Chapter 3

System Design

3.1 System Definition and Specifications

Based on referenced research works, the definition of what a stimulation system should be and how should it be implemented was achieved, in order to cover particle manipulation tests and procedures over a wide range of applications. Research areas include:

Frequency range: Based on the state of the art in particle manipulation, define and justify a frequency range eligible for a wide range of applications in manipulation procedures and tests.

Frequency synthesis methodology: Explore existing implementations, examine their applicability to this work and decide if the output frequency to system clock ratio can be achieved with them or if a novel methodology is needed to generate data for single and superimposed frequencies.

System design: Explore the design options for the system in this work and justify the selection, from logic-only, programmable array based, and processor based implementation.

Optimization and modularity: Analyze the trade-offs of the selected design scheme about performance (output frequency to clock frequency ratio), circuit size, and power consumption considering portable applications as the target. Explore the trends on intelligent systems about modularity, re-usability, integration and interconnection capabilities. Explore the core-based design methodology.

Configurability: Define and justify the parameters that should be open and configurable in order to obtain an electric stimulation system that covers the majority of electro-kinetically driven micro-fluidic devices.

Prototype implementation: Define a feasible prototype hardware implementation to run the application program so functional specifications and frequency synthesis methodology can be evaluated.

3.1.1 Motivation

Existing systems for manipulation and separation of particles depend on previously known information about the type of target particles or by experimenting on them; such experiments consist of controlling and changing or repeating electrical stimulation, analyzing response and sweeping signal parameters until desired results are achieved. An automated, programmable, configurable system is needed where reliable stimulation is needed for efficient and faster advances on research work about particle manipulation. Advantages of an automated, programmable, intelligent manipulation system:

- Multiple tests can be done and repeated by programming test sequences.
- Previously programmed test parameters for a known test sequence can be stored, accessed, and repeated.
- More reliable data results are obtained due to precise reproduction of test parameters.
- User interface allows rapidly configuring and operating the system for new tests and procedures.

- An intelligent design targets future Lab-on-chip implementations and portable Lab devices.
- A programmable system allows to run original application or to load a new one.
- A scalable design provides interconnection and communication channels so it can be integrated to other systems.

3.1.2 Statement of the Problem

There are current problems and limitations in particle manipulation procedures and research works, so present needs should be detected and solved; overcoming the state of the art and anticipate for future needs in stimulation systems would allow researchers to speed up experiments and results.

The trends show that experiments need more controlled testing environments by using more complex electric stimulation, which only programmable systems can deliver, like signal composition, dual frequency signals, traveling wave fields, mixing sine with square and triangle signals, and what may come in the future.

Besides, if frequency range of output signals could cover a wide spectrum of particle types, sizes, and shapes, research work would be more efficient and might reveal results from previously unknown experimental circumstances.

Last but not least, current implementation schemes for digital frequency systems should take as primary goals a low power, minimum size, and high performance design.

Figure 3.1 illustrates how the research work in all the related disciplines and the corresponding tasks lead to specific outcomes and contributions of this work in each of the four related research areas: the effect of electric fields in electro-kinetically driven fluidic devices, frequency synthesis methodologies, Lab-on-Chip systems, and System-on-Chip design.

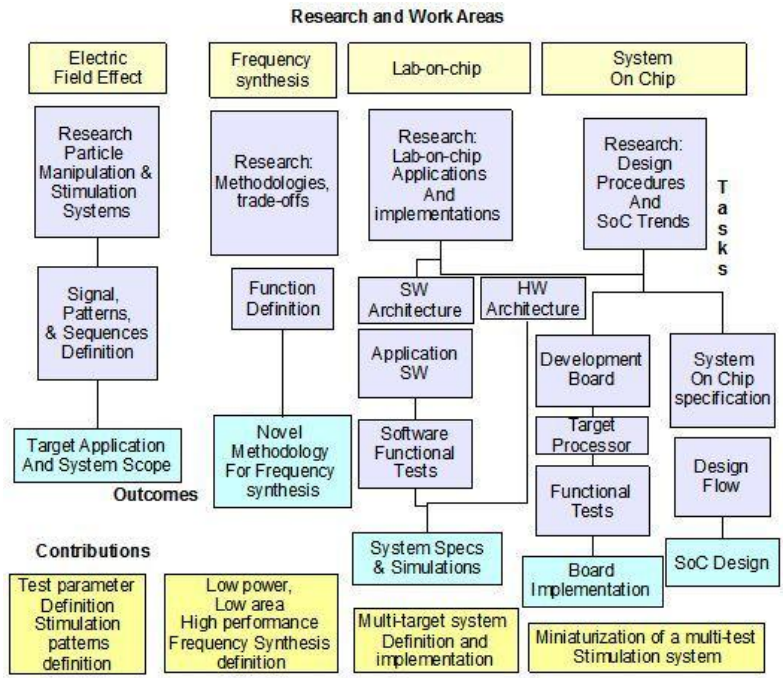

Figure 3.1 *Performed tasks, achieved outcomes and contributions made in the four research areas.*

The overall goal is to specify, define, design, and implement an open processor-based system that allows users from different areas to configure and automate their tests over a specific target particles or cells to obtain reliable and repeatable results in order to achieve the desired mobility effect. It is also desirable to have configurable system variables and test parameters that can be selected or programmed before the experiment or test is executed.

3.1.3 Proposed System

This work presents a processor-based stimulation system to generate signals and configurable tests for stimulation of micro-fluidic devices. It delivers multiple waveforms and patterns to cover a wide range of experiments and applications.

Specific tests or sequence of tests that should be made on specific particles may not be known by publication time since this is a developing area, so this system is designed to be configurable and to deliver a variety of signal patterns and combinations within a frequency range. The design consists of a set of cores integrated as a System-on-Chip (SoC) to configure, operate, and execute a stimulation system which delivers desired data.

This stimulation system includes user interface capabilities for configuration and operation, a memory system to upload and contain the application software for frequency synthesis, a processor to execute the program, and output ports to deliver data from synthesized frequency as shown in Figure 3.2.

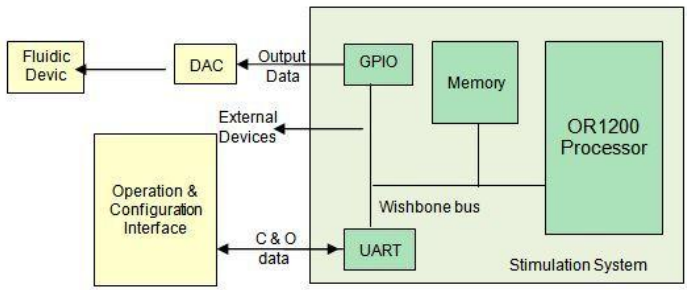

Figure 3.2 *Proposed systems.*

This system is also designed to favor an easy integration to existing or on-going designs of Lab on a chip: it provides input/output Wishbone buses for data and instructions so the system can be application independent by using a

ROM based Bios that loads the selected application software depending on the desired use. Existing or proposed Lab-on-chip systems to be connected to this stimulation system, which is using another standard communication bus like AMBA, can use a converting bridge for interconnection without changing the current design.

One of the possible implementations presented, shows how a different application program can be uploaded before operation, so modifications and additions to the program can be made and tested outside the system and later uploaded to an in-chip memory for operation. This capability, besides making the system adaptable to future applications, makes possible its integration to existing systems.

The user interface allows configuring the system, select mode of operation, select desired type of signal, selecting single or superimposed frequencies, and visualizing data being delivered. User interface interacts with the system via a standard serial port.

The memory system consists of a ROM to contain the boot-loader which uploads the application software at the beginning of operation and Harvard architecture of RAM to contain the program and the data for operation. The processor selected for the SoC design, the OR1200, is the best option from the available open source cores, and its corresponding instruction set covers the needs for this application software.

The on-chip communication is achieved using the Wishbone bus, which is the standard bus for open source cores, and allows a smooth integration of all the components in the system. Two Wishbone buses, one for data and one for instruction, are taken outside the chip so it can be integrated to other systems.

The hardware architecture for the chip is oriented to low power, low area, and low execution times, and by using open source cores this is a design that can be completed with no licensing cost during design and fabrication stages.

The application software implements the novel frequency synthesis methodology designed during this work, so it optimizes hardware resources such as memory map, instruction set, and system clock, in order to achieve maximum output-frequency/system-clock rate in output signals. The software can be tested on development boards based on the same or similar processor.

This system is also designed on a modular basis so it can be integrated, as is, into Lab-on-Chip systems or by adding new driver cores and the application software is designed in an open-source style so it can be configured or extended for future applications.

3.1.4 Frequency Synthesis

The novel frequency synthesis methodology developed for this system integrates the advantages of both, memory intensive and computation intensive approaches into one new synthesis methodology while keeping a low implementation area, low power, and high performance design.

For this system a look-up table is used to store base sine sampled data for a complete sine cycle. As any digital design the best tradeoff between hardware and software implementation should be selected: hardware is used for data storage and software for data processing computation. A software implemented algorithm is defined to select data from look-up table for target frequencies and store it in temporary tables; process data from temporary tables to get single or dual frequency data samples, and store them in output buffer tables.

Finally, and most important, a computation-free algorithm loads data from output buffer table and stores it in one or more output ports depending on the operation mode. An external conditioning circuitry including a DAC, a current to voltage converter and a voltage amplifier converts sampled data into the finally delivered analog signal.

3.1.5 Comprehensive System

As state of the art, research shows highly controlled experiment environments can be achieved when using more complex stimulation, so this system needs to deliver configurable multi-waveform, dual-frequency signals to speed-up research work on multi-particle manipulation tests. The system architecture has the foundation for control purposes, data storage and signal processing; it can be customized to achieve particular control and operation purposes of stimulation systems.

The mix of single-frequency signal generation along with signal superposition and system configuration capabilities, allows a wide control range on stimulation tests. Besides, with minimal modifications other waveforms and patterns can be obtained. A frequency range sweep can be run as a sequence of several user selected exposure times to analyze results under several stimulation conditions on the same experiment.

Execution times are lowered to its minimum so maximum output frequency is dependent only on the processor specification. Besides, a size optimized routine does not change for different generation modes or for different waveforms, so memory usage is kept at a minimum regardless the operation mode.

Memory architecture is designed for minimum area: data samples for sine, triangle and saw tooth waveforms are stored in base data tables using them, temporary tables are constructed based on selected output frequency and desired number of voltage steps; output or buffer tables are finally calculated after a time match operation depending of the number of channels to be updated simultaneously.

Data pre-processing and table preparation reduces computation instructions during signal generation achieving maximum output frequency to clock frequency ratio. The system can generate any periodic waveform as long as it is stored in memory data tables.

