# List Based Scheduling in High Level Synthesis

A study on List Based Scheduling algorithm for application in high level synthesis

Emirkan Sali
*Electronic Engineering Student*
*University of Applied Sciences Hamm-Lippstadt*
Lippstadt, Germany
emirkan.sali@stud.hshl.de

***Abstract*—In Hardware/Software Codesign, developers and engineers desire to design systems more efficiently when needed. Utilizing High Level Synthesis is a way for Designers of a system to create Hardware description language for their system in a fast and less expensive way using high level coding. This tool converts the operations of a high level function automatically into a Control data flow graph and Schedules the operations according to the set constraints, as well as allocates needed elements and memory and finally connects every element together to create accurate Hardware description language. List based Scheduling in High-Level Synthesis is a Heuristic algorithm to properly schedule operations in a Control Data flow Graph with respect to Design and resource constraints.**

***Index Terms*—High-Level, Synthesis, Codesign, Scheduling, Heuristic, List-Based, Design, Constraints**

## I. Introduction

In today's development cycles of Systems for any area of application there is never a system that works out perfectly after the software and hardware design is completed and the system is up for testing. Often times, engineers and designers find some mistakes in the hardware or software later or some problems become only clear when seeing test results of systems in later stages of development. Often times, this means that the system has to be redesigned again, which can be tedious and time consuming. Usually when designing systems, the hardware components needed for the system are specified first and then building on that, the software is being developed to meet the systems functional and non-functional requirements [1]. A typical design flow for a traditional system is shown in figure 1 taken from [1]

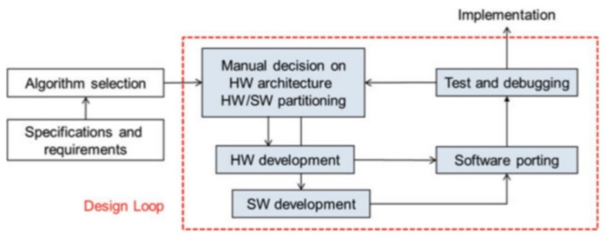


Fig. 1. Visualization of a typical design flow of an electronic system taken from [1]

In this type of Design flow, designing hardware and software components is done separately, while the software design needs the basis of the hardware to start development [1]. To avoid this time consuming process, Hardware/Software Codesign (HW/SW Codesign) methods are introduced, where both hardware and software are developed at the same time using algorithms and synthesis tools [1]. Especially High-level Synthesis (HLS) is a crucial part of HW/SW Codesign, because it processes inputs from a High level language like C or C++ into a low level Hardware description language (HDL) [2]. The behavior of the system is described using the high level language and an automated tool converts it into a register transfer level (RTL) description [2]. A visualization of the HLS process can be seen in figure 2 from [2].

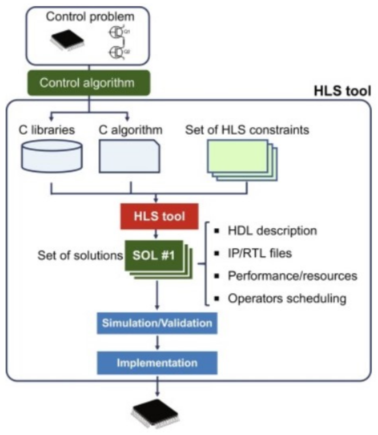


Fig. 2. Visualization of the design flow of an electronic system using High-level Synthesis taken from [2]

This paper addresses the flow of HLS with some examples and sections about each of its steps. The main focus is the scheduling part of HLS with further focus on the list-based scheduling algorithm and the comparison to other scheduling algorithms in its domain.

