PARLAK: Parametrized Lock Cache Generator

Bilge E. S. Akgul and Vincent J. Mooney
School of Electrical and Computer Engineering, Georgia Institute of Technology, USA
{bilge, mooney}@ece.gatech.edu

Abstract

A system-on-chip lock cache (SoCLC) is an intellectual property (IP) core that provides effective lock synchronization in a heterogeneous multiprocessor shared-memory system-on-a-chip (SoC). We present PARLAK, a parametrized lock cache generator tool. PARLAK generates a synthesizable SoCLC architecture with a user specified number of lock variables and user specified number and type(s) of processor(s). PARLAK can generate a full range of customized SoCLCs, from a version for two processors with 32 lock variables occupying 1,790 gates of area to a version for 14 processors with 256 lock variables occupying 37,380 gates of area (in TSMC 0.25μ technology). PARLAK is an important contribution to IP-generator tools for both custom and reconfigurable SoC designs.

1. Introduction

The SoCLC has been shown to achieve speedups of 55% and 27% in realistic examples when compared to the traditional spin-lock mechanism at a very small ($<$ 13,000 gates) hardware cost [1], [2], [3]. However, it is also desired to be able to customize/configure and parameterize (according to the customer specifications) the SoCLC with the minimum engineering effort possible in an automated fashion. One approach to solve these demands can be referred to as an IP-generator tool. In this context, we present PARLAK, parametrized lock cache generator, that generates a custom SoCLC for an SoC including reconfigurable and/or custom logic and multiple heterogeneous processors such as in Figure 1.

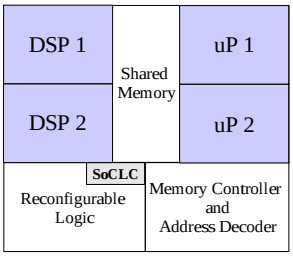


Figure 1: A typical target SoC architecture for which PARLAK can be used to reconfigure SoCLC for four processing elements.

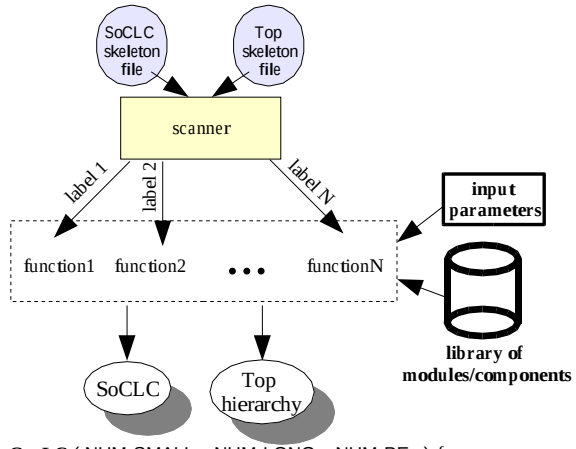
2. Lock Cache Generator

2.1. PARLAK

The output of PARLAK is the specified SoCLC architecture in Verilog. The modules that comprise the lock cache are parametrized. The number of small critical section (CS) locks, number of long CS locks, number of processing elements (PEs) and the type of the PEs are parameters that are used by PARLAK to build the lock cache. On the other hand, PARLAK uses a library of components of the base architectures – built previously – of the lock cache. Each module of the lock cache is customized using the parameters specified by the user and using the library components. The command executed by the user is as follows:

GenLC { PE_TYPE NUM_PEs }$^+$ { NUM_SMALL } { NUM_LONG }

where NUM_SMALL, NUM_LONG, PE_TYPE and NUM_PEs specify the number of small CS locks, number of long CS locks, type of processor and number of processors, respectively. The GenLC command needs to know the type(s) of processor(s) used so that the corresponding PE module and memory/interrupt controller module instantiations are performed in the top hierarchy. Also, note that {PE_TYPE NUM_PEs}$^+$ specifies that one or more occurences of "PE_TYPE NUM_PEs" may appear in the command line. Currently, GenLC is geared to generate code for simulation in Seamless CVE and only two PE_TYPEs are supported: MPC750 and MPC755.




```
GenLC ( NUM_SMALL,  NUM_LONG,  NUM_PEs ) {
  /* Begin scanning the SoCLC skeleton file */
  L = First_label_of_SoCLC_skeleton_file;
    WHILE (L) /*loop until labels are exhausted*/
    {     switch (L)  /*generate customized code for each label*/
                case(1): { function1();}
                case(2): { function2();}
                ...
                case(N): { functionN();}
          Insert_customized_code_into_SoCLC_output_file_for_L;
          L = Next_label_of_SoCLC_skeleton_file;
    }
}
```

Figure 2: PARLAK building blocks and pseudo algorithm of SoCLC code-generations.