For signal updating, multiple simultaneous writes to output port are made so no loss in output frequency occurs when two or more signals are being delivered simultaneously. This port partitioning scheme is particularly convenient for this application.

The proposed system combines efficient use of hardware and software resources: minimum generation code, no computation during synthesis, and minimum memory access times.

3.1.6 Scope and Limitations

About research, the commitment is to review the state of the art on the four areas mentioned to identify the common ground for electric stimulation of fluidic devices, to keep-up with trends, and to anticipate to future stimulation needs. About new developments the challenge is to deliver a flexible and programmable system which runs a novel signal generation methodology and to prove its functionality.

About system design: the goal is to identify the system requirements, to define its functional specifications, for System-on-Chip -standard functionality version- and for development board -extended functionality version-, and to use available software and hardware resources to implement the design.

Limitations are related to time and to available resources: time because design decisions for this system are made based on what current research work shows and what can be identified as a trend; and resources are related to budget dependencies and access to licensed or open CAD tools to achieve the intended design.

Implementation on development board is limited to available processor, instruction set and communication ports specifications for that board. For chip implementation, system specifications like circuit area, power consumption and maximum output frequency are defined and limited by three factors: fabrication technology, physical libraries available, and efficiency of application software; the first two are resources dependent and the third is designer dependent.

Signal waveforms to be delivered are sine, triangle, and square and saw tooth wave. For dual superimposed frequencies the ratio between frequencies define the memory size needed for temporary and buffer tables: if frequencies are not exact multiples a hyper-cycle for resulting signal is not possible or very large, and that leads to unfeasible, large or infinite, memory needs.

3.2 Software Architecture

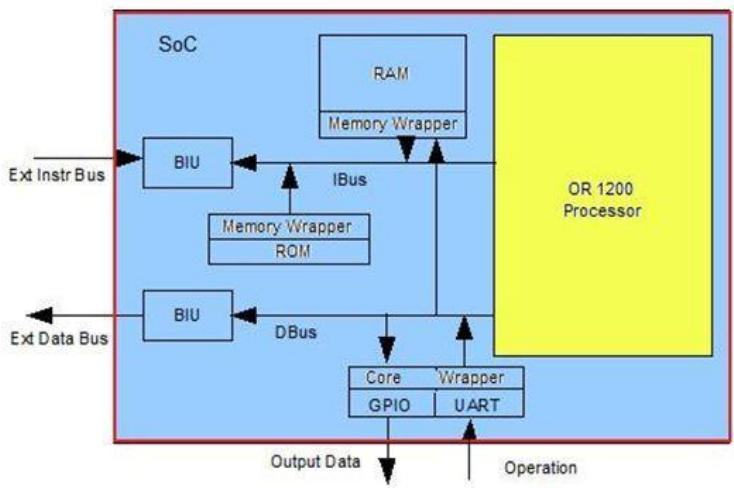

Figure 3.3 *System-on-Chip block diagram for the platform based design.*

The hardware architecture of the SoC is a platform and bus based architecture; it uses a selected set of open source cores and the Wishbone on-chip communication bus. The selection of the OR1200 processor is selected due to its previously demonstrated implementations in FPGAs and to its open instruction set. The on-chip memory array may contain the application software developed specifically for stimulating fluidic devices or a boot-loader to up-load different applications. In chip data, memory contains a complete cycle of the three base waveforms in 256 samples of 8-bit data each. A UART port is used to configure operation, to program sequence tests, to control operation and to visualize data during execution. The four 8-bit GPIO ports deliver processed data points for output signals, which may be, according to selection made in configuration: sine, triangle, or saw tooth and presenting single, dual or superimposed frequencies. Figure 3.3 shows interconnection between in-chip blocks; processor and Bus Interface Units connect directly to Wishbone

instruction and data buses, and memories peripherals connect via wrappers. Table 3.1 details the function of each primary block in the SoC.

Table 3.1 *Function description for primary system blocks.*

Element	Description
CPU (OR1200)	RISC CPU, Harvard architecture, cache memory for data and instructions, operates at 250 MHz max using 180 nm standard cells TSMC technology.
Wishbone Bus	On-chip bus for cache, main memories and interface peripherals
GPIO	Grouped in four I/O 8-bit ports, from open source cores: used as inputs for configuration and operation, as outputs for data
Clock and Reset	Receives clock from crystal oscillator, generates clock and reset signals for system operation, base clock for processor blocks and ¼ base clock for Wishbone bus.
UART	Serial port controller provides connection for external configuration and operation device. Open core source is used.
RAM and ROM	Memory blocks built with Artisan memory generators for verilog, vclef and gdsII views.
Interrupt controller	Exceptions handler from open cores included in OR1200 architecture.

3.2.1 Processor Based Implementation

This particular implementation for the processor was based on the open source files of the OR1200. The Open RISC 1200 is a synthesizable CPU core from Open Cores.org; it is a configurable open source Verilog implementation of the Open RISC 1000 architecture. The OR1200 is intended to be used in a variety of embedded applications. Some open source software, such as Linux, has been ported over to the OR1200 platform.

The GNU tool chain, including GCC, has also been ported to the architecture to aid in software development. The clock cycle for the OR1200 is 250 MHz at a 0.18 μm, 6ML fabrication process.

Estimated power consumption of this processor running at 250 MHz and implemented in 0.18μm technology is less than 1W at full throttle.

Available libraries: 180nm from TSMC will be used for this design. System specifications will meet application software requirements for standard version.

Components: Processor has been selected from available open source cores. Memories have been obtained through memory generators, Peripherals have been selected from open source cores.

Design constraints: Constraints are considered in this order: Performance, Area and Power. Target clock frequency is around 250MHz for an 180nm technology implementation. Processor area budget is less than $2mm^2$; on-chip memory area is less than $3mm^2$. Power budget for OR1200 processor in this technology is 1W at full throttle, added blocks should not exceed that by more than 20%.

3.2.2 SoC Components

The OR1200 is a RISC, Harvard Architecture processor with basic DSP capabilities. As an open source, customizable, core it is not optimized for power or size. This particular implementation for the processor was based on the open source files of the OR1200.

The Open RISC 1200 is a synthesizable CPU core from OpenCores.org; it is a configurable open source Verilog implementation of the Open RISC 1000 architecture.

It specifies a Central CPU/DSP block, Direct mapped data cache, Direct mapped instruction cache, Data MMU based on hash-based DTLB (Translation Lookaside Buffer), Instruction MMU based on hash-based ITLB, Power management unit and power management interface, Tick timer, Debug unit and development interface, Interrupt controller and interrupt interface, Instruction and Data WISHBONE interfaces, and a MAC unit. Peripherals and a memory

subsystem may be added using the implementation of a standardized 32-bit Wishbone bus interface.

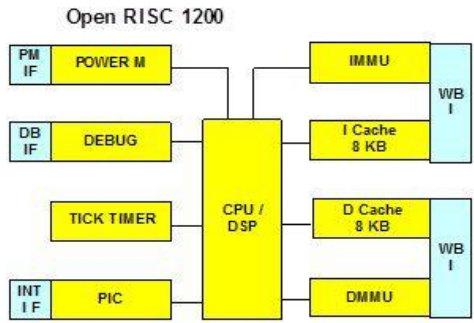

Figure 3.4 *OR1200 Architecture.*

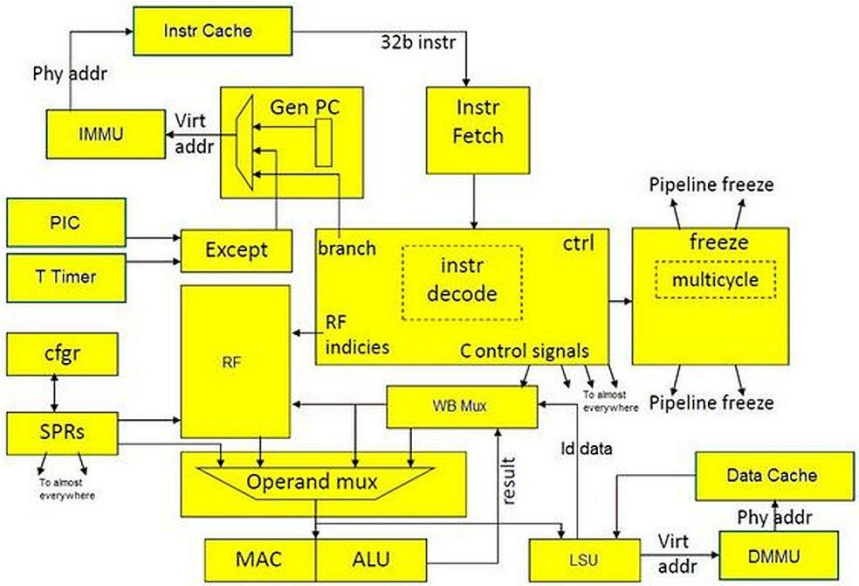

Figure 3.5 *OR1200 internal cores.*

The CPU is an implementation of the 32-bit ORBIS32 Instruction Set Architecture. It has five instruction formats and supports two addressing modes; it has a single-issue 5-stage pipeline, single cycle execution on most instructions.

It uses a Harvard architecture with separate MMUs for data and instruction memories, with support for virtual memory, a hash-based 1-way direct-mapped TLB with page size of 8 KB and a default size of 64 entries, and one-way direct-mapped D-Cache and I-Cache, 8 KB each.

Table 3.2 *Specifications of the OR1200.*

Concept	Specification
Architecture	32-bit scalar RISC, Harvard
Pipeline	5 stage, 32-bit integer instructions
DSP	Basic capabilities
Caches	Separate instruction and data, 1-way direct mapped. Configurable to 1, 2, 4, 8KB
Virtual memory	64-entry hash based 1-way direct mapped TLB for data and instruction MMU
Speed	Worst case: 150 dhrystone 2.1 MIPS at 150MHz (typical corner 250MHz) Best case for 180nm implementation: 300 dhrystone, 2.1MIPS at 300MHz
Size	Default configuration about 40K ASIC gates, 1M transistors
RTL status	Not optimized for speed or area
Instruction Set	Instruction unit handles only ORBIS32 instruction class. ORFPX32/64 and ORVDX64 instruction classes are not supported.
Configurable	Major characteristics can be set by user (see Core HW configuration table)
Communication bus	Wishbone, internal and for SoC interconnection
GPRs	General Purpose Register file is implemented as two synchronous dual-port memories, 32 words, and 32 bits per word.
Exceptions	Transparent to user software, same mechanism to handle all types of exceptions, control is transferred to an exception handler. See Exceptions table.
Interrupt controller	Direct enabled Int0 and Int1, Masked Int [31:2]
Tick timer	Clocked by RISC clock, re-start able, mask interrupt, Max count 2^32
Power management	Dynamically activated modes: slow and idle, Doze, Sleep, and Clock gating
Debug unit	Basic debugging; No watch points, break points or program flow control registers.
Clock & Reset	Core has several clock inputs: clk_cpu, clk_dc, clk_ic, clk_dmmu, clk_immu, clk_tt; all clocks must be in phase and as low skew as possible. Reset signal rest reset all flip flops when asserted high; when not asserted reset exception start.
Wishbone interface	Rev. B compliant, 32-bit bus width may connect to external peripherals and external memory subsystem.