## II. High Level Synthesis

The general procedure of using HLS can be described by 4 sub-tasks [3]:

1) Behavior description of the system using a hardware description language like VHDL and Representing the description in a graph-based way called control data flow

graph (CDFG) which can be a control flow graph (CFG) and data flow graph (DFG) [4] [3]

2) Scheduling, in which every operation in CDFG receives a control step (c-step), i.e a clock cycle signalising its execution [3]
3) allocating the desired number of components, connections etc. for the scheduled system while considering restraints. [3]
4) Finalizing task ”binding”, in which operations are assigned to functional units, values to all storage elements and production of an data path by interconnecting all components with each other [3]

## A. *Elaboration*

A behaviour description could be any function that takes variables as inputs and returns an output based on computations done in that function [4]. The computations in these functions are being visualized in a CDFG like in the following Example:

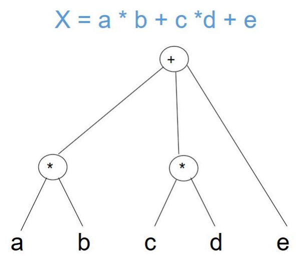


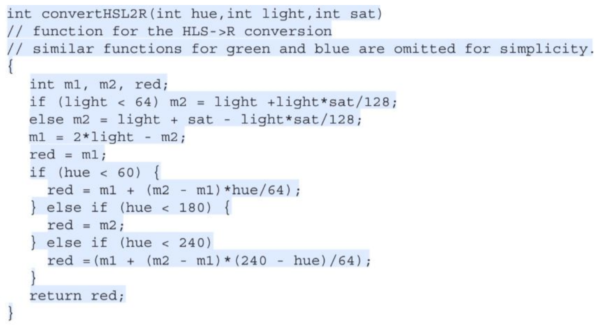
Fig. 3. A simple computation that could be in a function written in high level code visualized in a DFG

In figure 3 for example, we have a simple computation, where variables $a, b, c, d$ and $e$ are added together or multiplied. Through the process of HLS, this computation is visualized in the Data flow graph in figure 3. HLS needs this DFG for scheduling the operations later on.

A simplified example for a behavioural Description could be an function to convert the hue-saturation-light format for color determination into the more common red-green-blue format shown in the following C-code example from [4] in figure 4 :

This code alone is not enough for HLS, because it does not include any design constraints [4]. This can be specified when the designer knows for example, that this function computes the needed pixel color of a screen and a new pixel needs to be produced every 10 nanoseconds with a clock cycle of 5 nanoseconds [4]. that would mean that the system has a required throughput rate of 100 MHz and the latency can not be higher than two cycles per pixel computation [4]. Constraints like these can be added in two different ways:

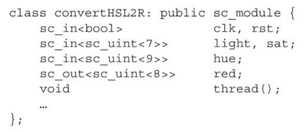
1) Latency annotations specifically made understandable for the HLS tool [4]
2) Adding timing statements in the code like $wait()$ for SystemC [4]



```
int convertHSL2R(int hue,int light,int sat)
// function for the HLS->R conversion
// similar functions for green and blue are omitted for simplicity.
{
   int m1, m2, red;
   if (light < 64) m2 = light +light*sat/128;
   else m2 = light + sat - light*sat/128;
   m1 = 2*light - m2;
   red = m1;
   if (hue < 60) {
     red = m1 + (m2 - m1)*hue/64);
   } else if (hue < 180) {
     red = m2;
   } else if (hue < 240)
     red =(m1 + (m2 - m1)*(240 - hue)/64);
   }
   return red;
}
```

Fig. 4. An example C-Code from [4] to convert the hue-saturation-light format into the red-green-blue format

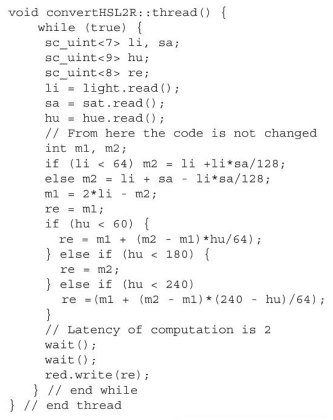
These Input specifications are coded in SystemC by creating a class first to specify the clock and the other needed variables shown in figure 5 from [4]:



```
class convertHSL2R: public sc_module {
    sc_in<bool>             clk, rst;
    sc_in<sc_uint<7>>       light, sat;
    sc_in<sc_uint<9>>       hue;
    sc_out<sc_uint<8>>      red;
    void                    thread();
    …
};
```

