CHAPTER 3

MOTIVATION AND LITERATURE REVIEW

3.1 MOTIVATION

In recent years, power dissipation, scheduling the optimal solutions has become an increasingly critical issue in IC design. The optimal design of hardware with low power, faster performance and less chip area is the most important requirement for embedded systems. Due to low power requirements in many portable applications such as mobile phones, as well as packaging cost consideration, low power design is essential. Technical feasibility of high performance computation is due to heat extraction. It is determined by total number of control steps (clock cycles) a system takes, which is the clock period of the slowest logic stage in the system. Minimization of chip area reduces the amount of silicon used. This also increases yield since the causes of failure like crystal defects, defects in the masks, defects due to contact with dust particles, etc are less likely to affect a chip when area is smaller. Larger power consumption of FPGA chips is a constraining factor for FPGA designs to enter main-stream low-power applications. This research work aims to optimize the control steps and silicon area.

The reconfigurable computing technology is relatively new and there is a dire need for CAD tools that support the automated design flow for such architectures. The state-of-the-art design flow for RCs begins at the behavioral level of abstraction and HLS forms a central part of the design flow. Here present important issues related to high-level design optimization factors like speed, area and power.
In this research uses the different novel techniques to get optimization via clock cycles, power and area. In the first phase, the idea is implemented using the bus matrix AMBA AHB architecture and in the second phase, the integration of scheduling algorithm is used to optimal solutions.

3.2 LITERATURE REVIEW

3.2.1 Force Directed List Scheduling (FDLS)

Force Directed List Scheduling (FDLS) algorithm produces shorter schedules than List Scheduling (LS) (Su & Chakrabarty 2008) and Path Scheduling (PS) (Verhaegh et al. 1995) in some cases. LS computes a priority function for each vertex in the DAG. Vertices that can be scheduled immediately (sources) are sorted in priority order using a priority queue. The main loop of the algorithm steps through the schedule, one time-step at a time, until all vertices are scheduled, or it becomes apparent that no feasible schedule is possible, given the resource constraints. At a given time step, a vertex is available if all of its predecessors have completed their operations.

Force Calculation: The first step is to determine the set of all time steps at which operation can be scheduled. In the resource-constrained context, it is NP-complete to determine if a legal schedule of length K can be found, given a pre-allocated set of resources (Micheli 1994).

3.2.2 Classical Force-Directed Scheduling (CFDS)

The Force-Directed Scheduling (FDS) algorithm, widely used in high-level synthesis, is an example of a constructive approach for scheduling. The constructive algorithm builds a schedule by adding new operations to the existing schedule in each iteration. In this approach for instruction scheduling is based on the work by (Paulin & Knight 1989). FDS is a latency-constrained algorithm. The intent of the FDS, which aims to reduce the resource
requirements by balancing the concurrency of the operations assigned to them without violating the latency constraints. The algorithm is iterative, with one operation scheduled in each iteration. In this version of the algorithm, all the operations are assumed to execute in a single control step. This work describes a new instruction scheduling algorithm based on the classical force-directed approach. This modified the FDS to optimize power through instruction scheduling.

The advantage of using the FDS is that it tries to find a globally optimized solution. In each iteration of the algorithm, FDS takes into account the global effect of every tentative assignment and thus avoids getting trapped into a locally optimized solution. Most of the existing instruction scheduling techniques for power optimization use list scheduling approach. List scheduling heuristic suffers the shortcoming that it can be trapped in local minima of the cost function and therefore may not find globally optimal solution. This motivates us to use the FDS, which promises globally optimal solution. It should be noted that this type of approach for instruction scheduling is based on the force-directed approach, however the model itself is quite different in terms of its parameter modelling, their role in the force equation and also the level (i.e. compiler-level) at which the model is applied.