Table 3.2 and 3.3 summarizes the specifications of the OR1200 selected for the design; Figures 3.4 and 3.5 shows the processor architecture.

Table 3.3 *List and description of the processor cores and the peripherals for the System-on-Chip.*

Core	Block description	Core	Block description
OR1200_alu	Arithmetic and Logic Unit	OR1200_lsu	Load and Storage Unit
OR1200_cfgr	Configuration Registers	OR1200_mult_mac	Multiply and MAC Unit
OR1200_ctrl	Control Unit	OR1200_operandmuxes	Operand Mixes
OR1200_dc_top	Data Cache	OR1200_pic	Programmable Interrupt Controller
OR1200_dmmu_top	Data Memory Management Unit	OR1200_pm	Power Management Unit
OR1200_du	Debug Unit	OR1200_rf	Register File
OR1200_except	Exceptions Unit	OR1200_sb	Store Buffer
OR1200_freeze	Freeze Unit	OR1200_sprs	Special Purpose Registers
OR1200_genpc	General Program Counter	OR1200_tt	Tick Timer
OR1200_gpio	General Purpose Input Output	OR1200_uart	Universal Asynchronous Rec/Trans
OR1200_ic_top	Instruction Cache	OR1200_wb_biu	Wishbone Bus Interface Unit
OR1200_if	Instruction Fetch	OR1200_wbmux	Wishbone Mux
OR1200_immu_top	Instruction Memory Management Unit	-	-

Communication Bus. The wishbone bus has become the standard communication bus for the open source cores. It serves as the in-chip bus and as the interface bus for the SoC with the external world, and as for Harvard architecture there are separated buses for data and for instructions.

Instruction interface is used to connect OR1200 core to memory subsystem for purpose of fetching instructions or instruction cache lines. Data interface is used to connect OR1200 core to external peripherals and memory subsystem for

purpose of reading and writing data or data cache lines. Table 3.4 lists signals for instruction lines (data lines are named dwb_xxx).

Table 3.4 *The Wishbone instruction bus.*

Signal	Width	I/O	Description
iwb_CLK_I	1	I	Clock input
iwb_RST_I	1	I	Reset input
iwb_CYC_O	1	O	Indicates valid bus cycle (core select)
iwb_ADR_O	32	O	Address outputs
iwb_DAT_I	32	I	Data inputs
iwb_DAT_O	32	O	Data outputs
iwb_SEL_O	4	O	Indicates valid bytes of data bus (during valid cycle it must be 0xf)
iwb_ACK_I	1	I	Acknowledgment input (normal transaction termination)
iwb_ERR_I	1	I	Error acknowledgment input (abnormal transaction termination)
iwb_RTY_I	1	I	In OR1200 treated same way as iwb_ERR_I.
iwb_WE_O	1	O	Write transaction when asserted high
iwb_STB_O	1	O	Indicates valid data transfer cycle

Memory System: As Harvard architecture, Instruction memory and Data memory are kept separated. Instruction Memory stores the application program for System configuration, System operation, Frequency Synthesis, and Output data delivering. Data Memory stores data samples for required waveforms: look-up tables containing sine, triangle, and saw tooth wave signal data; Temporary tables for processed data, and Buffer output tables for final processed data.

Possible implementations are explored: storing application software in on-chip ROM adds circuit area and restrain functionality to what's stored; loading application software to RAM using a boot-loader reduces memory area and allows additions to application software to be made and debugged in development board before being loaded into the chip.

A similar concept was developed for waveform data tables: data can be stored in on-chip ROM with fixed data width and samples per waveform cycle, this increases in-chip data memory space and reduces flexibility to change data size and samples per cycle. Otherwise sample data can be modified outside the chip, as well as the number of data samples per cycle, and then uploaded to RAM using the boot-loader. This shows the trade-off between fully customized designs versus a configurable programmable circuit design.

User Interface. A UART port has been selected for system configuration and operation. It was selected over a USB port since it takes less circuit area and speed is not important during configuration and operation setting. For extended interfacing capabilities, Wishbone data and instruction buses have been added, they can be used for external memory access and to connect this system to external peripheral systems, drivers or bridges.

User interface functionality has been summarized in Table 3.5 according to the system configuration and operation needs:

Table 3.5 *User interface: primary functions.*

Function	Description
Load	Load application software from available selections
Select	Define waveform to be used for next experiment
Configure	Select waveform, samples per cycle, output frequencies, operation mode, and exposure time for signal generation.
Operate	Start signal generation disabling other functionalities to maximize output frequency. Keep continuous operation until exposure time finishes or stop request is received.
Monitor	Displays feedback info from operation and data being sent to output port. This function can be disabled to eliminate execution time for monitoring and maximize output frequency.

Output Ports: Parallel ports have been used as output channels: a 32 bit (4x8 bit) GPIO port is used to deliver output data. Standard data is 8 bit wide, so up to four channels are available. Output data is delivered in one of this

forms depending on operation mode: in mode 1, 8-bit data samples containing one single frequency are delivered; in mode 2, 16-bit data samples containing two separate frequencies are delivered in two 8-bit channels, 8-bit each; in mode 3, 8-bit data samples containing two superimposed frequencies are delivered in one 8-bit channel.

3.3 Challenges for Variable Optimization

Here are shown the major challenges to be faced in the definition and design stages of the system; also is presented the decision to be analyzed and justified at each concept.

To obtain the maximum output frequency from a base operation frequency on the value you need. Variable optimization: processor selection, design and fabrication models availability.

1. To minimize execution code when in signal generation routine, so nominal output frequency is not reduced. Decision: Base data tables storage scheme, output memory buffer use, only one executing thread when running signal generation.

2. To define a data selection algorithm for waveform construction to reduce harmonic addition. Decision: Define an algorithm to select a set of data that minimizes gap between voltage steps.

3. To optimize code for minimum execution time on procedures like:

    - Select data from tables. Variable optimization: Equally time separated data or equally voltage separated data for harmonic reduction.

- Addressing memory and load data from memory. Decision: schemes for storing and addressing original waveform data, in cache or external memory.

- Buffer memory table construction. Variable optimization: separate, continuous, adjacent or superimposed memory segments when generating more than 1 signal, to reduce access times and maximize output frequency.

4. To keep power consumption low. Variable optimization: Consider the system as a whole or by operating mode, since most of the time the system will be in stand-by or configuring mode.

- Software related consumption. Application routines should be minimized on code size and execution time, in all operating modes: stand-by mode, configuring mode, pre-processing mode, and signal generation mode.

- Hardware related consumption. Determine power consumption for:

- Processor, in all operating modes.

- Memory, in read and write access.

- For every core include activity factor in power estimation.

3.4 System-on-Chip Specifications

The bus based architecture has been defined, and four implementation options have been explored, in order to compare performance and circuit size for each of those implementations. Since main impact in circuit area is due to instruction and data memory, this has been the parameter to be set first, keeping the configurable

and programmable capabilities in focus. Option 4 from Table 3.6 has been selected for the SoC implementation; for the development board implementation there were less memory restrictions and load-store-execute flow was defined by software development tools.

1. Parameters:

   - Instruction and Data Cache size: 2 K-bytes blocks, up to 8 K-bytes total.
   - Instruction on-chip RAM: 2 K-bytes blocks, up to 8 K-bytes total.
   - Instruction on-chip ROM: 256 bytes for boot-loader or 2 K-bytes blocks, up to 8 K-bytes total for on-chip application.

2. Application Software functionality.

   - Standard implementation on SoC. Program size < 4Kb. Data size < 1Kb. Output data: two output channels; Operation: resolution from coarse 8bit data. Output patterns: sine, triangle and saw tooth for single or superimposed frequencies.
   - Extended implementation on development board. Program size < 8Kb. Data size < 4Kb. Number of output channels is board dependent. Output patterns: sinusoidal, triangle, and saw tooth, for single or any mix of two superimposed waveforms of different frequencies. A sequence of user defined test with different time exposure.

3. Variables

   - Circuit Area.
   - Processor area: ALU, Registers, MMU, Exceptions, Control.

- On-chip memory area: Instruction and Data RAM, Instruction ROM.
- Total chip area: processor, memories, peripherals.

4. Performance

- Pre- processing times: time to store processed and buffer data tables.
- Execution times: for each functional module in application.
- Time between samples: maximum time between stores to GPIO port, minimum output frequency.

5. Hardware configurations. The way to load and store application software, along with the selected functionality for it, can fit into several architectures. Application software can be stored and modified on external flash or EEPROM memory, and loaded into in-chip RAM using a small boot-loader.

6. Possible implementations: select block size for Instruction cache memory, Data RAM, and ROM.

7. Operation Flow. Application software may be stored in on-chip ROM with no further modification or debugging capabilities after fabrication, or stored in external memory for debugging, modifications, and up-grade test in development board, to be loaded into chip during boot-load. A top-level operation flow is shown in Table 3.6 for four possible implementations.

Table 3.6 *Possible implementations, shown at top-level operation.*

	Application Software stored in in-chip ROM	**Application Software loaded from external Flash**
In-chip RAM size is smaller than application software size	Option 1 Go to program Start instruction. Load pre-processing program from in-chip ROM into I Cache.	Option 3 Start boot loader. Load external pre-processing program into in-chip RAM.

	Application Software stored in in-chip ROM	Application Software loaded from external Flash
In-chip RAM size is larger than application software size	Read operation parameters from interface. Load data from in-chip ROM into D Cache. Generate and store temporary and buffer data tables in D Cache. Load data generating program from in-chip ROM into I Cache. Store data in buffer tables from D Cache to output port.	Read operation parameters from interface. Load external data into in-chip RAM. Generate and store temporary and buffer data tables in D Cache. Load external data generating program into in-chip RAM. Generate and store temporary and buffer data tables in D Cache. Store data in buffer tables from D Cache to output port.
	Option 2	Option 4
	Go to program Start instruction. Load program from in-chip ROM into I Cache. Read operation parameters from interface. Load data from in-chip ROM into D Cache. Generate and store temporary and buffer data tables in D Cache. Store data in buffer tables from D Cache to output port.	Start boot loader. Load external program into in-chip RAM or I Cache. Read operation parameters from interface. Load external data into in-chip RAM or D-Cache. Generate and store temporary and buffer data tables in D Cache. Store data in buffer tables from D Cache to output port.

