Italian Workshop on Embedded Systems (IWES 2017)
Work in Progress Session

CC4CS: A Unifying Statement-Level Performance Metric for HW/SW Technologies

V. Stoico[1], V. Muttillo[1], G. Valente[1], L. Pomante[1], F. D'Antonio[2]
[1] Università Degli Studi Dell'Aquila – Center of Excellence DEWS, L'Aquila, Italy
vincenzo.stoico@student.univaq.it, vittoriano.muttillo@graduate.univaq.it, giacomo.valente@graduate.univaq.it, luigi.pomante@univaq.it
[2] Thales Alenia Space, Via Campo di Pile, L'Aquila, Italy
fausto.dantonio@gmail.com

Introduction: HW/SW Co-Design

- The adopted design methodology is of critical importance during the development of an embedded system.
- Working on a higher level of abstraction is crucial to evaluate alternatives and make better architectural choices.
- A Co – Design strategy is useful to improve design quality, design cycle time, and cost.

Classic Design

Co – Design

Introduction: MIPS

- Early performance estimation is a fundamental step.
- MIPS (*Million Instructions Per Seconds*) is one of the most common metric for performance analysis. [1]
- Useful for comparing two microprocessors with the same ISA (*Instruction Set Architecture*) but it is pointless to compare ones that have different Instruction Set .
- **Problem**: most of the performance metrics are too bonded to low level details.

Introduction: CC4CS

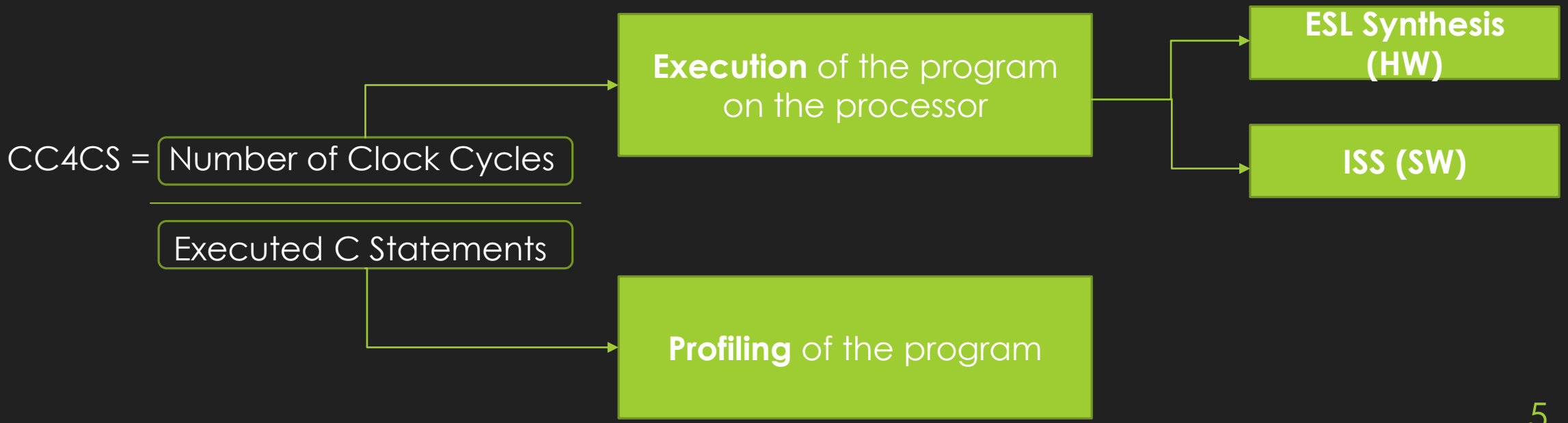
- **Possible Solution**: Analyze the meaningfulness of a metric related to C programming language statements suitable both for SW and HW implementations
  - For this the metric targets also HW/SW co-design
- CC4CS (*Clock Cycles for C Statements*) is the ratio between the number of clock cycles required by the processor to run an application on the number of executed C statements.
- A framework that helps to calculate this metric has been realized.

$$CC4CS = \frac{\text{Number of Clock Cycles}}{\text{Executed C Statements}}$$

How to calculate numerator and denominator?

