György Pilászy

New algorithms for extending the high level synthesis methodology to designing and optimizing distributed systems

PhD thesis summary

Supervisor: Dr. Péter Arató
Advisor: Dr. Géza Móczár

The digital version is available in the following URL:

http://directory.iit.bme.hu/~gpilaszy/phd/

Budapest, 2013
Introduction

Due to the permanent technological development, a single chip nowadays may consist of even more hundreds of millions transistors. Therefore, very high-performance VLSI components (FPGA-s, processors, etc.) became available on extraordinary low price. Among such components there are also the microcontrollers and the diverse signal processing units with memory and suitable input/output facilities. These complex signal processing units are able to solve increasingly complex tasks independently as components of a distributed system.

For designing and optimizing distributed systems containing the above complex components, the existing more or less systematic system level synthesis (HLS) methods, like the high level synthesis algorithms are not applicable without changes and extensions. These methods generally starts from a dataflow graph obtained by decomposing (partitioning) the task specification described by a high level programming language (e.g. C) [7, 30]. The existing design tools attempt to optimize directly from the dataflow graph supposing relatively simple execution unit as components. These existing tools are not able to handle complex processing units specified by the design procedure.

The dissertation aims to extend the methodology of the high level synthesis for making it applicable in the systematic design and optimization of distributed (multiprocessing) systems.
Specifically:

- To elaborate a method for handling the transfer times of the communication buses between the complex processing units.
- To examine the effect of incrementing the latency in pipeline systems.
- To elaborate a method for the impact assessment, whether a calculated latency increment could decrease the cost at preserving the throughput (restart time).
- To develop an algorithm for fulfilling predefined separation and fusion conditions in the synthesis procedure.

![Diagram of the design procedure of distributed systems](image)

**Figure 1. Extension of the HLS methodology to systematic design of distributed systems**

The design procedure of distributed systems—in the above sense— is summarized in Figure 1. The extensions and the new algorithms are illustrated in thick frames.
The results of the dissertation

1. Communication time estimation

The most existing HLS tools [3, 25] do not assume any time demand for the communication between the operations in the initial dataflow graph or between the complex processing units specified by the decomposition and allocation steps. In [6, 31, 32] there are arguments for the importance of handling the communication properly, but the communication is characterized only by the number of bits to be transferred and by some basic properties without calculating the time demand [6]. Such an approach is acceptable inside an FPGA, but between the separate complex processing units (e.g. microcontrollers, microprocessors with integrated communication channels), the proper time demand estimation for the communication cannot be avoided just at the beginning of the design procedure. [5]

The methodology presented in this dissertation handles the communication as an additional elementary operation. In this case, the communication time demand is represented by the execution time of this additional operation. The exact communication time is not always known in advance (e.g. because of data-dependent bit insertion), therefore a realistic estimation is crucial. Underestimation yields function especially in pipeline mode. Overestimation slows down the resulting system unnecessarily. A proper upper limit in the estimation may help in avoiding such problems.

By analysing the bus systems as examples in the dissertation, the following main properties can be observed:

- The smallest data unit to be transferred is one byte in most interface systems.
- The data bytes are often provided with additional bits (e.g.: start, stop, ack).
- To the communication belong both an additional bitfield of fix length and an additional bitfield of variable length in each frame (package).
- The communication time-demand can properly be estimated as the integer multiple of the bit transfer time.

Figure 2 shows the flow chart of the time estimation algorithm.
The parameters for the four bus systems analysed in the dissertation for some further bus systems (not detailed here) are summarized in Table 1. The descriptions of the latter communication interfaces can be found in [S4, 27, 28, 29, 30].