3.5 Signal Generation

A goal of this work is to achieve the maximum output frequency for a given architecture, processor speed, and memory access times. The key tasks for this achievement have been: a) data pre-processing, b) the signal generation scheme, and c) the memory access routine.

The equation for output frequency in the selected waveform is:

$$F_0 = \frac{1}{tbs * spc}$$

Where F_o is the output frequency, *tbs* is time between samples (the time between two consecutive data samples to be sent to the output port), and *spc* is

the number of samples per cycle (the number of data samples used to build a complete cycle of the selected waveform).

The value of tbs depends on the system clock cycle and the instructions that take for the signal generation routine to get a new data sample from the output table and send it to the output port:

$$tbs = (\#\ of\ \ instructions)(clock\ \ period)$$

For example, for a system clock of 100 MHz, using 12 samples per sine cycle, and a simple load-store routine of 6 instructions, the cycle period would be 1/100E+6 = 10 ns and the Output frequency:

$$F_0 = \frac{1}{(6)(10E-9)(12)} = 1.388MHz$$

As can be seen from (3), to get higher frequencies a tradeoff can be made by, e.g., using fewer samples per waveform cycle, eliminate check-for-stop during generation or change instruction counting for timer operation.

A precise count for clock cycles between output data samples, and therefore maximum output frequency, can be calculated after compiling the application software for the target processor, the OR1200, based on execution cycles per instruction type shown in Table 3.7.

Table 3.7 *Instruction Set Architecture execution times.*

Instruction type	Cycles to execute
Load	2, if cache hit
Store	1, if cache hit
Integer arithmetic	1
Multiply	3
Compare, logical	1
Rotate, Shift	1

Another way to look at output frequency is to calculate from processor speed and from cycles per instruction in the signal generation cycle:

$$T_{min} = \frac{Cycles\ between\ samples}{Pr\,ocessor\ speed}$$

For OR1200 running @ 200MHz:

$$T_{min} = \frac{10}{200\text{MHz}} = 0.05\mu\text{s}$$

If 10 data samples for the sinusoidal signal are desired, then:

$$T\,sin = (10)(0.05\mu\text{s}) = 0.5\mu\text{s}$$

$$F_0 = \frac{1}{T\,sin} = \frac{1}{0.4\mu s} = 2\ MHz$$

Which, according to the "Nyquist–Shannon sampling theorem", states that perfect reconstruction of a signal is possible when the sampling frequency is greater than twice the maximum frequency of the signal being sampled. In this case minimum execution time is 0.05 μs, so the system could generate, at most, 10 MHz signals. At this sampling rate an external filter will be needed to improve spectral purity.

3.5.1 Frequency Synthesis Methodology

Frequency synthesis has been historically achieved using several analog and digital approaches. The digital approach favors miniaturization and additional functionalities such as data storage and data processing. Actual devices which deliver output signals for a wide frequency range can be found already in wireless communications, but they are application specific and do not allow additional

functions. A variety of signals and patterns have been found to be useful in the mentioned range of applications, where researchers currently work with manual procedures using regular equipment as signal generators or oscillators. A more controlled experiment setting is desired so research results can speed up, and it can be achieved by having complex electric stimulation and varying signal parameters such as frequency, waveform, superimposed patterns, etc.

Besides the problem of delivering complex signal patterns, electric stimulation has to be implemented in a small size device because typical applications demand portable stimulation and test instruments, our approach integrates frequency synthesis, with multi-waveform generation, multi-waveform superposition, and operation configurability, so the implementation can be customized for multiple applications and processes. Besides, as a modular processor based solution is implemented, the methodology takes advantage of it by remaining generic and open to modifications, additions, and upgrades.

At the end, a small, low power implementation was achieved. Original digital frequency synthesis schemes calculate sine values on the fly (computation intensive scheme) or access pre-stored values from memory (memory intensive scheme). None of both schemes are useful by themselves in this system due to the goal of maximizing output frequency to clock frequency ratio, and to the memory and speed optimization tradeoffs between both schemes.

For performance optimization in this system the main intention of pre-processing the base sine data sample is to reduce computation during signal generation (i.e. while sending processed data to the output port); this way pre-processing times does not impact maximum output frequency. Computation intensive versus Memory intensive schemes are shown in Figure 3.6:

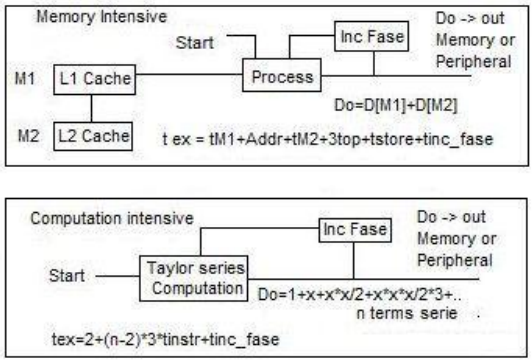

Figure 3.6 *DDFS: Memory intensive versus Computation intensive methodologies.*

3.5.2 Output Data

Available signal waveforms for board implementations are sine, triangle, and saw tooth. Preprocessing base data into temporary and into output buffer tables allows application software to do computation intensive tasks before operation and simple address-load-store operations during signal generation: Data input for pre-processing algorithm comes from base data samples tables. Selected samples are extracted by translating temporal spacing into memory spacing to achieve target signal frequency. Number of data samples per cycle is user defined.

When superimposed frequencies are desired pre-processing is executed twice with correspondent time-space parameters. Intermediate data is stored in separate temporary tables for each processed frequency.

Final processed data is stored in one output buffer table for simple access, low execution times.

Figure 3.7 shows data processing for one single frequency, and 9b for two single frequency outputs – the last step would be add for superposition and concatenate for separate frequencies-. See appendix A2 for the complete data

tables of base waveforms and an example of data processing for superimposing two frequencies.

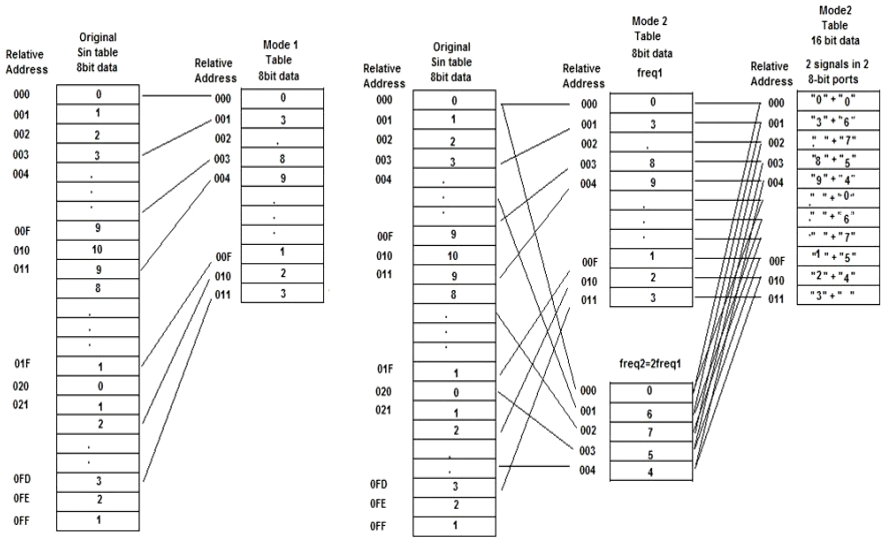

Figure 3.7 *a) Data processing for one single frequency. b) Data processing for two single frequency outputs. Last processing step is Add for frequency superposition or Concatenate for separate frequencies.*

3.5.3 Frequency Sweep and Superposition

Preprocessed data from temporary tables are superimposed by time matching of data samples from both separate frequencies. Data samples are added when time matches among samples and time holes due to frequency difference is filled with last data. Output data is stored into buffer tables. If there is no common factor between frequencies, buffer table size can grow indefinitely.

A set of sequenced tests can be programmed to be executed when characterization test need to go through a frequency range to identify particle's properties or behavior. These sequenced tests can sweep a desired frequency

range in user defined steps, for example, a 1 MHz signal is delivered for 30 seconds and a 10 MHz signal is delivered for 60 seconds after the first one, and so on. This frequency sweeps are useful when the particle's behavior is unknown or when running a characterization experiment.

3.5.4 Methodology Software Architecture

Application software is developed to perform four main tasks: Define and store waveform data samples, get configuration and operation parameters from user, process data to obtain selected output signals, and execute frequency synthesis to deliver processed output data samples.

Waveform data samples for a complete cycle are pre-calculated and stored as integer numbers in a 0-255 scale, being 0 the lowest peak value of the waveform (-V), 127 the mid-value (0), and 255 the highest peak value (+V). Values are stored in data RAM to reduce ROM needs. For the three stated waveforms 3 x 256 bytes of data space is needed. As these values are stored into RAM along with the program code uploading, new data tables with different waveforms can be included in the source files of the application and the software architecture remains unchanged.

Configuration and operation parameters are for user to select the type of waveform desired, the operation mode to be executed, and the time period to deliver the outputs.

When operation mode users select 1 out of 3: one output signal with one single frequency, two separate output signals with two different frequencies, or one output signal with two superimposed frequencies.

Data processing takes base waveform data to generate a temporary table containing selected data to form the desired frequency. Temporary tables for two different frequencies are generated for modes 2 and 3. Temporary tables in modes 2 and 3 are processed to form one output table containing two separate or superimposed frequencies. Values for temporary and output tables are stored in RAM during processing.

Output data delivering, to complete frequency synthesis process, takes data from processed output table and sends it to output port, using 8 bits for modes 1 and 3, or 16 bits for operation mode 2. Figure 3.8 shows application software flow:

Application functionality is achieved by independent tasks. Table 3.8 presents function description for detailing software execution.

Table 3.8 *Application Software, Function Description.*

Function	Description
Get operation parameters. Configure operation.	Gets operation parameters: frequency for output signals, samples per cycle, operation mode
Generate data samples for waveforms	Calculates and stores base sine data into memory table.
Calculate time and space between samples to construct desired frequency, mode 1.	Calculates time and space intervals to extract data from base table to construct one desired frequency.
Mode 1: generate output table for selected frequency	Process data from base table and store into temporary table.
Calculate time and space between samples for two frequencies, modes 2 and 3.	Calculates time and space intervals to extract data from base table to construct two different desired frequencies.
Modes 2 and 3: generate separate temporary tables for each frequency	Process data from base table and store into two separate temporary tables.
Mode 2: generate output table for selected concatenated frequencies	Process data from temporary table and store into one output table containing two frequencies in two separate signals.
Mode 3: generate output table for selected superimposed frequencies	Process data from base table and store into one output table containing two frequencies in one signal.
Send data from output table to output port	Take processed data from output table and store it into output port. 8 bits for modes 1 and 3, 16 bits for mode 2.