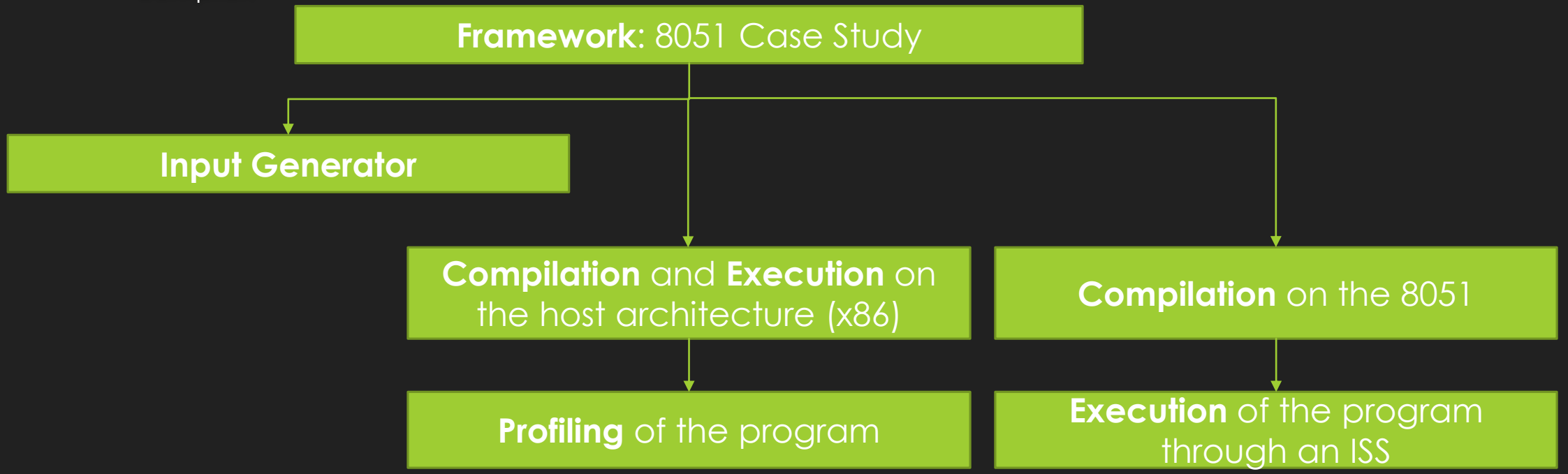
Validation: Framework

- From the definition of the metric has been outlined the working process of the framework.



Framework: 8051 Case Study

- In order to perform a very first analysis, an instance of the framework has been implemented using the original Intel 8051 microcontroller as first target.
  - The original 8051 core is a 8-bit CISC CPU one, has 128 byte of Internal RAM 64K of internal ROM
    - Without cache and external memory it is a good starting point since there are limited degree of freedom for the compiler.



Framework: Input Generation

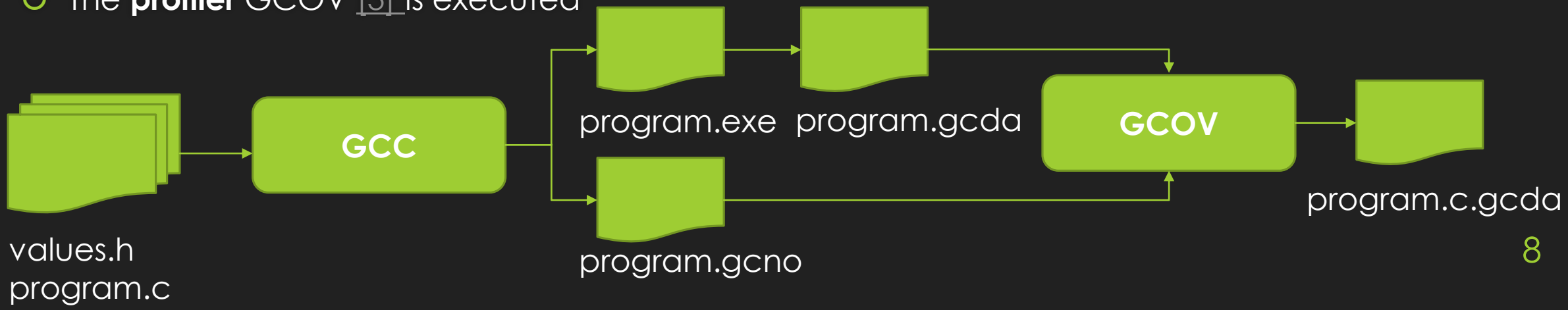
- Is based on a module that automatically generates constrained random inputs for a given function.
- For each parameter, the user is asked to insert a range for meaningful values (min, max) and the number of values to be randomly generated.
- In case of functions that requires more than one variable, the Cartesian Product of generated values is provided.
- For each combination a header file is created that contains the values of a single combination.

