	digital ground
77	io_tvg	in	test vector generator select DATA=0 TVG=1
78	vddcore1_2	in	digital power
79	gndcore1_2	in	digital ground
80	vddcore1_2	in	digital power
81	gndcore1_2	in	digital ground
82	io_valid	out	data out is valid
83	NC	-	-
84	NC	-	-

Table 7: **Right Side Pinout**

Pin Number	Name	Direction	Description
85	NC	-	-
86	NC	-	-
87	io_spec_reset	in	reset DSP and test controller
88	vddcore1.2	in	digital power
89	gndcore1.2	in	digital ground
90	vddcore1.2	in	digital power
91	gndcore1.2	in	digital ground
92	io_tc_mode	in	test controller mode HARD=0, SOFT=1
93	gnd	in	ground
94	vdd	in	power
95	gnde	in	pad supply ground
96	vdde	in	pad supply power
97	io_tc_busy	in	flag, set to 1 if reading data out
98	vddcore1.2	in	digital power
99	gndcore1.2	in	digital ground
100	vddcore1.2	in	digital power
101	gndcore1.2	in	digital ground
102	serial_misr_chain	out	data out from test controller
103	gnd	in	ground
104	vdd	in	power
105	gnde	in	pad supply ground
106	vdde	in	pad supply power
107	io_serial_program	in	valid program flag
108	vddcore1.2	in	digital power
109	gndcore1.2	in	digital ground
110	vddcore1.2	in	digital power
111	gndcore1.2	in	digital ground
112	io_serial_force_in	in	push state machine to OFFLOAD
113	vddcore1.2	in	digital power
114	gndcore1.2	in	digital ground
115	vddcore1.2	in	digital power
116	gndcore1.2	in	digital ground
117	io_scan_reset	in	analog scan chain reset
118	gnd	in	ground
119	vdd	in	power
120	gnde	in	pad supply ground
121	vdde	in	pad supply power
122	io_serial_data_in	in	test controller serial data in
123	vddcore1.2	in	digital power
124	gndcore1.2	in	digital ground
125	vddcore1.2	in	digital power
126	gndcore1.2	in	digital ground
127	io_start_stop	in	single spectrum=1, accumulation=0

Table 8: **Top Side Pinout**

Pin Number	Name	Direction	Description
128	gnd	in	ground
129	vdd	in	power
130	gnde	in	pad supply ground
131	vdde	in	pad supply power
132	io_serial_clock	in	test controller clock must be 4x slower
133	vddcore1_2	in	digital power
134	gndcore1_2	in	digital ground
135	vddcore1_2	in	digital power
136	gndcore1_2	in	digital ground
137	io_sar_rst	in	analog reset
138	vddcore1_2	in	digital power
139	gndcore1_2	in	digital ground
140	vddcore1_2	in	digital power
141	gndcore1_2	in	digital ground
142	io_misr_out	out	data out from MISR
143	gnd	in	ground
144	vdd	in	power
145	gnde	in	pad supply ground
146	vdde	in	pad supply power
147	io_clkout	out	digital clock out
148	vddcore1_2	in	digital power
149	gndcore1_2	in	digital ground
150	vddcore1_2	in	digital power
151	gndcore1_2	in	digital ground
152	tisarvss	in	analog ground
153	tisarvss	in	analog ground
154	vcmb	in	odd channel common mode
155	tisarvd2	in	analog power: odd channels, clock circuitry
156	tisarvd2	in	analog power: odd channels, clock circuitry
157	vrp1	in	ref voltage for CAPDAC in odd channels
158	vrp1	in	ref voltage for CAPDAC in odd channels
159	tisarvss	in	analog ground
160	io_adc_clkp	in	high speed clock inp
161	io_adc_clkn	in	high speed clock inn
162	tisarvss	in	analog ground
163	tisarvss	in	analog ground
164	tisarvss	in	analog ground
165	tisarvss	in	analog ground
160	io_adc_inn	in	high speed input n
161	io_adc_inp	in	high speed input p
168	tisarvss	in	analog ground

Table 9: Left Side Pinout

References

- [1] Stepanovic, D.; Nikolic, B., "A 2.8GS/s 44.6mW time-interleaved ADC achieving 50.9dB SNDR and 3dB effective resolution bandwidth of 1.5GHz in 65nm CMOS," VLSI Circuits (VLSIC), 2012 Symposium on , vol., no., pp.84,85, 13-15 June 2012.
- [2] Richards, B.; Nicolici, N.; Chen, H.; Chao, K.; Abiad, R.; Werthimer, D.; Nikolic, B., "A 1.5GS/s 4096-point digital spectrum analyzer for space-borne applications," Custom Integrated Circuits Conference, 2009. CICC '09. IEEE , vol., no., pp.499,502, 13-16 Sept. 2009.
- [3] https://casper.berkeley.edu/wiki/The_Polyphase_Filter_Bank_Technique
- [4] <http://assets00.opalkelly.com/library/WP-SemiconductorEval.pdf>
- [5] Markovic, D.; Chang, C.; Richards, B.; So, H.; Nikolic, B.; Brodersen, R.W., "ASIC Design and Verification in an FPGA Environment," Custom Integrated Circuits Conference, 2007. CICC '07. IEEE , vol., no., pp.737,740, 16-19 Sept. 2007.
- [6] Davis, W.R.; Ning Zhang; Camera, K.; Markovic, D.; Smilkstein, T.; Ammer, M.J.; Yeo, E.; Augsburger, S.A.; Nikolic, B.; Brodersen, R.W., "A design environment for high-throughput low-power dedicated signal processing systems," Solid-State Circuits, IEEE Journal of , vol.37, no.3, pp.420,431, Mar 2002.
- [7] http://www.nasa.gov/mission_pages/maven/main/index.html
- [8] <https://casper.berkeley.edu/wiki/ROACH2>
- [9] <https://chisel.eecs.berkeley.edu/>