Function	Description
Format data to 8 bits in base waveform tables	Scale base sine data from -1 to 1 to 0-255 (8bits).
Continuous cycle of signal generation	Deliver output data continuously until stop requested or sequence time finished.
Non-multiple frequencies protection	Eliminates remaining cycle data for superimposed frequencies when no hyper-cycle is possible.
Variable size protection during data processing	Scale to 8 bits intermediate data resulting from operations to fit temporary and output tables.
Transmit output data for record and re-use	Transmit delivered data to serial port to be displayed or stored by user interface for visualization, record,, or future use.
Scale data, modes 1 and 3	Scale output data in modes 1 and 3 to 8 bits
Scale data, mode 2	Scale output data in mode 2 to 16 bits

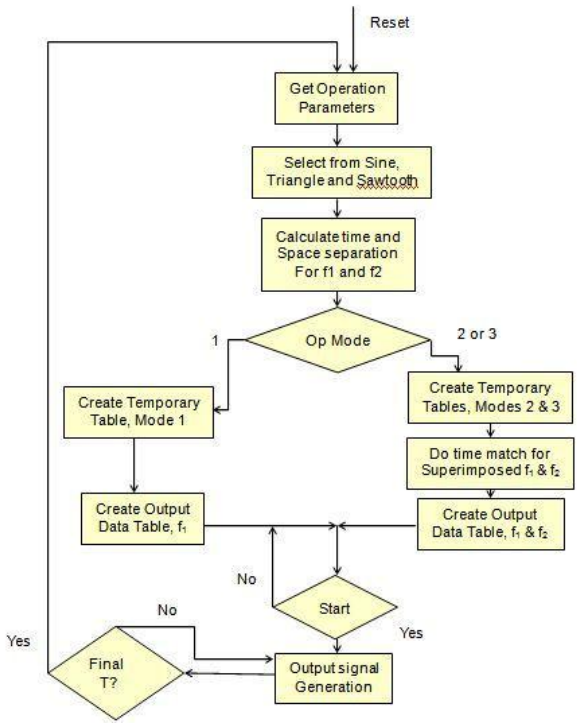

Figure 3.8 *Application Software Flow.*

3.6 ASIC Design Flow

In this section are presented a description of the design methodology, the CAD tools used in the chip design flow, and the procedures and results on the clusters involved in the design: synthesis, timing, memory blocks generation, place and route of the system cores, power consumption analysis, IO ring design, and integration at chip level.

3.6.1 Design Methodology

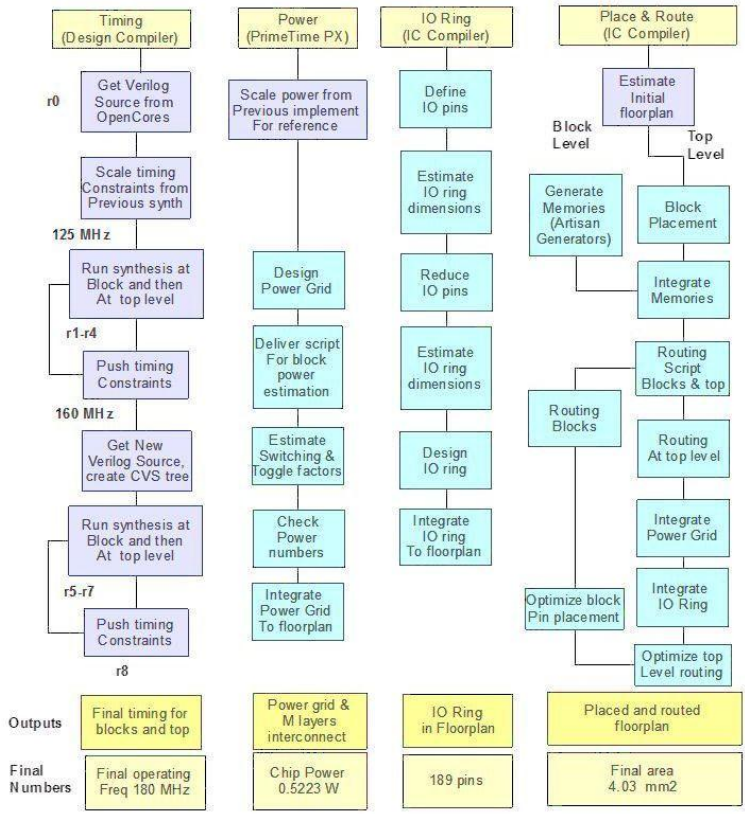

Figure 3.9 *Chip design flow on four areas: Timing, Power, IO pins, Area.*

Open source cores have been used to integrate a modular architecture over an open source bus. The design flow has been followed for optimal performance, minimum chip area and minimum power consumption, in that priority order when a compromise was needed. Work on all the areas relevant to physical design has been done: timing optimization, power estimation and grid design, IO pin selection and ring design, block placement and pin optimization for routing, and layout generation for minimal area.

3.6.2 CAD Tools

During the design flow, CAD tools have been used and the design process is not seamless due, mostly, to input-output file formats and compatibility between tools. As a reference, a tutorial on design flow was reviewed. Table 3.9 shows the tools used for each of the main design tasks. Tasks have been performed sequentially in the first design steps; in the later design steps they were performed in parallel and iteratively to work on the trade-offs of the design: chip area, power consumption, and system timing/performance. Figure 3.9 illustrates the design flow.

Table 3.9 *CAD Tools Used in design flow.*

Concept	Tasks	Tools
Timing	Synthesis process at block level and at processor level. Set initial time constraints at block level. Iteratively push timing constraints at block and chip level. Deliver block and global timing analysis.	Synopsys, Design Compiler (DC)
Place and Route	Initial area estimations at block level. Deliver an initial floor plan for routing. Placement and routing at block level. Global routing. Placement optimizations.	Synopsys, Design Compiler (DC) Synopsys, Integrated Circuit Compiler (ICC)
Memories	Generate logical and physical views for RAM and ROM blocks. Generate different aspect ratio implementations for placement optimization. Create memory wrappers for integration.	Artisan, Memory Generators Synopsys, Design Compiler (DC)
Power	Initial power estimations at block level. Iterative Power Grid Design.	Synopsys, Design Compiler Synopsys, PrimeTime PX

Concept	Tasks	Tools
IO Ring	Iterative IO Ring design.	Synopsys, Integrated Circuit Compiler (ICC)
Clock	Clock tree synthesis at chip level.	Synopsys, Integrated Circuit Compiler (ICC)
Integration	Integrate Power grid to floor plan. Integrate IO ring to floor plan. Integrate memory blocks at chip level. Integrate clock tree at chip level.	Synopsys, Integrated Circuit Compiler (ICC) JupiterXT, Synopsys
Verification	Functional verification at block and processor level	Mentor Graphics, ModelSim

3.6.3 Synthesis and Timing

During the synthesis process the local and global timing has been optimized and initial area and power estimations have been done. Eight rounds of synthesis have been performed until timing convergence and closure have been achieved. Synthesis rounds to get minimum slack time, area estimations, and power estimations have been done with Design Compiler from Synopsys. Selected processor cores from Open Cores have been used for bus compatibility and minimum edition of source codes. The available versions of these open source cores are not optimized for performance or area, so optimization in these areas have been done during this design process.

A simulation for functionality, at processor level, has been done before starting synthesis process, using ModelSim from Mentor Graphics.

The tasks performed and the obtained results from the rounds of synthesis were:

Round 0: Estimate initial timing constraints for each core based on gate count, estimated size and worst case data path.

1st to 4th first rounds: Run synthesis at block level pushing timing constraints; consider adjacent blocks in data path to push constraints for required arrival times and arrival times.

5th and 6th rounds: Run synthesis at top level. Identify critical paths. Push timing constraints on critical paths. Use new version of open core source repeat from round 5.

7th and final 8th rounds: Update CVS tree (source version control) to match last rounds. Round 8 was the final round for pushing the block and global timing constraints.

The global path delay at top level synthesis has been calculated by:

Global path delay= Arrival time + Pass-through + (cycle time – Required arrival time) + interconnect delay

Where the first three terms depend on both, block level synthesis and top level synthesis, and the last term depends on floor-plan and time of flight for metal 3 and metal 4.

Final results show that the minimum clock cycle is the requested 4 ns plus the worst slack time achieved of -0.978 ns, which leads to a minimum required clock cycle of 4.978 ns, equivalent to a processor frequency of 200.8 MHz; Figure 3.10 shows the occurrence distribution of the single and double cycle paths in the architecture, being the ones on the left the worst slack times observed.

Table 3.10 *Results from the rounds.*

Round	Slack	Action taken	Improvement
1	-7.9ns		
2	-6.4ns	Iterate timing constraints at block level	1.5ns
3	-4.5ns	Consider pin info in constraints	2.1ns
4	-4,3ns		
5v1	-2.3ns	Push synthesis effort	2ns
5v2	-1.2ns	Identify multi-cycle paths	1.5ns
5v3	-0.75ns		
6	-1.0ns	Source files change	-0.25ns

Round	Slack	Action taken	Improvement
7	-0.82ns	Source files change	0.18ns
8	-1.1ns		-0.28ns
Final	-0.978ns	Consider interconnect delay	0.12ns

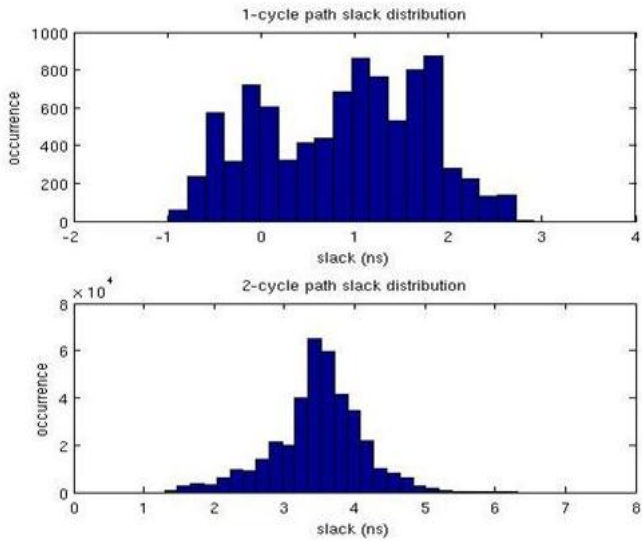

Figure 3.10 *Occurrence distributions of the single and double cycle paths in the architecture.*

Memory blocks.