Based on the algorithm, Fig. 3 shows the communication times of the bus systems analysed in details by assuming a data transfer speed 1Mbit/s and a data transfer 1-32 bytes. It can be observed for the CAN bus that the limited 8 byte frame increases the communication overhead, since further frames are required for transferring longer data.
<table>
<thead>
<tr>
<th>Interface</th>
<th>T_{bit}</th>
<th>C</th>
<th>B</th>
<th>K</th>
<th>M</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI</td>
<td>50ns min.</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I²C, (100kHz-1MHz) 7bits address</td>
<td>1-10μs</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I²C, (100kHz-1MHz) 10 bits address</td>
<td>1-10μs</td>
<td>0</td>
<td>1</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I²C, High speed (3.4MHz), 7 bits address</td>
<td>295ns</td>
<td>25</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I²C, Ultra high speed (5MHz),7 bits address</td>
<td>200ns</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UART (1 stop, 0 parity)</td>
<td>100ns min.</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UART (1 stop, 1 parity)</td>
<td>100ns min.</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CAN 2.0A</td>
<td>1μs min.</td>
<td>0</td>
<td>2</td>
<td>56</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>CAN 2.0B</td>
<td>1μs min.</td>
<td>0</td>
<td>2</td>
<td>81</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>BME-PS CAN plus</td>
<td>1-10μs</td>
<td>0</td>
<td>2</td>
<td>113</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Ethernet 802.3. (10Mbps)</td>
<td>100ns</td>
<td>0</td>
<td>0</td>
<td>26</td>
<td>1500</td>
<td>46</td>
</tr>
<tr>
<td>USB low-speed (1.5Mbps), control</td>
<td>666ns</td>
<td>0</td>
<td>1</td>
<td>504</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>USB low-speed (1.5Mbps), interrupt</td>
<td>666ns</td>
<td>0</td>
<td>1</td>
<td>152</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>USB full-speed (12Mbps), control</td>
<td>83.3ns</td>
<td>0</td>
<td>1</td>
<td>360</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>USB full-speed (12Mbps), Interrupt</td>
<td>83.3ns</td>
<td>0</td>
<td>1</td>
<td>104</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>USB full-speed (12Mbps), Isochronous</td>
<td>83.3ns</td>
<td>0</td>
<td>1</td>
<td>72</td>
<td>1023</td>
<td>0</td>
</tr>
<tr>
<td>USB full-speed (12Mbps), Bulk</td>
<td>83.3ns</td>
<td>0</td>
<td>1</td>
<td>104</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>USB high-speed (480Mbps), control</td>
<td>2.083ns</td>
<td>0</td>
<td>1</td>
<td>1384</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>USB high-speed (480Mbps), Interrupt</td>
<td>2.083ns</td>
<td>0</td>
<td>1</td>
<td>124</td>
<td>1024</td>
<td>0</td>
</tr>
<tr>
<td>USB high-speed (480Mbps), Isochronous</td>
<td>2.083ns</td>
<td>0</td>
<td>1</td>
<td>112</td>
<td>1024</td>
<td>0</td>
</tr>
<tr>
<td>USB high-speed (480Mbps), Bulk</td>
<td>2.083ns</td>
<td>0</td>
<td>1</td>
<td>440</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>SATA (1.5GB/s), parancskeret</td>
<td>666.6ps</td>
<td>0</td>
<td>2</td>
<td>96</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Hyper Transport (2bit,200MHz)</td>
<td>2.5ns</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>Hyper Transport (32bit,200MHz)</td>
<td>156.25ps</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>PCI-Express x1 (2.5Gb/s)</td>
<td>400ps</td>
<td>0</td>
<td>2</td>
<td>144</td>
<td>4096</td>
<td>0</td>
</tr>
<tr>
<td>PCI-Express x2 (2.5Gb/s)</td>
<td>200ps</td>
<td>0</td>
<td>2</td>
<td>288</td>
<td>8192</td>
<td>0</td>
</tr>
<tr>
<td>PCI-Express x4 (2.5Gb/s)</td>
<td>100ps</td>
<td>0</td>
<td>2</td>
<td>1152</td>
<td>16384</td>
<td>0</td>
</tr>
<tr>
<td>PCI-Express x8 (2.5Gb/s)</td>
<td>50ps</td>
<td>0</td>
<td>2</td>
<td>2304</td>
<td>32768</td>
<td>0</td>
</tr>
<tr>
<td>PCI-Express x16 (2.5Gb/s)</td>
<td>25ps</td>
<td>0</td>
<td>2</td>
<td>4608</td>
<td>65536</td>
<td>0</td>
</tr>
<tr>
<td>PCI-Express x32 (2.5Gb/s)</td>
<td>12.5ps</td>
<td>0</td>
<td>2</td>
<td>9216</td>
<td>131072</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1 Parameters for estimation algorithm

![Figure 3 Communication time for 1-32 bytes with 1us T_{bit}](image-url)
Applying the results in extending the HLS algorithms

The HLS tools usually represent the task to be solved by an initial dataflow graph (EOG) called often elementary operation graph (EOG). In Figure 4 a simple example is shown for inserting the additional communication operation \( e_3 \) between the elementary operations \( e_1 \) and \( e_2 \). For illustrating, the EOG of HLS tool PIPE is applied.