Fig. 5. Class input specification from [4] for the HSL tool for the function in figure 4

The function from figure 4 is rewritten to include the input specifications and design constraints in figure 6 from [4]



```
void convertHSL2R::thread() {
    while (true) {
     sc_uint<7> li, sa;
     sc_uint<9> hu;
     sc_uint<8> re;
     li = light.read();
     sa = sat.read();
     hu = hue.read();
     // From here the code is not changed
     int m1, m2;
     if (li < 64) m2 = li +li*sa/128;
     else m2 = li + sa - li*sa/128;
     m1 = 2*li - m2;
     re = m1;
     if (hu < 60) {
       re = m1 + (m2 - m1)*hu/64);
     } else if (hu < 180) {
       re = m2;
     } else if (hu < 240)
       re =(m1 + (m2 - m1)*(240 - hu)/64);
     }
     // Latency of computation is 2
     wait();
     wait();
     red.write(re);
   } // end while
} // end thread
```

Fig. 6. Function of figure 4 rewritten in a way that includes design constraints and design specifications

This entire process is part of the Elaboration step of HLS in which the High level code is then processed into a CDFG like in figure 3 [4].

## B. *Optimization and Micro-architecture transformations*

Optimization of the produced CDFG is necessary, because in-optimized CFG’s or DFG’s can lead to extra hardware being utilized in the output generation [4]. Extra hardware always means a higher cost and is therefore a problem to avoid to be able to compete in the hardware market [4].

Methods for optimization are common compiler optimizations like dead code elimination or common sub-expression extraction and more, but also methods specific for hardware like bit-trimming [4].

Architectural choices for the outputted HDL also need to be made by designers of the system [4]. Generally they fall into two types of categories [4]:

1) Restructuring control or design hierarchy, where choices for handling loops and functions are made [4]
2) Specifying memory architecture, where decisions about how to store the data in the computations are made [4]

### C. Scheduling

Scheduling in HLS is done with the sequencing graphs of the previous steps and its given constraints [5]. It precisely determines the start of each task in the sequencing graph and is therefore impacting performance of the HLS [5]. Scheduling is taking into account all ressources, constraints and operations and assigning the resources to the needed computations [4].It is also making clear how these computational pieces are associated with all the states on the thread [4]. Here, a chosen scheduling algorithm assigns control steps to operations based on design constraints. [1] They can be divided into two types of algorithms:

1) Exact algorithms
2) Heuristic algorithms

Because of the high execution time problem of Exact algorithms, even though they provide optimal schedules, many algorithm based on heuristics have been developed. In these heuristic algorithms the optimal control steps for operations are selected on a local basis without considering past decisions or any predictions towards future selections [1]. This may be producing not an optimal solution when looked at globally, but the produced output is given in a short time period while being sufficiently near to the optimal solution [1]. Some examples of heuristic algorithms are:

- As Soon As Possible (ASAP) algorithm [1]
- As Late As Possible (ALAP) algorithm [1]
- Force-Directed Scheduling (FDS) algorithm [1]
- List Scheduling algorithm [1]

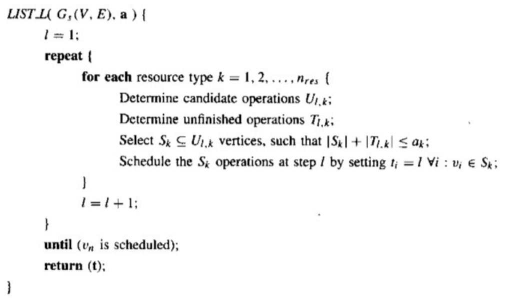
In an scheduling algorithm, to describe the behaviour of the algorithm and the scheduled sequencing graph, the following notations are introduced according to [5]:

- Vertex set $V$ that describes all inputs, signals in a sequencing graph
- Vertices in set $V = [v_i; i = 0, 1, ....n]$
- set of operations and edge set $E = [(v_i; v_j); i, j = 0, 1, ....n]$
- source vertex $v_0$ and the sink $v_n$, with both being no operation vertices
- Operation delays $D = [d_i; i = 0, 1, ....n]$
- Start time for operations $T = [t_i; i = 0, 1, ....n]$
- latency of the system being $\gamma = t_n - t_0$
- the number of resources available for operations $a_i = [1, 2, ....n]$ with $i$ being the type of ressource

## III. List-Based Scheduling

List-based scheduling is a heuristic type of scheduling algorithm for HLS that solves ressource constraint problems and is ultimately "an extension of Hu's algorithm to handling multiple operation types and multiple-cycle execution delays" [3] [5].