3.2.3 Low Power Force-Directed Instruction Scheduling (FD-ISLP)

The new instruction scheduling method for power optimization called Low-Power Force-Directed Instruction Scheduling (FD-ISLP). FD-ISLP is a latency constrained algorithm that reorders the instructions in a basic block so that the total power consumption of the basic block is reduced. Given an assembly code, it is divided into basic blocks and a data dependency graph (DDG) is constructed for each of the basic blocks. SimplePower is used for power characterization of the ISA to obtain a power dissipation table (PDT). The FD-ISLP takes as input, a DDG for the basic block to be
scheduled for low power and the PDT. FDS has been typically applied for operation scheduling in high-level synthesis. FDS at the compiler level for reordering instructions for power optimization, whereas the original FDS targets at the resource optimization. The FDS is a resource optimization algorithm, which tries to reduce the resource requirements of a design. Therefore, FDS models the resources as a force to be optimized. Resource optimization is done by increasing the operation concurrency, which is the ability to schedule the similar type of operations in the same control step.

Input: Data dependency graph, DDG and power dissipation table, PDT
Output: Low power schedule, LP-schedule

(01) Algorithm FD-ISLP(Graph DDG, PDT)
(02) Begin
(03) ASAP-schedule(G)
(04) ALAP-schedule(G)
(05) unscheduled ← total-inst(G)
(06) for each inst ∈ G
(07)  Determine-time-frame(inst)
(08)  if (inst.time-frame = 0) then
(09)   LP-schedule[inst ASAP] ← inst
(10)  unscheduled ← unscheduled - 1
(11) end for
(12) while(unscheduled) do
(13)   for each inst ∈ G
(14)    if (inst.scheduled = FALSE) then
(15)      for each tstep ∈ time-frame(inst)
(16)       Make a tentative assignment(inst,tstep)
(17)      Compute self-force(inst,tstep)
(18)      Compute ps-force(inst,tstep)
(19)      total-force(inst,tstep) ← self-force(inst,tstep) + ps-force(inst,tstep)
(20)    end for
(21)  end for
(22)  min-force(inst,tstep) ← minimum-total-force(inst,tstep)
(23)  LP-schedule[tstep] ← inst
(24)  inst.scheduled ← TRUE
(25)  Update-ASAP(G,inst)
(26)  Update-ALAP(G,inst)
(27)  Compute time frames
(28)  unscheduled ← unscheduled - 1
(29) end while
(30) return LP-schedule
(31) End

Figure 3.1 FD-ISLP algorithm
3.2.4 Scheduling and Genetic Algorithms

Genetic algorithms are probabilistic search algorithms. Given an optimization problem they try to find an optimal solution. Genetic algorithms start by initializing a set (population) containing a random selection of encoded points of the search space (individuals). By decoding the individual and determining its cost the fitness of an individual can be determined, which is used to distinguish between better and worse individuals. A genetic algorithm iteratively tries to improve the average fitness of a population by construction of new populations. A new population consists of individuals (children) constructed from individuals of the old population (parents) by use of re-combination operators.

During high-level synthesis the behavioral descriptions of a chip is often represented by a data-flow graph. In a data-flow graph nodes represent operations and control structures, and edges model flow of data. In the following Figure 3.2 an example of a data flow graph of a fast discrete cosine transform filter (Mallon & Denyer 1990) can be found.

(Source: Mallon & Denyer 1990)

Figure 3.2 Fast discrete cosine transform filter
Most scheduling methods based on genetic algorithms don’t regard feasible schedule ranges directly. Ignoring these constraints would lead to generic search strategies that would evaluate in-feasible schedules most of the time. Scheduling with genetic algorithms will finally lead to an efficient time constrained scheduler with fast execution times and good quality results.

A genetic search strategy for scheduling (Heijligers & Jess 1995) which doesn’t use schedule specific knowledge gives poor results. An efficient constructive scheduler based on topological sorting combined with genetic search techniques and incorporation of schedule specific knowledge like lower bound estimations, constraint satisfaction and the possibility of allocation of extra resources has resulted in a time constrained scheduler which offers high quality results and fast execution times. A proves has been given that the search space of the genetic scheduling algorithm contains the optimal schedule.