Memories for this system (instruction cache, data cache, in-chip RAM and in-chip ROM) have been generated using a Memory Generator tool from Artisan which takes as input an abstract description of the memory blocks and produces several memory formats suitable for various tools and purposes. Using a memory generator instead of synthesizing a memory can optimize speed, for density and for power, can control the memory blocks aspect ratio for efficient floor planning, deliver timing and power models for integrate to other design tools, allow configurable word-write mask and redundancy options.

Memory blocks of 2 K bytes, 4 K bytes and 8 K bytes have been generated to facilitate block placement and routing in chip.

A set of views can be generated: PostScript data sheet, ASCII data table, GDSII layout file, LVS netlist, Synopsys model, PrimeTime models, TLF models, VCLEF footprint, Verilog model, and VHDL model. The Relative footprint shows how the aspect ratio of the memory changes as the words, bits, and Mux parameters are varied. The instance and the power ring are included in the footprint.

RAM architecture, timing specifications, and physical characteristics:

1. Synchronous Random Access Memory is triggered by the CLK rising edge.
2. Pins: CLK, CEN, WEN, OEN, A[m-1:0], D[n-1:0], output Q [n-1:0].
3. Memory blocks are cut in symmetrical sides to easy clock distribution and layout.
4. Dual port memories provide dual ports for all, input and output signals.
5. Power rings. Power rings can be generated around the SRAM, size them properly. Size depends on the chip-level power distribution, the number, width, and placement of supply wire connections to the power rings, and the current consumption. Recommendation: supply current evenly at the edge of the instance where the pins are located.
6. Top metal layer: metal1 to metal4 are used in the design and blocked for routing. Layers above m4 can be routed over the memory.

7. I/O pins are located along the bottom edge of the memory block on any of the metal layers, and they are large enough to accommodate a pre-determined on-grid width wire connection.

8. Verification: The views produced by the generator can be verified with standard tools.

ROM architecture, timing specifications, and physical characteristics:

1. Synchronous Read Only Memory is triggered by the CLK rising edge.

2. Pins: CLK, CEN, A[m-1:0], Q [n-1:0]. If CEN is high then memory is in standby mode and Q has last data, if CEN is low memory is in read mode and Q has data from address A.

3. Memory blocks include Row and column decoders, Clock generator, Memory array and Amplifiers/IO buffers for the outputs.

4. Power rings: multiple, evenly spaced connections have been used from core V_{dd} and V_{ss} to the rings around the instance on the side where the I/O pins are located.

5. I/O pins are located along the bottom edge of the memory block on any of the metal layers.

6. ROM code File. An Artisan format ROM code file must be provided for each generated instance.

   - Format: Code file contain only 0s and 1s.

   - The line number in the file is equivalent to (address-1).

- Each character of a line corresponds to the bits of a word. Character in column 1 of a line is the most significant bit.
- Address goes from m to 0, bits and columns go from n to 0.
- This file is needed for behavioral or physical views such as Verilog, VHDL, Tests can, Sunrise, GDSII and LVS Netlist.

Tool Verification. Views and files generated have been verified with Synopsys Design Compiler.

Before Place and Route, a FRAM view has been created for all memories in the design by importing the VCLEF and running the Blockage, Pin and Via (BPV) to create the FRAM.

3.6.4 Place and Route

P&R, the process of placing each individual block within the top level design, has a major impact in chip area: it must use the area optimized netlist for each block and use the interconnect area efficiently for routing.

Several iterations for P&R have been made due to changes in synthesis, pin placement and block aspect ratio impact placement and routing results. Integrated Circuit Compiler (ICC), from Synopsys, has been used to route signals within blocks, to place blocks within layout, to route signals between blocks, and to optimize block and pin placement for optimal routing.

A preliminary floor-plan has been delivered for power grid design. Block placement re-runs have been made with different aspect ratios for each block, until better area utilization is achieved.

About 30% of space is left between blocks for interconnect routing, clock tree and power grid.

3.6.5 Power Analysis

To estimate and analyze the power consumption for this system, a sequence of iterative tasks have been performed: estimate each block power using Design Compiler from Synopsys, include activity factors in power calculations, calculate Switching Power, Cell Internal Power and Cell Leakage power for each block, design a power grid using Prime Time PX, and integrate power grid to routed layout.

Power estimations for individual blocks are made initially from block size and gate count. Activity factors have been integrated using a code benchmark. Later, a preliminary floor-plan from ICC has been used as the base for power grid.

Vertical power lines go on M5, Horizontal on M6. No power ring has been added to ease integration with power in M3 and M4 and to power pins in IO ring. Power grid includes V_{dd}, clk, rst and V_{ss} lines, with spacing and widths: 5λ, 2λ, 2λ, 2λ, 2λ, 2λ, 5λ, 2λ respectively. Total spacing between two V_{dd} will be 22λ, which should be multiple of power grid spacing in M3 and M4 for interconnection.

Power estimation for each block has remained similar, regardless the changes in source code occurring during synthesis and integration. Power grid at M5/M6 considers block placement and individual block consumption, while power grid inside blocks on M3/M4 will be done automatically.

A verification run for obtaining activity factors by block from simulation instead that from estimations has also been made.

Dependencies for Power IR drop analysis, which is the voltage drop due to the resistance of interconnects in power network are illustrated:

- Additional code lines at top level are needed for power grid.
- IO ring needs to be hooked up to the top level design.
- All blocks should place their pins only in M3 and M4.
- The IO pads have to be hooked up to the power rail, to get info about external source of power rails.
- Modules should be power routed in M4 and hooked up to the power grid using via M4 and M5.
- De-coupling capacitor filler cells must be inserted in empty spaces: within individual modules and in the full chip level between modules.
- Design Rule Check should be executed after hooking up power routes.
- Design decision to make for each individual block: where to put the power pins for minimum route to power grid.

Results from final run of power estimation are shown in Table 3.11.

Table 3.11 *Power Consumption Values.*

	Block Activity *(Net Switching Power + Cell Internal Power) + Cell Leakage power					
	Block Activity	**Net Switching Power**	**Cell Internal Power**	**Cell Leakage Power**	**Total Power**	**Weighted value**
alu	0.4	4.8500E−03	6.1020E−03	1.0290E−07	1.1000E−02	4.3809E−03
cfgr	0.02	2.4420E−04	3.0160E−04	1.0400E−08	5.4580E−04	1.0926E−05
ctrl	0.4	1.4490E−03	3.0220E−03	7.0550E−08	4.4710E−03	1.7884E−03
dc_top	0.5	3.0410E−03	5.9600E−02	1.0080E−05	6.2600E−02	3.1330E−02
dmmu_top	0.5	1.2410E−03	9.6630E−03	6.0270E−06	1.0900E−02	5.4580E−03
du	0.01	1.7200E−02	1.9800E−02	6.5830E−07	3.7100E−02	3.7065E−04
except	0.4	3.1590E−03	1.1200E−02	1.9230E−07	1.4300E−02	5.7437E−03
freeze	0.4	2.5380E−05	6.8790E−05	1.9090E−09	9.4170E−05	3.7669E−05
genpc	0.5	3.0450E−03	5.7010E−03	1.2740E−07	8.7470E−03	4.3731E−03
gpio	0.5	7.4720E−04	2.3390E−03	1.9830E−07	3.0860E−03	1.5432E−03
ic_top	0.5	1.3880E−03	5.7500E−02	1.0040E−05	5.8900E−02	2.9454E−02
if	0.5	7.3330E−04	2.2270E−03	3.9420E−08	2.9600E−03	1.4801E−03
immu_top	0.5	2.3510E−03	1.0600E−02	6.0780E−06	1.3000E−02	6.4815E−03
iwb_biu	0.1	7.1960E−04	1.4030E−03	8.6690E−08	2.1220E−03	2.1234E−04
lsu	0.6	5.7150E−03	4.5360E−03	9.4160E−08	1.0300E−02	6.1506E−03
mult_mac	0.1	6.9460E−03	2.8100E−02	5.1140E−07	3.5000E−02	3.5051E−03
operandmux	0.4	2.2710E−03	2.6400E−03	6.4490E−08	4.9110E−03	1.9644E−03
pic	0.01	5.3930E−04	1.1050E−03	3.0240E−08	1.6450E−03	1.6473E−05
pm	0.01	2.3360E−04	4.0410E−04	9.5910E−09	6.3770E−04	6.3865E−06
rf	0.5	2.9220E−03	2.6000E−02	7.3280E−07	2.9000E−02	1.4461E−02
sb	0.2	5.6610E−04	7.5770E−04	1.5890E−08	1.3240E−03	2.6477E−04
sprs	0.02	5.5530E−03	5.2600E−03	1.3510E−07	1.0800E−02	2.1639E−04
tt	0.01	9.8770E−04	1.7000E−03	4.6980E−08	2.6880E−03	2.6923E−05
uart	0.5	4.8040E−04	2.8240E−03	2.8800E−07	3.3050E−03	1.6524E−03
wb_biu	0.1	5.0040E−04	9.5720E−04	4.9520E−08	1.4580E−03	1.4580E−04
wbmux	0.4	2.3900E−03	3.6520E−03	7.5520E−08	6.0420E−03	2.4168E−03
Total		6.9298E−02	2.6746E−01	3.5766E−05	3.3693E−01	1.2349E−01

The power consumed by electronic devices has been on a downward path for many years as a result of the hard work and creativity of talented engineers. Despite the obvious gains, the creation of lower power designs continues to be a major concern of modern engineering. There are two facets to this engineering problem. One is simply the desire to consume less power; to extend battery life and to make wall-powered devices cheaper to operate and ecologically friendlier. The other, perhaps less obvious problem, is that all power consumed must also be dissipated. Power dissipation has become more difficult as devices have become more complex yet smaller. Of course, the best way to help the dissipation problem is to consume less power in the first place. This course looks at the fundamentals of achieving the low power operation needed with nearly all of today's leading-edge chip designs.

3.6.6 IO Ring

The Input-Output pad ring on a chip acts as a communication link between the chip core and the outside world. IO pad ring is a collection of open metal areas usually located at the periphery of the chip. When a chip is being packaged a mechanical wire bonder connects the open metal surface of an IO pad with the corresponding package pin.

The circuit functions of an IO pad ring are listed below:

1. ESD protection – The IO pad ring has diode protection circuitry which protects the gates connected to the pads from any external electrostatic discharge.

2. Buffering the output signal – Usually, digital output pads have buffers to allow driving huge external world capacitances of the order of 30 pF.

3. Buffer the input signal – Digital input pads can have buffers to isolate the external input from the signals inside the core chip. A digital input pad can also have a noise tolerant functionality which removes any noise that might have coupled to the external input. A Schmitt trigger circuit is used to perform this function. The circuit generates a cleaner signal, mitigating any effect that noise might have on the circuit performance. A tradeoff of using Schmitt trigger circuits is the fact that they are power hungry and are slow.