A ressource constraint problem specifies that the algorithm needs to find the fastest schedule that satisfies given constraints with a maximum number of ressources that cannot be exceeded in the process [6]. Ressources in this case are the function units required for the system [6]. The Algorithm can be denoted in code form from [5] in figure 7:



```
LIST_L( $G_s(V, E)$, a ) {
    $l = 1$;
    repeat {
        for each resource type $k = 1, 2, \dots, n_{res}$ {
            Determine candidate operations $U_{l,k}$;
            Determine unfinished operations $T_{l,k}$;
            Select $S_k \subseteq U_{l,k}$ vertices, such that $|S_k| + |T_{l,k}| \leq a_k$;
            Schedule the $S_k$ operations at step $l$ by setting $t_i = l \ \forall i : v_i \in S_k$;
        }
        $l = l + 1$;
    }
    until ($v_n$ is scheduled);
    return (t);
}
```

Fig. 7. Code for List Scheduling from [5]

The algorithms that are list-based work according to the selection steps, where a priority list of all operations is used to assign the right control step to each operation [5]. Commonly, vertices of a priority list are labeled with a weight that represents their longest path to the sink and ranked in a decreasing order [5]. As a consequence, the most urgent operations are scheduled to be the first ones to be executed [5]. This list is modified by giving each operation a deadline based on timing constraints [5].

In a self-made example inspired by the example in [5] we consider the control data flow graph in figure 8 to be scheduled with list scheduling.

As resource constraints we have $a_1$ = 2 multipliers and $a_2$ = 1 arithmetic logic unit (ALU) with delays being $d_1$ = 2 for the multipliers and $d_2$ = 1 for the ALU. Given these constraints, one can make a List schedule following the priority based on weighting principle [5]. For the given example a simple list schedule is visualized in figure 10:

In this list schedule, we can see that the 2 available multipliers assign the vertices $v_6$ and $v_8$ to be scheduled in the first time slot. Those two vertices are part of the longest path to the sink and are therefore, based on the weighting principle, the most urgent multiply operations in the given example. The ALU is scheduled to execute the operation in $v_3$, because there is no other ALU operation available at that point in time.

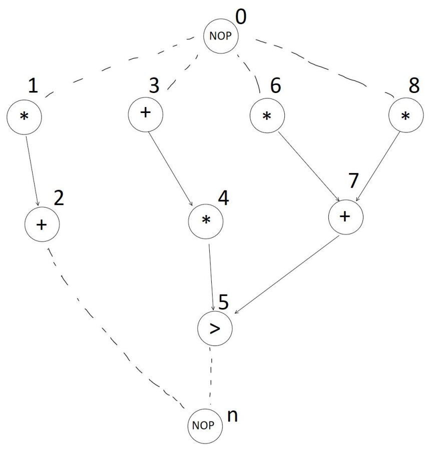
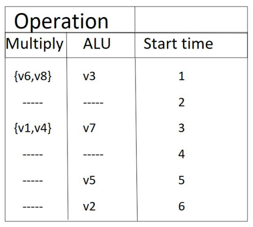


Fig. 8. An self-made example CDFG from [5]



| Operation | | |
|---|---|---|
| Multiply | ALU | Start time |
| {v6,v8} | v3 | 1 |
| ----- | ----- | 2 |
| {v1,v4} | v7 | 3 |
| ----- | ----- | 4 |
| ----- | v5 | 5 |
| ----- | v2 | 6 |

Fig. 9. list schedule of the CDFG in figure 8

After $v_6$ and $v_8$ are finished, the remaining 2 multiply vertices $v_1$ and $v_4$ are assigned to the available multipliers at time = 3. Beyond that, the remaining ALU operations are completed and the list schedule is generated.