Comparison with other high-level synthesis scheduling methods show that these schedulers offer better results than other existing heuristics. Extensions, like the incorporation of register costs can be easily made, and give good results. Although the scheduling technique does not guarantee optimality, the algorithm produced optimal results for all examples tested in the different levels of resource and time constrained scheduling algorithm.

3.2.5 FPGA based Multi-core Architecture

The current trends towards the integration of embedded systems for cost and power savings (Daniel & Voit 2014). Consumer electronics such as mobile devices follow this trend. The partitioning techniques are normally applied to integrate multiple critical software functions in a single embedded platform. As the overcome of the existing techniques and solutions, this concept shows a new proposed methodology present robust, reliable and
efficient architecture with support of safety-and security-critical embedded systems. FPGA technology follows Moore’s Law and we can expect FPGAs to grow significantly in logic capacity. Devices with over one million Lookup Table (LUTs) are available. Luckily, current FPGA families provide wide and fast memory attachments, mostly implemented as hard macros that are faster than configurable logic. To overcome this performance gap, this architecture that combines the specific needs of partitioning software with the flexibility of reconfigurable hardware.

3.2.6 Synthesizable SystemC Benchmark for HLS

High-Level synthesis has evolved significantly over the last decade and the QoR of commercial HLS tools has improved to the level where HLS has begun to be used for commercial designs (Benjamin Carrion & Anushree 2014). The adaptation of HLS as part of standard VLSI design flows has led to the proliferation of HLS tools. The main problem faced by many designers, wanting to transition from traditional RTL to C-based design, is the absence of validated standards to evaluate the different HLS tools available (a good comparative review of these can be found at (Meeus et al. 2013). The evaluation phase is crucial in order to find the best HLS tool for the type of applications being designed, but the lack of expertise of most RTL designers in HLS combined with busy schedules makes it hard to set up an efficient evaluation methodology. All designs were successfully synthesized using a commercial HLS tool for validation. S2CBench is mainly targeted for designers wanting to evaluate different commercial HLS tools, as all of main commercial HLS tools support SystemC’s synthesizable subset. The test cases have been carefully chosen to represent different applications do mains amiable to HLS and each of them serves to test the extension of the language support, specific synthesis optimizations and tool performance.
3.2.7 State-of-art-of High-Level Synthesis for FPGAs

AutoPilot is one of the most recent HLS tools, and is representative of the capabilities of the state-of-art commercial HLS tools available today. AutoPilot accepts synthesizable ANSI C, C++ and OSCI SystemC (based on the synthesizable subset of the IEEE-1666 standard) as input and performs advanced platform based code transformations and synthesis optimizations to generate optimized synthesizable RTL. AutoPilot outputs RTL in Verilog, VHDL or cycle-accurate SystemC for simulation and verification. To enable automatic co-simulation, AutoPilot creates test bench wrappers and transactors in SystemC so that designers can leverage the original test framework in C/C++/SystemC to verify the correctness of the RTL output. In addition to generating RTL, AutoPilot also creates synthesis reports that estimate FPGA resource utilization, as well as the timing, latency and throughput of the synthesized design. The reports include a breakdown of performance and area metrics by individual modules, functions and loops in the source code. This allows users to quickly identify specific areas for QoR improvement and then adjust synthesis directives or refine the source design accordingly.

Finally, the generated HDL files and design constraints feed into the Xilinx RTL tools for implementation. The Xilinx ISE tool chain and Embedded Development Kit (EDK) are used to transform that RTL implementation into a complete FPGA implementation in the form of a bitstream for programming the target FPGA platform.

3.2.8 Optimization using Soft Constraints

When the schedule is optimized for operation gating, along with other objectives such as latency, different algorithm frameworks can be used. As the problem is intrinsically difficult even without the consideration of
operation gating, it is often solved using heuristics like list scheduling or force-directed scheduling. The postprocessing technique and the approach of (Chen & Sarrafzadeh 2002) can be viewed as natural adaptations of previous heuristics to the problem with consideration of operation gating.