![Diagram of EOG with communication](image)

**Figure 4 Representation of communication in EOG**

The execution (latency) time of operation \( e_1 \) is \( t_k \). Since the communication time \( T_k \) depends on the bit transfer speed, therefore it has to be transformed into the timing system of the HLS tool. In tool PIPE, the basic time unit is the clock period \( T \), thus:

\[
t_k = \frac{T_i}{T}
\]

According to one of the solutions proposed in the dissertation, the communication operation should be inserted into each data connection of the initial EOG at the beginning of the design. After the decomposition, when the segments become known, \( t_k = 0 \) values have to be set inside the segments. This situation is illustrated in Figure 5.

![Diagram of interpreting communication operations](image)

**Figure 5 Illustration of interpreting the communication operations inside the segments and between the segments**
1. Thesis

- The concept of elementary operations can be extended to operations with arbitrary complexity by specifying them at the beginning of the design procedure or by producing them during the decomposition or allocation. Between such complex components (microprocessors, microprocessors, signal processing units, etc.), the communication time has to be taken into consideration.

- I elaborated a method for handling the communication as additional operations between the complex processing units. The execution time of such additional operation is the calculable or estimable communication time.

- I developed an algorithm for calculating and estimating the communication time. The algorithm is illustrated for some often used bus systems.

\[ \text{Estimated communication time: } T_k \]

- I have shown on an example that the traditional HLS method may result in structure which could not be implemented by applying the usual embedded communication channels.

- I have shown that the extended HLS (PIPE) method yields an implementable solution by applying the algorithm I have elaborated.

- The elaboration of the method and the algorithm are my own results

- The publications containing the results of this thesis: [S2], [S6], [S8] and [S9].
2. The effect of increasing the latency time in high level synthesis

In pipeline systems the HLS tools attempts to optimize the restart (initialisation) time (R) and the latency (L) is an output parameter obtained at the end of the calculations [3]. It is not examined, whether the intentional increasing of the latency how could influence the complexity and the cost in pipeline systems. This is the aim of the following examination by applying the HLS tool PIPE, but the conclusions remain valid for other tools as well. The cost reduction can be expected if originally overlapped operations become not overlapping due to longer latency by proper scheduling. The operations not overlapping any more may be implemented in the same execution unit by the allocation.

**Necessary (but not satisfactory) rules for cost reduction:**

1. Obviously, only such execution units are worthy to be examined which occur in more copies in the allocated EOG.
2. It is beneficial to specify the trivially lowest cost. Having achieved this cost value, further cost reduction cannot be expected. For example: if an EOG consists of four different operations which cannot be implemented by common execution unit, then at least four execution units are required for the solution.
3. The minimal required copies (number of pieces) of the different execution units can be calculated for a given restart time. Based on this result, the theoretically achievable minimal cost can be calculated. This cost value should be considered as a stop condition for examining the effect of increasing the latency.
4. In the case of identical operations started at the same time, the latency should be incremented at least by the execution time of the operation, because overlapping operations cannot be implemented by common executing unit.
5. Overlapping operations can be scheduled after each other by a latency incrementation at least the overlapping value, if there is no other exclusion condition.
6. For using the complex execution unit $M_i$ k-times during a restarting period, the sum of the execution times ($t_k$) of the k pieces of elementary operations implemented by $M_i$ must not be greater than the restart time: $\sum_k t_k \leq R$

If the execution times of the elementary operations allocated in $M_i$ are identical, then: $k \cdot t_k \leq R$

In the dissertation, more example is shown for the impact assessment of increasing the latency time. Figure 6 shows for instance the EOG of task F2.
Figure 6 The F2 EOG

The properties of the three execution units assumed in the EOG are summarized in Table 2.