Applying the schedule to the graph in figure 8, we obtain the scheduled graph in figure 10

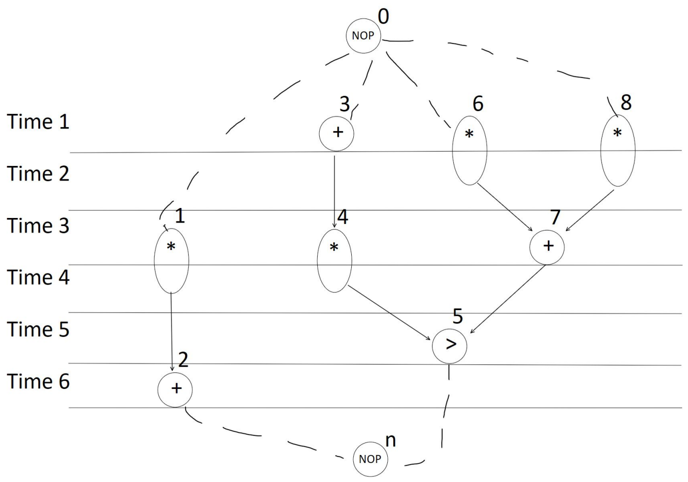


Fig. 10. graph from figure 8 scheduled with list based scheduling

### A. Hu's Algorithm

For a quick comparison, Hu's algorithm is suited. Hu's algorithm attempts to solve the ressource-constrained minimum latency problem. It is the basis of the List-based scheduling approach, but has a different, more greedy scheduling strategy. For each control step, it schedules as many operations as possible for the operations, whose predecessors have been scheduled already. Specifically, vertices with the largest labels are the ones picked first by the algorithm to schedule [5].

### B. ASAP and ALAP algorithm

Another notable set of algorithms are the As Soon As Possible (ASAP) algorithm and As Late As Possible Algorithm(ALAP).

ASAP is an unconstrained algorithm that simply schedules in the following way: "The start time for each operation is the least one allowed by the dependencies" [5].

ALAP is a lateny constrained algorithm and it seeks the longest path bettween one operation and the sink [5].

## IV. Conclusion

High level synthesis is a powerful tool for engineers and designers in order to translate the behaviour they want a system to have from high level code into a HDL or to assist engineers in creating a changed system after developing a failed prototype without high cost or time.

List based scheduling is a simple but efficient algorithm in HLS scheduling, that schedules any sequence graph with respect to design constraints. It may not find the absolute optimum solution for a schedule unlike other algorithms, but it offers a less time consuming and cost effective scheduling process instead while still creating a sufficient schedule for most cases.

## References

[1] D. Mueller-Gritschneder and A. Gerstlauer, *Handbook of Hardware/Software Codesign*. Springer, 2017, relevancy for paper: Overall relevancy for the topic of HW/SW Codesign.

[2] Óscar Lucía, E. Monmasson, D. Navarro, L. A. Barragán, I. Urriza, and J. I. Artigas, "Chapter 29 - modern control architectures and implementation," in *Control of Power Electronic Converters and Systems*, F. Blaabjerg, Ed. Academic Press, 2018, pp. 477–502, relevancy for paper: Excerpt from a book about High level synthesis in general. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128161364000300

[3] A. Sllame and V. Drabek, "An efficient list-based scheduling algorithm for high-level synthesis," in *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, Sep. 2002, pp. 316–323, relevancy for paper: Description of HLS and a good portion about list based scheduling.

[4] L. Scheffer, L. Lavagno, and G. Martin, *EDA for IC System Design, Verification, and Testing (Electronic Design Automation for Integrated Circuits Handbook)*. USA: CRC Press, Inc., 2006.

[5] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. McGraw-Hill Higher Education, 1994, relevancy for paper: Examples and explanations of List-based scheduling and HLS in general.

[6] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464–475, 1991.

*Statement of authorship*

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

Lippstadt, June 5, 2023 [illegible] . . . . . . . . . . . . .