4. Mixed voltage interface – An IO pad ring usually provides a mixed voltage interface. The external IO pads are usually running at higher voltage while the cores chip inside run at a smaller voltage, to minimize power. The IO pad ring contains Level shifter circuits that perform this function.

Due to the limitation in resolution of the mechanical wire-bonding tool, a minimum open metal surface area and a minimum pad pitch need has been maintained. Maintaining a minimum pad pitch resulted in the limitation of the number of input/output pins possible for a chip to the minimum presented ahead. Table 3.12 shows the final IO pins for the designed SoC.

Table 3.12 *SoC IO Pin List.*

Signal	From block	Signal	From block	Signal	From block
clk_i	RISC 250MHz	iwb_dat_i(31:0)	Wishbone	pm_ic_gate_o	Power
rst_i	RISC rst	iwb_cyc_o	Wishbone	pm_dmmu_gate_o	Power
clmode_i	RISC clock control	iwb_adr_o(31:0)	Wishbone	pm_immu_gate_o	Power
pic_ints_i(3:0)	PPIC interrupts	iwb_stb_o	Wishbone	pm_tt_gate_o	Power
iwb_clk_i	Wishbone	iwb_we_o	Wishbone	pm_wakeup_o	Power
iwb_rst_i	Wishbone	iwb_sel_o	Wishbone	pm_cpu_gate_o	Power
iwb_ack_i	Wishbone	GPIO(31:0)	GPIO	pm_1volt_o	Power
iwb_err_i	Wishbone	pm_cpustall_i	Power	pm_clk_sd_o (3:0)	Power
iwb_rty_i	Wishbone	pm_dc_gate_o	Power		

The design of the IO ring for this system has been made based on the needed pins for this specific application and has been integrated later to the complete and routed layout. These pads are available ready-made in TSMC's digital IO pad library. These pads have been piled together in a rectangle to create the pad ring. The open metal area surface of the pad has a 50um length. The pad length itself is 70um. To keep sufficient spacing between two metal area surfaces - where the wire-bonder would come to attach the bonding wires-, a spacer of 10um was inserted between each pad. This increased the pad pitch to 80um.

The input pads were chosen without Schmitt trigger functionality (PDIDGZ) because Schmitt trigger circuits are power hungry and slower. Only general purpose IO pads (PRU08SDGZ) had schmitt trigger circuitry inside them. They also had control enable based input/output configuration functionality. The output pads were chosen based on the current driving capability required from the pad. The average current was estimated for the pad assuming a 30pF load and a 25% rise time at 62.5MHz. Based on the average current calculation for some of the pads, the average current requirement was 4.8mA. An output pad (PDO16CGZ) with a current capability of 16mA was therefore chosen to safely meet the current requirements. 3.11 shows a close up to the physical IO ring.

To find out whether the area was pad limited or core chip limited, a very conservative area estimate was made by adding the block areas and multiplied it by double to account for wire routing overheads. The area assuming a square came out to be 0.583976mm^2 (0.7mm x 0.7mm) – which denoted severely pad limited die size.

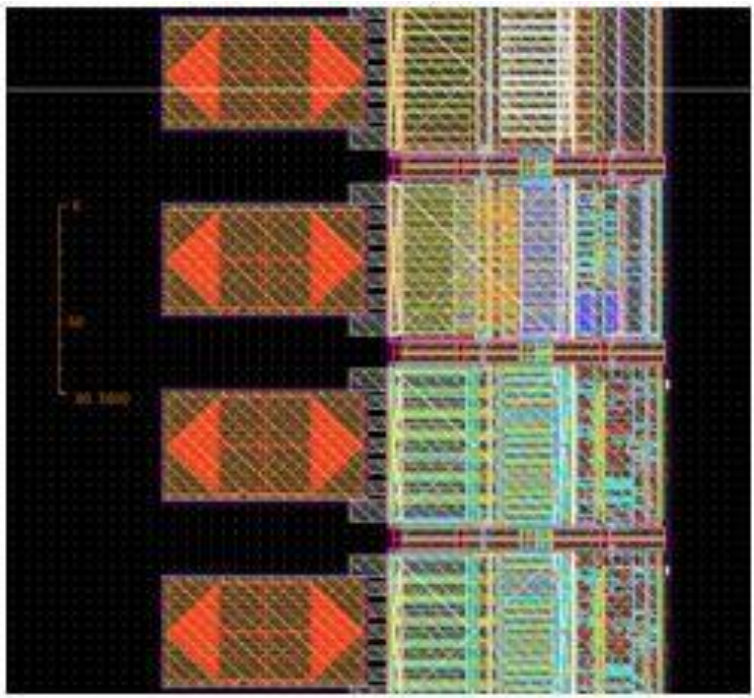

Figure 3.11 *Close-up view of the IO pad ring, pad pitch is 80nm.*

Number of V_{dd} - V_{ss} pair pin calculation:

There are two power supply voltages on the chip. The first is 3.3V volt which is used for input/output signals from the external world to the IO pads. The other voltage is 1.8V which is the V_{dd} for the core chip. The IO pad ring performs the task of converting the voltage levels.

An adequate amount of V_{dd} - V_{ss} pins is needed to allow sufficient current source and sink-in capability. The more the number of IO pads the larger the number of V_{dd} - V_{ss} pins. For this design a 3.3V V_{dd} - V_{ss} pair for every 8 IO pins was chosen. While two 1.8V V_{dd} - V_{ss} pairs were placed on each side of the die.

There are more 3.3V V_{dd} - V_{ss} pairs due to the amount of power dissipation of the IO pads needed to drive the external 30pF buffers. The core chip itself requires less power so fewer IO pins were dedicated to it.

3.6.7 Clock Tree Synthesis

Clock tree synthesis (CTS) is a separate design process which consists on building a balanced buffer tree from clock input pin to all clock sinks in the design blocks.

Clock design includes clock generation, clock regeneration and clock distribution. Tree design, leafs, sinks and location have been set according to this chip needs. The input files needed for the clock design are top level DEF file and top level Verilog net-lists.

To do clock tree synthesis SOC Encounter from Cadence has been used. When starting from a placed net-list, the flow is to perform CTS, do global routing and block level routing.

Clock Tree design is critical for system synchronization: if clock does not arrive on time to each block depending on its location within the data-path, all instruction flow gets wrong.

Clock distribution and Clock pin placement are design placement dependent, so every new place and route run requires a new clock tree design.

Figure 3.12 shows a sample run of the clock tree generation:

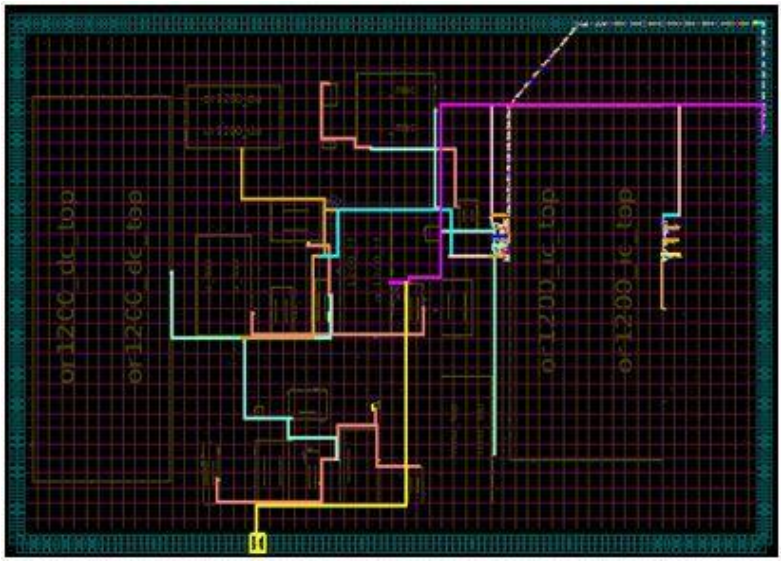

Figure 3.12 *Sample run of the clock tree generation.*

Hardware design in high-performance applications such as communications, wireless infrastructure, servers, broadcast video, and test and measurement equipment is becoming increasingly complex as systems integrate more functionality and require ever-increasing levels of performance. This trend extends to the board-level clock tree that provides reference timing for the system. A "one size fits all" strategy does not apply when it comes to clock tree design. Optimizing the clock tree to meet both performance and cost requirements depends on a number of factors, including the system architecture, integrated circuit (IC) timing requirements (frequencies, signal formats, etc.) and the jitter requirements of the end application.

3.6.8 Integration

Integration work is an inter-dependent task, since results from power, timing, placement, and routing affect other results, so iterations have been done until

satisfactory results in all areas have been reached. First designs for power grid and clock tree have been done based on the first preliminary floor-plan delivered. As part of integration, different versions of definers' file, one from verification, one from Timing and one from Place & Route have been merged into one common file to check and eliminate inconsistencies.

When cores have not been completely compatible with the bus based architecture, wrappers have been needed, since cores are generic open source code and customization is needed, especially for hardware integration and in-chip communication. A Wishbone compliant wrapper has been added for: an 8K SRAM for the memory module, a 4x8 bit GPIO core, and a standard UART. When synthesizing at top level some signal and bus inconsistencies arose: missing pins, incompatible bus widths, and unreferenced instantiations, among others.

Most Verilog sources have gone under editing and current control version is maintained for code consistency since it is critical for integration.

When integrating at chip level, hardware hierarchy has been redefined: block level is the open source for each individual block; CPU level is the unit build by interconnecting the individual blocks; OR1200 level is the processor built with CPU, memories, debug, and system units; SoC level is the system built with OR1200 processor, SRMA, ROM, GPIO, UART, power grid, and clock tree.

Tables 3.13 and 3.14 show details of the tasks performed and results obtained during the last two rounds of integration.