Here propose to use integer-difference soft constraints to express the design intention for operation gating, and rely on the scheduling engine described to make global tradeoffs among operation gating and other design intentions. Recall that in the scheduling formulation, every operation \( v \) in the CDFG is associated with an integer-valued scheduling variable \( s_v \). When it is preferred that a Boolean value \( c \) is computed before another value \( v \) so that \( v \) can be avoided when \( c \) takes a certain value, an integer-difference soft constraint can be added as

\[
s_c - s_v \leq -b - d_c + 1
\]

where \( d_c \) is the number of clock cycles operation \( c \) spans, and \( b \) is the number of clock cycles needed to separate operations \( c \) and \( v \). The value of \( b \) depends on the power management technique and the target platform: a typical value of \( b \) is 1 if clock gating or operand isolation is used; it means that the condition should be available at least 1 cycle before it can be used as a predicate for clock gating. For power gating, the number \( b \) is probably larger than 1.

3.2.9 Thermal Optimization Techniques

Various thermal optimization techniques and their effects are categorized as below.

Among the stages of the HLS that includes scheduling, allocation and binding, scheduling indirectly controls the temperature of the FUs;
binding determines the activity of the FUs thereby affecting their thermal profile. Both these stages are necessary for efficient thermal management (Yen & Chu 2002). (Rajashi & Seda Ogreneci 2006) presented an integrated approach to thermal management in architectural synthesis (Rajashi & Seda Ogreneci 2006). The normal HLS flow is unaware of the target technology. The normal HLS flow is modified to become a thermal aware flow.

Rajashi & Seda Ogreneci (2006) proposed techniques to incorporate temperature awareness into HLS. Temperature need to be taken into account at all levels of abstraction of the design process. The authors developed temperature aware resource allocation and binding algorithms to be deployed with HLS, to accomplish this goal. A temperature aware binding controls the maximum temperature reached in a design more effectively, thereby minimizing the occurrence of hotspots. Temperature has an additive nature i.e., temperature at a given point in time will depend on the entire history of activity in the past. If this is not taken into account, cumulative heat will cause hotspots although the average activity seems to be well bounded (Rajashi & Seda Ogreneci 2006).

Peak local temperature influences the reliability, packaging costs, cooling costs, bulk and performance of IC. These considerations are important for the portable devices. Power and thermal variations can also lead to significant timing uncertainty, requiring more conservative timing margins, thereby reducing performance (Gu et al. 2006). Voltage islands enable core level power optimization for System-on-chip designs by utilizing a unique supply voltage for each core (Jingcao et al. 2004).

The thermal optimizations that could be achieved by incorporating thermal awareness into the High level as well as Physical level flows and these are Slack distribution, Voltage partitioning, Thermal aware floor planning.
As there is technology scaling, designs with multimillion transistors is becoming common. This increase in the number of transistors with the limited silicon area causes a number of crucial design problems to be solved. One of them is power consumption that leads to the increase in the chip temperature. Since different components of a chip can have different execution profiles, the temperature of the components of chip are not uniform. Hence the designer has to design the chip in such a way that hotspots do not appear. The proposed iterative thermal aware flow minimizes the maximum switching activity of modules. It includes the following steps:

3.2.10 System-on-Chip Design using High-Level Synthesis Tools

The hardware considerations that software engineers need to apply when designing hardware modules using HLS tools. Fast Fourier Transform (FFT) implementation in ANSIC C is examined in order to explore the important design issues such as concurrency, data recurrences and memory accesses that need to be resolved before generating the hardware using HLS tools. The objective of this design is to generate the hardware of a FFT (Fast Fourier Transform) block based on the reference C code using HLS tools. Multiple modifications are needed in order to generate an optimal hardware in term of resource usage and throughput. As an example, it generates an 8-bit 1024- point radix-2 FFT. The output is on 18 bits and will be available in natural order. The size of the data width inside the FFT has been chosen so that the HLS FFT gives the same results as the Xilinx FFT core.

