Project 22703: SYRF

Deliverable 6.2.3

Mixed Controller/Data-path Design

---

1 Approach

Thanks to results obtained in previous periods (6.1.2, 6.2.2), two domains have been explored this year. The first one is *High Level Pre-Synthesis* and the second one is *Functional units assignment under constraints during data path generation*. Our activity in the latter is prolongation of these taken in the first one.

In the context of the first activity we have proposed a new methodology based on SIGNAL internal representation which is HCDG *Hierarchical Conditional Dependency Graph* for improving synthesis process. Combination of pre-synthesis and powerful HCDG representation is a principal novelty of our approach. HCDG offers the following advantages:

- unification of control and data representation (leading to insensitivity to syntactic variance),
- extensive and uniform identification of mutual exclusiveness,
- possibility to eliminate a higher number of false paths,
- formally proved graph optimizing transformations thanks to the formal semantics of the HCDG,
- more efficient scheduling of conditional behaviors by either ameliorating the performance of standard algorithms or proposing a new algorithm tailored to the HCDG generation of efficient HDL code to exploit the underlying tool capabilities while circumventing their limitations.

Our high level pre-synthesis process is composed of several steps. Firstly, pre-scheduling transformations are performed on HCDG. One advantage of these transformations is that description becomes insensitive to syntactic variances. Next, scheduling under constraints is applied followed by post-scheduling transformations, which are necessary to incorporate in the HCDG information about resource sharing and speculative execution. The last step is a VHDL code generation at the behavioral level using extended lower level optimizations like: *don't care* values assignment and value range information incorporation.

The result of high level pre-synthesis process is a VHDL code written at behavioral level. One of the possible extensions is generating a code directly at RTL level due to an information provided by scheduler. We have investigated this possibility. In this context we have explored functional units assignment in order to assure the same activity level of all functional units in data path. This is necessary to avoid thermal shock in integrated circuits. We have developed a new methodology based on static measure of operators activities. This information is necessary to build a fitness function used by genetic algorithm, which find the best operators binding to functional units. We have also investigated register-sharing methodology that we have extended by using exclusiveness information.

2 Achieved Results

1. A quasi-automatic pre-prototype has been developed implementing high level pre-synthesis step.
2. A pre-prototype has been developed for finding the best possible operators binding to functional units. This should probably be demonstrated at the review.
3. A pre-prototype has been developed for efficiency register sharing.

3 High Level Pre-Synthesis

Here we give additional details on the first topic, see the references below for more information.

3.1 Motivation

Taking advantage of existing High-Level Synthesis (HLS) tools is a possibility that merits consideration. Existing HLS technology took many years to develop and has reached a stage where complex combinations of optimization trade-off's can be handled. At this stage, developing a whole new tool chain based on a different internal representation seems to be quite impractical. Nevertheless it is always possible to add some high level-optimizing step which permits to improve the performance of existing tools. In this context we propose high level pre-synthesis optimization using our HCDG internal representation. Pre-synthesis process is composed of following steps:

- pre-scheduling transformations,
- scheduling under constraints,
- post scheduling transformations,
- VHDL code generation using extended optimizations.

3.2 Pre-scheduling transformations

In this context the following transformations are applied:

- *lazy execution*: it consists in replacing the definition guard of the node by its utilization guard

- *node duplication*: for nodes which produce data consumed by other nodes under mutual exclusive conditions

- *elimination of false paths*: A false path has its guard equal to false; a path guard is defined as the conjunction of the guards associated to the HCDG edges of a path

3.3 Scheduling under constraints

Our HCDG-based scheduling approach exploits most of the existing scheduling optimization techniques, enjoying their combined benefits. Both speculative execution and conditional resource sharing are combined in a uniform and consistent framework. Even more, it does not suffer from the effects of syntactic variance at both the mutual exclusiveness detection and scheduling levels, as CDFG, CFG based approaches. Structural, behavioral and data-flow exclusiveness can be extensively detected using graph transformations and guard information. The hierarchical control representation permits to minimize the number of mutual exclusiveness tests and also develop probabilistic priority functions that account for the conditional behavior of the design.