Table 3.13 *Tasks and results, 7th round of synthesis and integration.*

Cluster	Tasks/Problems/Results
P&R	Preliminary floor-plan has been delivered for power grid design. To be final it needs: resize IMMU cache, consider 30% of interleaving space for routing, clock and power lines. Do block placement re-runs with different aspect ratio for each block, until better area utilization is achieved.
Timing	Worst slack time at -800ps over a 4ns period. That leads to a frequency of 208 MHz.
Top level synthesis	Synthesis script and netlist for OR1200 top level: ready. Synthesized OR1200_top.v, net-list generated. To do: re-synthesize to be consistent with updated defines file, due to inconsistencies in source files.
Integration	Different versions, one from verification, one from Timing and one from Place & Route of definers' file should merge into one common file. Example inconsistencies: "OR1200_ARTISAN_SSP" has been commented out, `define OR1200_ASIC is commented out in one version, FPU related macros to be removed.
Verification	Verification is complete with the new Verilog files, re-do when definers' file is common to all clusters. Use SAIF from Synopsys (1st option, for tool compatibility) or VCD from Model-Sim to write out activity factor, for a given timing window of simulation. SAIF: forward_saif file required for Model-Sim to generate a backward_saif, the required output file
Verification for Power	VCD: To write out VCD: Read DC synthesized net-list into Model-Sim. Write out a VCD from Model-Sim from the designed test bench using time window which power numbers are going to be generated; zip VCD: "zip -r DESIGN.vcd.zip DESIGN.vcd" To convert between formats: Invoke Synopsys DC and use "vcd2saif" command
Source	Wrappers are written for SRMA, GPIO, and UART. These blocks, along with OR1200, will build the SoC top level. UART and GPIO connect to the data wishbone of the processor. New pins created for GPIO wrapper: *aux_i, ext_pad_i, ext_pad_o, ext_padoe_o, clk_pad_i,* For UART: i*nt_o, stx_pad_o, srx_pad_i, rts_pad_o, cts_pad_i, dtr_pad_o, dsr_pad_i, ri_pad_i, dcd_pad_i* SRAM: Wishbone compliant wrapped 8K SRAM memory module. No syntax errors. Needs functional verification, run design compiler to get the gate level Verilog.

Table 3.14 *Tasks and results, 8th round of synthesis and integration.*

Cluster	Tasks/Problems/Results
P&R	Re-run the global P&R scripts, since the floor-plan has changed significantly. All of the objects are placed. Rebuild each using synopsis/icc/2010 and make the OR1200_top library.
Power	As Verilog source files change, new SDC files for the new net-lists are needed for each block.
	After generating sdc files for each block, check for consistency among the gate.v net-list and new sdc file. Upload new files in the corresponding folder because existing scripts take them from there. Results from last run of power analysis are shown in a table below.
	Needs from Global P&R for Power IR drop analysis - Due to the resistance of the interconnects in power network, there is a voltage drop -:
	Add code lines for power grid
	IOs still not hooked up in the milky way
	Check all blocks place pins in M3 and M4.
	Hook up the IO pads to the power rail, to get info about external source of power rails
	Modules should be power routed in M4 and hooked up to the power grid using via 4-5
	Insert de-coupling capacitor filler cells in empty spaces: within individual modules and in the full chip level between modules
	Design Rule Check should be executed after hooking up power routes
Timing	Final run: squeeze out as much extra timing as possible. Reference to: For each pass-through in a critical path, a log has been used to show the average path slack, average time for the pass-through, and the number of times that pass-through appears in a critical path. Locate the pass-through routes in each block that occupy a good amount of time in a critical path to tighten while also locating a path with slack to give that you can relax. Due to changes in defines file between r6 and r7, the timing became a little worse, now at 200MHz. Some pins of debug unit and control unit are causing the problem. These pins pop up only in this release, probably due to changing of the define file.
Integration	Design decision to make: where and how to put the power grid on blocks?
Source	Defines file at processor top level was modified to meet requirements from: cache memory blocks (`define OR1200_ASIC and `define OR1200_ARTISAN_SSP 0), register file block (Type of register file RAM: `define OR1200_RFRAM_GENERIC), wishbone bus (OR1200_CLKDIV_4_SUPPORTED, This will allow us to use 50 MHz for the external wishbone bus.), Power management unit (`define OR1200_PM_IMPLEMENTED), and eliminated references to floating process unit since it is not implemented.

3.7 Design Evaluation

This system design was completed through the stages of the design. The hardware architecture was synthesized with Synopsys Design Compiler using TSMC Physical Libraries for 180μm technology. RAM and ROM arrays were

built with Artisan Memory Generators. Place and Route, Clock Tree synthesis and IO Ring design were realized with Synopsys IC Compiler. Power analysis and power grid design was made with Prime Time-Px.

Source cores -Processor, Peripherals, and on-chip Communication Bus- are synthesizable cores from OpenCores.org.

Here are presented the resulting design parameters:

Timing Analysis was made during synthesis process. Timing parameters like system clock frequency, bus clock frequency, and port clock frequency are implementation dependent only. Output update rate considers n data samples per waveform cycle. Table 3.15 shows operating frequencies per block group.

Table 3.15 *Operating Frequencies.*

Concept	Operating Frequency
System Clock	190 MHz
Wishbone bus clock	47.5 MHz
GPIO, 8 bit data L&S	5.94 MHz
Output data update rate	(48/n) MHz

Execution times were obtained from simulations of the application software and from actual execution times on a development board –the LM3S6965-. Application functions are grouped to present their execution times: Parameter set up function gets the operation parameters from user, and generates related data; Data processing function creates temporary and output table generation processing two waveforms -frequencies f_1 and f_2 -, 32 samples each.

An example where $f_2 = 4f_1$ is used. Output update function refers to signal generation - sending waveform data samples to output port-, and corresponds to 1 cycle for addressing, 1 cycle for load from memory, and two cycles for store

in parallel port. Execution times marked with * do not impact maximum output frequency since they are executed during system configuration or data processing, i.e. before signal generation begins.

Table 3.16 *Execution Times.*

Function	Clock cycles	Exec time (ns)
Parameters setup, 8 parameters, 4 cycles each	4 x 8	168
Data processing for temporary tables	2 (4 x 32)	1347
Data processing for output table	$5(32)(f_2/f_1)$	3368
Output update	4	21

Delivered Signals The system can deliver any mix of sine, triangle and saw-tooth waveforms with different frequencies in single or superimposed patterns.

Power analysis Power analysis was made for individual blocks then grouped for simplicity. Power values in Table 3.17 consider full throttle operation for that group of blocks and % of total power is calculated considering activity factor.

Table 3.17 *Power Consumption By Block Group.*

Block	Power, Watts	% of total power
Processor	0.176965	52.3
Memories	0.145400	42.97
Peripherals	0.006385	1.887
Bus	0.009620	2.843

Table 3.18 *Area Use By Block Group.*

Block	Area, (mm^2)	% of chip area
Processor	1.4866	25.73
8K IC RAM	0.9824	17.01
8K DC RAM	0.9736	16.85
256 bytes ROM	0.0212	0.37
Peripherals	0.0522	0.90

Bus & Interconnect	1.4694	25.44
IO Ring	0.7912	13.70

Areas The minimum total and interconnect areas were achieved by varying the aspect ratio of major blocks: memory and processor cores. Area use per block group is shown in Table 3.18.

3.8 Application Software

Based on the hardware specifications, the application software is developed including start up and load procedure, sine/triangle/saw tooth waveform's data storage, configuration and operation setting, and frequency synthesis process using the novel methodology created for this application.

Separate application software versions for development board and IC are kept due to differences in hardware resources.

Board version is an extended functionality version where hardware resources are only limited by the board features.

The SoC version is called the standard version: it has less functionality than the board's due to area budget, processor, and open source restrictions.

3.8.1 Program Flow

Table 3.19 shows a simplification of the application flow by listing only the main tasks; secondary tasks as data protection, data scaling and frequency synthesis details are not shown.

Table 3.19 *Application Program Steps.*

1. Configure General Purpose I/O port
2. Get configuration and operation parameters
3. Calculate time separation between data and shift factor for data extraction
4. Prepare temporary and output tables for requested mode
4.1 Mode 1: 1 8-bit table, single data, for 1 frequency signal output
4.2 Mode 2: 1 16-bit table, inter lapped double data, for two frequency signal outputs
4.3 Mode 3: 1 8-bit table, superimposed data, for two frequencies on 1 signal output
5. Configure four 8-bit GPIO port for output signals.
6. Send data samples for sine signal to output port, with a load-from-table/store-to-port cycle
7. Update output port continuously while waiting for stop signal
8. Restart operation to get new configuration parameters

3.8.2 Standard Version

A standard version of the application program has been developed to run on OR1200 based architecture; this version is the foundation for the SoC design and implementation. The three waveforms can be delivered to output channels and dual superimposed frequencies are included if frequencies are exact multiples, due to in-chip memory limitations. To be interrupted during operation using master reset only. Maximum output frequency is limited by the timing constraints of the available physical libraries for implementation and the timing optimization of the set of processor source files. Standard version takes less than 1 Kbytes of data RAM and less than 2 Kbytes of instruction RAM.

Table 3.20 shows a description of the routines in the standard version.

Table 3.20 *Routine List and Description, Standard Version.*

Routine	Description
Calculate Data Separation	Desired output frequency and number of voltage steps determine how many data points will be extracted from the original sine table in order to construct desired output signal. Two data separation parameters are needed for operation modes 2 and 3.
Calculate base time	According to desired frequency and number of voltage steps, there is a base times that indicates the time between data points are sent to output port.

Routine	Description
Create Table-Mode1	Extract data from original sine table needed to construct 1 output signal, 1 single frequency: each 8-bit data from original sine table is stored as the 8 least significant bits of the 32-bit output port.
Create Table-Mode2	Extract data from original sine table needed to construct two output signals, two separated frequencies: if two 8-bit output ports can be stored at the same time with 32-bit data, two data points from original sine tables must be concatenated before stored in buffer memory table.
Create Table-Mode3	Extract data from original sine table needed to construct 1 output signal, two superimposed frequencies: data points for different frequencies should be added to achieve superimposition.
Output Signal Generation	Continuous, uninterrupted loop, for loading data from buffer memory and storing it on output ports. No memory other than buffer is read, no instructions other than those for signal generation are executed.
Start/Stop external interrupt	Start/Stop button is enabled as an external interrupt in two execution moments: at startup to be ready for accepting configuration and operation parameters, and during Output Signal Generation routine to stop signal.
Timer interrupt generation	Within Output Signal Generation routine: When low frequency output is desired, a timer is used to update data to outputs at a base time determined by Calculate Base routine. For high frequencies time is achieved by cycle and instruction count.

See Appendix A1 for Application software C code, standard version.

3.8.3 Extended Version

The extended version of the application program was developed to run on an ARM9 or Cortex-M3 based development board. A functional implementation of this version is presented in chapter 5. Extended functionality is added, such as delivering data via an USB port for further analysis of monitored or stored data, mixing different waveforms in a superimposed signal for more controlled experimental environments and an interactive user interface for configuration and operation. Data tables for the three signal wave forms (sine, triangle and saw tooth) can be displayed at start-up for demonstration purposes of the novel frequency synthesis methodology to show the frequency superposition effect. Extended version takes less than 4 Kbytes of data RAM - 1 Kbytes for base data and 3 Kbytes for temporary and final data-, and less than 3 Kbytes of instruction

RAM. Although the Extended version has additional functionality it fits in the original SoC design which has separated RAM blocks of 8KB of data RAM and 8KB of instruction RAM.

See Appendix A2 for Application software in C code, extended version.