Since the reference C code is using floating point numbers, a fixed-point library is needed. For example, PICO, the HLS used in this demonstration, provides such library. The PICO fixed-point arithmetic library derives its semantics from the SystemC fixed-point library and it supports signed and unsigned arithmetic operations.
3.2.11 Extend Force-directed Scheduling for System-Level Synthesis

Scheduling time-constrained task graph to minimize resource requirement is a common and important problem in system-level synthesis (SLS) for system-on-chip (SoC) designs. System-level Synthesis (SLS) is the key step in SoC design where systematic decisions are made, including hardware-software partitioning, resource allocation, task scheduling and task-resource binding. For time-constrained applications, scheduling will always be performed in the SLS to minimize the resource requirement with all deadlines met. Usually, this is performed after the hardware-software partitioning to have the resource type mapping and execution time of tasks determined. Intensive research has been carried out to address this issue. Most of the proposed algorithms are based on the list scheduling to achieve a relatively quick generation of the result.

To apply FDS in SLS, the notion of time is the first concern. In HLS, the time has been divided to discrete intervals, called control steps. The resource requirement distribution function and the force function are hence defined and calculated on these discrete intervals. In SLS, the notion of time is not suitable to be restricted to discrete intervals, since the granularity in SLS is higher than that in HLS. A task at system-level may consists many operations at high-level and spans millions of clock cycles. The communications between system-level tasks are always considered asynchronous. It is not applicable to count cycles or control steps in SLS for timing. So the notion of time at system-level is always continuous rather than discrete at high-level. The second concern to apply FDS in SLS is the calculation of resource requirement distribution and the related force function under the notion of continuous time. The force-directed heuristic to the hardware-software partitioning and co-synthesis respectively, but restrict themselves on the task partitioning and assignment problems. In addition,
retains the concept of control step which is not natural in SLS, and can follow the force definition and calculation in FDS without substantial extension. Generalized force-directed heuristics uses the force to measure the possibility of each processing element on which the tasks are to be assigned. Since the number of processing elements is limited in the whole system, the force function hence defined is still in a discrete manner.

In SLS for SoC design, scheduling is always performed on the task graph, which is the intermediate representation for the system functionality. Time constraints are represented with the deadlines imposed on the task graph.

3.2.12 Bitwidth-Aware Synthesis Flow

The gap between design productivity and complexity continues to grow larger. According to ITR, the increasing rates for VLSI complexity and design productivity are 58% and 21% per year, respectively. This large gap must be shortened to satisfy the time-to-market and design cost requirements. Using high-level languages is one of the most promising solutions for improving design productivity by raising the level of abstraction. Alleviating the design complexity and producing high-quality products are the two key points for making high-level languages successful and accepted by designers.

Many high-level description languages, such as C/C++ or Java, lack the capability to specify the bitwidth information for variables and operations. Synthesis from these specifications without bitwidth analysis may introduce wasted resources. Furthermore, conventional high-level synthesis techniques usually focus on uniform-width resources, thus they cannot obtain the full resource savings even with bitwidth information. This work develops a bitwidth-aware synthesis flow, including bitwidth analysis, scheduling and binding, and registers allocation and binding, to exploit the multi-bitwidth
nature of operations and variables for area-efficient designs. The proposed bitwidth-aware high-level synthesis flow, which is composed of four steps. First, a behavioral description in C is transformed into the Machine-SUIF intermediate representation. After compilation optimizations, such as dead code elimination and peep-hole optimization are applied, the bitwidth analysis is performed as a stand-alone Machine-SUIF pass. The goal of the bitwidth analysis is to automatically decide the minimum bitwidth for each variable and operation while retaining the program correctness. We adopt the bitwidth analysis method introduced, which uses constants, array indices and computation to decide the minimum bitwidth. The output is a DFG annotated with bitwidth information.