3.4 Post scheduling transformations

After scheduling has been performed, information relating to conditional resource sharing and speculative execution can be modeled by HCDG transformations to produce an equivalent HCDG that incorporates this information and on which further scheduling iterations or allocation/binding can be performed. Modeling speculative execution corresponds in changing the control dependencies and the guards labeling of nodes that are speculatively executed. Conditional resource sharing is modeled by fusing the nodes that are shared into a single node. A new guard is computed and the initial input data dependencies are multiplexed to produce the input dependencies of the new node.

3.5 VHDL code generation using extended optimizations

During the generation of the HDL code there are some coding guidelines that can be implemented and enabled by appropriate HDL code generation options. Using them results in significant circuit area reduction. Two such possibilities have been identified:

- initial assignments of *don't care* values which is important for mainly two reasons. First, it avoids the inference of unnecessary latches, by the underlying synthesis tool, due to incompletely specified conditional constructs. Second, it enables the optimization of the circuit by permitting the synthesis tool to choose the best value, for the *don't care*, that will minimize the synthesized logic.

- use of value range information explicitly incorporated in the high-level specification in order to minimize bus widths and silicon area.

4 Functional units balancing in High Level Synthesis

4.1 Motivation

Relying on the HCDG, we combine a formal reasoning on events with a genetic algorithm to find an optimal assignment in terms of execution balance between functional units. Thus, we need to focus on the probabilistic analysis of some event combinations. This may lead to avoiding transient or permanent faults in a single chip, due to heat dissipation (arising when the circuit undergoes logic transition, mainly).

4.2 Design space exploration via an evolutionary method

Firstly we have defined a *harmony matrix* (here denoted by $\mathcal{M}_{\mathcal{A}}$) that explains in what manner during any iteration the functional units are used.

$$\mathcal{M}_{\mathcal{A}} = \begin{pmatrix} \mathcal{H}_0(A) & \mathcal{H}_1(A) & ... & \mathcal{H}_{CS}(A) \\ \mathcal{H}_0(B) & \mathcal{H}_1(B) & ... & \mathcal{H}_{CS}(B) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{H}_0(Z) & \mathcal{H}_1(Z) & ... & \mathcal{H}_{CS}(Z) \end{pmatrix}$$

Element $\mathcal{H}_i(A)$ of this matrix gives the probability of functional unit A to be activated exactly i times in any iteration. The matrix is calculated in static fashion using information about the probability of operators activity. A Matrix $\mathcal{M}_{\mathcal{A}}$ is necessary to define:

- $\mu(\alpha)$ value which gives total number of operations/iteration performed by functional unit α

$$\mu(\alpha) = \sum_{i=1}^{n_{CS}} i.\mathcal{H}_i(\alpha)$$

- $\mu(\mathcal{A})$ value that is a measure of load balance between the units in the design

$$\mu(\mathcal{A}) = \sum_{i=1}^{n_{UNIT}} \sum_{j=i+1}^{n_{UNIT}} [\mu(\alpha_i) - \mu(\alpha_j)]^2$$

- $F(\mathcal{A})$ *fitness* function used in the next time by genetic algorithm.

$$F(\mathcal{A}) = \frac{1}{1 + \mu(\mathcal{A})}$$

At the beginning our algorithm creates a population at random (the population is composed of the individuals where each one gives an information about of particular biding of operators to functional units). New populations are then created based on *crossover* and *mutation* laws until the *fitness* of the best individual remains stable.

References

[1] J.C.Le Lann, C. Wolinski, Load balancing and Functional Unit Assignment in High-Level Synthesis, SCI'99/ISAS'99 Florida July 31 - August 4, 1999 (compressed postscript)

[2] A. Kountouris, C. Wolinski, High Level Pre-Synthesis Optimization Steps using Hierarchical Conditional Dependency Graphs EUROMICRO'99, IEEE Computer Society Press, Milano, Itali, AUG 1999. (compressed postscript)

[3] A. Kountouris, C. Wolinski, Combining Speculative Execution and Conditional Resource Sharing toEfficiently Schedule Conditional Behaviors, Proceedings of ASP-DAC'99,IEEE Computer Society Press, Hong-Kong, Jan. 18-21, 1999. (compressed postscript 45K)

[4] A.A. Kountouris, PhD Thesis, Univ. Rennes I, 1998.