<table>
<thead>
<tr>
<th>execution unit</th>
<th>Elementary operations</th>
<th>$t_i$</th>
<th>pieces in EOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$e_1...e_{12}$</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$e_{13}...e_{24}$</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$e_{25}...e_{36}$</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 2 The properties of the execution units

The comparison of the various solutions has been made according to a very simple cost calculation that is the product of the number of the applied execution units and their execution times Fig. 7 and Fig. 8. The lowermost curves correspond to the theoretical cost minimum at a given restart time. It can be observed that this cost is approached closer at longer latency values.
Figure 7 Cost functions of F2 in case of three different execution units

Figure 8 Cost functions of F2 in case of complex execution units
The algorithm proposed for the impact assessment of the latency incrementation

The parameters and notations for the algorithm are defined in the dissertation, only the resulted procedure are presented here.

1. Run PIPE on the initial EOG for the given R. Based on the result, $L_{\text{min}}$ is obtained.
2. Determine the minimal number of pieces ($N_{\text{MIN}}$) of each execution unit $P_i$;

$$N_{\text{MIN}} = \sum_k \left\lfloor \frac{n_k \cdot t_k}{R} \right\rfloor; \text{k refers for the operations which are executable by } P_i.$$

3. Calculate $C_{\text{MIN}}$.
4. Generate partition $\Pi_M$ based on set M. Select the blocks ($m_i$) of the partition for which $N_i > N_{\text{MIN}}$ and $T_i \leq R/2$ stand.
5. Determine a latency increment value $\Delta$ as the execution time $T_i$ belonging to $m_i$ having the greatest $N_i$ (In case of more greatest $N_i$-s, then select from them according to the smallest $T_i$).
6. Increment the latency by the selected $\Delta$.
7. Rerun PIPE
8. Calculate the cost $C$. Check the stop conditions, if not fulfilled, repeat from step 4.
9. Stop if either from the following conditions are fulfilled:
   - if the expected cost ($C_d$) given in advance achieved or
   - if the maximal latency ($L_{\text{MAX}}$) given in advance is achieved or
   - if the minimal cost value ($C_{\text{MIN}}$) is achieved.

The simplified flow chart of the algorithm is shown in Figure 9.

![Flow Chart](image)

**Figure 9** The flow chart of the algorithm for calculating the latency incrementation

The aim of the above algorithm is to avoid unnecessary reruns of PIPE. The rerun is started only if the latency increment is expected to cause cost reduction. Obviously, it does not make sense to increase the latency extremly, because the system would slow down without reason.
2. Thesis

In pipeline systems, the HLS tools attempts to optimize the restart time (R) and the latency (L) is an output parameter obtained at the end of the calculations. It is not analysed, whether the intentional increasing of the latency how could influence the complexity and cost in pipeline systems.

- **In the case of pipeline systems, a longer latency may cause cost reduction without slowing down the system, because the pipeline throughput are not affected significantly.**

- **I have elaborated a method and an algorithm for the impact assessment of incrementing the latency in the design procedure of distributed pipeline systems.**
  
  - I have evaluated the results by simulation.
  
  - I have developed a method for calculating beneficial increment values in increasing the latency.
  
  - The method allows the jointly modification of the restart time and the latency in order to approach the optimal cost.
  
  - Based on my proposal, the HLS tool PIPE has been modified in order become able to receive the latency as an input parameter.
  
  - The method and the algorithm summarized in this thesis are my own results.
  
  - The publication containing the results of this thesis: [S3].
3. New allocation method for predefined separations and fusions

In distributed systems (e.g. in the automatization of industrial systems), the improper task partitioning may cause difficulties. Although the whole system perform its specification, if each component (segment) functioning properly, but a malfunction of a component may prohibit the proper operation of such other components, which are affected only by the improper task partitioning and not by functional reason. Thus, the troubleshooting may become extremely difficult and time-consuming. Likewise, the improper partitioning may implement closely related task functions into different components (e.g. placed far from each other). Such a distribution of a function makes difficult the functional testing of the system, because all affected components should operate properly at the same time.