PARLAK (Figure 2) handles the lock cache generation process through the following three building blocks (see Figure 2 for pseudo code). The first building block is the set of input parameters specified by the user that determine the SoCLC size and capacity (for how many processors the SoCLC will be generated and how many small and long CS locks the SoCLC will support). The second building block is a skeleton SoCLC Verilog file which includes the base signal, process and module descriptions that do not depend on any input parameters. Moreover, this skeleton file is labeled at those signal/module/process locations that strictly depend on the input parameters. Based on this skeleton file, the corresponding parametrized descriptions are generated and inserted into

an output file incrementally at each label. The third building block consists of seed PARLAK functions that generate the actual parameter-dependent signal/module/process descriptions. These functions interact with the library that includes the code sections to be enumerated/instantiated according to the input parameters. The library has been manually extracted from a complete, fully customized SoCLC Verilog file. The functions are executed at the corresponding labels as the skeleton input file is scanned. Finally, there is PARLAK executable code that manipulates these three building blocks. The PARLAK executable gathers the parameters obtained from the user, scans the skeleton input file for labels, calls the relevant functions at each label encountered in order to generate the customized Verilog code and integrates the generated code with the skeleton into an output file (the pseudo code in Figure 2 depicts the steps taken by the PARLAK executable). Execution is continued until all the labels in the skeleton input file are exhausted. The resulting output file represents the final, synthesizable, customized SoCLC architecture that the user is interested in. Note that a similar flow of operations is performed in the top hierarchy generation as well (Figure 2).

2.2. Synthesis Results

This section presents the synthesis results of the SoCLC. The Design Compiler from Synopsys with a 0.25μ technology TSMC standard cell library from LEDA has been used for the synthesis of the SoCLC. Figure 3 illustrates how the SoCLC scales as the number of locks is increased from 32 to 256.

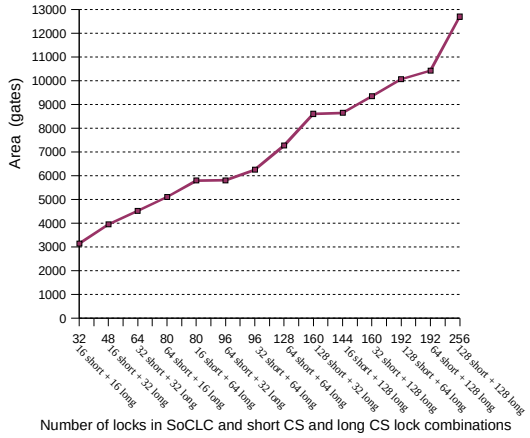


Figure 3: Synthesis results for increasing number of locks in the SoCLC. Number of PEs is equal to 4.

In case of four processors and 32 lock variables (16 short CS locks and 16 long CS locks), the SoCLC occupies an area of 3,146 logic gates. However, in the case of four processors and 256 lock variables (128 short CS locks and 128 long CS locks), the SoCLC occupies 12,699 logic gates. Here, the gate unit represents the area of a 2-input standard NAND gate. Figure 4, on the other hand, illustrates how SoCLC scales as the number of processors is increased for different combinations of number of lock variables from 32 locks to 256 locks. The number of small CS locks and long CS locks in each combination of Figure 4 are equal. While Figure 4 shows the total area of the SoCLC,

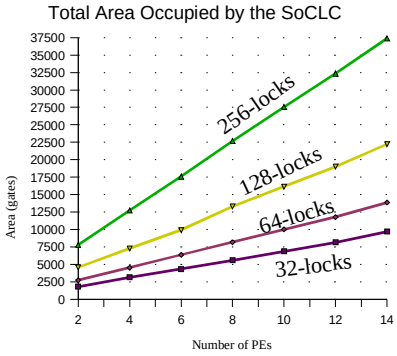


Figure 4: Synthesis results of the total area of the SoCLC for increasing number of PEs for number of locks = 32, 64, 128 and 256.

Figure 5(a) and Figure 5(b) illustrate the memory-only logic area and non-memory logic area, respectively (in short, adding Figure 5(a) and Figure 5(b) together results in Figure 4). As seen from the figures, the area increases linearly as the number of processors in the SoC and the number of lock variables residing in the lock cache are increased.

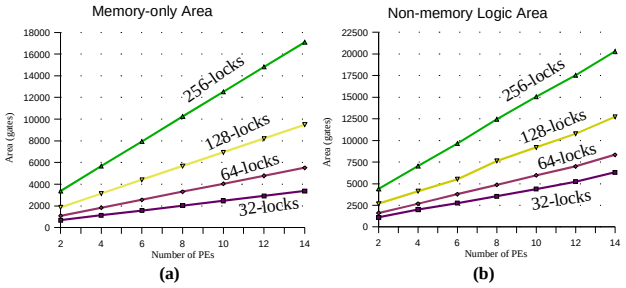


Figure 5: (a) Memory-only area of the SoCLC. (b) Non-memory area of the SoCLC.

3. Conclusion

In conclusion, SoCLC is a high performance, intelligent hardware solution, and, using the PARLAK tool, scalable, easily applicable, customizable versions of the SoCLC can be generated for a heterogeneous multiprocessor SoC design.

4. Acknowledgements

This research is funded by the State of Georgia under the Yamacraw initiative (www.yamacraw.org) and by NSF under INT-9973120, CCR-9984808 and CCR-0082164. We also acknowledge software donations from Mentor Graphics and Synopsys as well as hardware donations from Sun and Intel.

References

[1] B. E. Saglam (Akgul) and V. J. Mooney, "System-on-a-chip processor synchronization support in hardware," *Design Automation and Test in Europe (DATE'01)*, pp. 633–639, March 2001.

[2] B. E. S. Akgul, J. Lee, and V. J. Mooney, "A system-on-a-chip lock cache with task preemption support," *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'01)*, pp. 149–157, November 2001.

[3] B. E. S. Akgul and V. J. Mooney, "The system-on-a-chip lock cache," *International Journal of Design Automation for Embedded Systems*, vol. 7, no. 1-2, pp. 139–174, September 2002.