To avoid the above difficulties, predefined separations and fusions have to be ensured by the decompostion and allocation. The existing HLS tools are not prepared for fulfilling such boundary conditions.

The above problems are comparable with some boundary conditions in hardware/software codesign [22, 23, 24]. Namely, the task distribution must predefintely implement some functions into hardware components (e.g. vents, relays, sensors), and some others into software (e.g. programs for PLC or microcontrollers). There are also such task functions, which can be implemented either hardware or software. Besides, prescriptions in standards [11] (e.g. safety-critical rules) may also to be considered as predefinit boundary conditions.

Figure 10 illustrates the main steps of HLS [3, 24] affected by the separation and fusion conditions.

![Figure 10 The HLS steps affected by fusion and separation conditions](image-url)
Further on, the modification algorithm is detailed affecting the allocation step. It can be proven that the pairwise fusion condition of the elementary operations is a compatibility relation [12, 13]. Based on the maximal compatibility classes, a proper disjoint closed cover should be generated by the allocation.

**The proposed new algorithm for the allocation**

The main steps of the algorithm detailed in the dissertation are as follows:

1. Based on the predefined boundary conditions, generate the compatibility half matrix (CHM) containing the prescribed fusions, separations and conditional fusions.
2. If CHM contains conditional fusions, then ensure the closure of the condition chain by introducing proper separations (if necessary).
3. Generate the maximal compatibility classes.
4. Check the disjointness of the classes.
5. If the classes are not disjoint, then check if there are any classes to be neglected without hurting the closure. Such classes should be left out. If this neglecting hurts the closure, check the effect of neglecting only a subclass or rather separating an other class. This step is task-dependent and requires trials.
6. Ensuring the disjointness and the closure of classes may result in increasing number of classes.
7. If extremely great number of not disjoint classes are obtained, then the inequality $M \leq P \leq \min(n,k)$ [13] can provide information on the „goodness” of the actual results in order to avoid further trials.

- $P$: the number of classes in the actual result,
- $n$: the number of nodes in the input graph for the allocation,
- $k$: the number of maximal compatibility classes generated from the initial CHM,
- $M$: the number of maximal compatibility classes representing the simplest closed cover.
8. Each class of the result represents those operations from the input graph of the allocation, which should be allocated in a common execution unit (component).

The flow chart of the algorithm is illustrated in Figure 11.
Figure 11 The flowchart of the allocation algorithm
3. Thesis

- In the high level synthesis of complex distributed systems, operating, maintenance, standard and safety-critical prescriptions have to be taken into consideration beside the task functionality.

- I have elaborated a new algorithm for the allocation in high level synthesis, which – unlike the existing tools- can handle and consider predefined boundary conditions for fusions, conditional fusions and separations of operations.

- The algorithm can be applied in the systematic design of safety-critical complex industrial distributed systems.

- The algorithm can be applied in the allocation of communication operations (detailed in Thesis 1) for influencing according to the resources.

- Based on the algorithm, developing new scheduling and allocation procedures have been started.

- The method and the algorithm summarized in this thesis are my own results.

- The publications containing the results of this thesis: [S9], [S10].
Acknowledgement

I am very grateful to my supervisor, Dr. Péter Arató and to my advisor, Dr. Géza Móczár for their help in my research and developing work.

The support of the Hungarian Scientific Fund (OTKA K72611) and New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002) IKT-P5-T3 are gratefully acknowledged. This work is also belonging to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project.
Publications

Referred journal papers in English


S3. György Pilászy, György Rácz, Péter Arató, „The effect of increasing the latency time in High Level Synthesis”, Periodica Polytechnica, lektorálás és megjelenés folyamatban

Further journal papers and book chapters


S5. Pilászy György, Horváth Tamás, "Digitális technika feladatok és megoldásuk", Elektronikus formában közzétett egyetemi jegyzet az MSC villamosmérnökök szakra jelentkező hallgatók számára


Conferences


Other publications

Research report (internal)
References


[26] Gergely Suba, Hierarchical pipelining of nested loops in high-level synthesis, 2013, Periodica Polytechnica, lektorálás és megjelenés folyamatban


[29] Andrew S. Tannembaum: Számítógép hálózatok, Panem, 1999