Framework: Profiling on x86

- The program is **compiled** using GCC [4]:

```
gcc program.c -Iincludes/ -Iincludes/values/ -fprofile-arcs -ftest-coverage -o program.exe
```

- From the compilation two files are created: program.exe and program.gcno.
- The executable is launched and **`program.gcda`** is created.
- The **profiler** GCOV [3] is executed



Framework: Profiling Results

```
3:   46:          if(a[i] > a[i+1])
-:   47:          {
2:   48:              swap(i,i+1);
2:   49:              is_sorted = 0;
2:   50:              currentSwap = i;
-:   51:          }
```

Framework: Simulation on 8051

- The program is **compiled** on x86 with SDCC [5]:

```
sdcc program.c -Iincludes/ -Iincludes/values/ --mmcs51 --iram-size 128
```

- A file `.ihx` is created from the compilation. 8051sim needs a `.hex` file to perform the **simulation**, so a conversion is done through `packihx` command:

```
packihx program.ihx > program.hex
```

- At the end, the ISS is **launched** with the `.hex` and a text file where will be stored the statistics of the simulation:

```
8051sim program.hex program_simReport.txt
```

values.h
program.c

program.hex

program_simReport.txt

Analysis: 8051 results

- To analize CC4CS has been created a benchmark composed by 10 well-known algorithms.
- The metric has been evaluated with respect to 10.000 input files per function.
- Different data types has been considered (int8, int16, int32, and float).
- The following table shows the statistics calculated using the 8051:

Method	Min	Max	AM[a]	SD[b]	90[c]	95[d]
Int8	58	410	117,8	47,4	170	176
Int16	80	453	161,4	67,5	265	297
Int32	104	760	227,9	88,7	354	400
Float	4	1301	537,7	267,6	969	1173

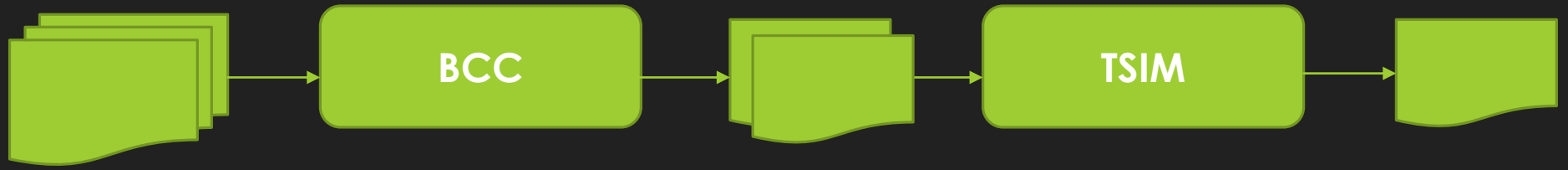
[a]AM: Arithmetic Mean, [b]SD: Standard Deviation, [c]90: 90[h] percentile, [d]95: 95[h] percentile

Framework: SparcV8 Leon3 Case Study

- In order to analyze a different microprocessor, has been implemented an instance of the framework using Leon3 as target microprocessor.
  - Leon3 is a 32 bit synthesizable soft-processor that is compatible with SPARC V8 architecture based on a Harvard Architecture, has two different caches (one for instructions and one to store the data).
- Cobham Gaisler free offers the evaluation version of **TSIM** that is an ISS for this microprocessor. This version has been used to perform the software simulation in our case study.
  - The evaluation version of TSIM/LEON3 implements 2*4 KiB caches (not removable), RAM size of 4096 KiB and a ROM of 2048 KiB.

Framework: Simulation on Leon3

- The program is **cross-compiled** with **BCC** (Bare-C Cross Compiler)[6]:
  - `sparc-elf-gcc program.c -Iincludes/ -Iincludes/values/ -o0 –o program_sparc`
- The executable program_sparc has been created.
- **TSIM** is lauched:
  - `TSIM program_sparc –c tsim_cmd > program_TsimReport.txt`



values.h
program.c

program_sparc
tsim_cmd

program_TsimReport.txt

Analysis: Leon3 results

- In the table are reported the statistics calculated using the Leon3:

Method	Min	Max	AM[a]	SD[b]	90[c]	95[d]
Int8	11	2197	193	304	536	721
Int16	12	2194	291,9	401,5	644	1322
Int32	23	2194	437,1	512,0	1047	2053
Float	28	2200	481,7	516,9	1326	2058

[a]AM: Arithmetic Mean, [b]SD: Standard Deviation, [c]90: 90[h] percentile, [d]95: 95[h] percentile

Framework: CC4CS in HW Domain

- In the work "*A Survey and Evaluation of FPGA High-Level Synthesis Tools*" were analyzed three academic (DWARV, Leg-Up, Bambu) HLS tools and a commercial one.
- Functions taken from CHStone benchmark Suite and a part from DWARV and BAMBU has been used to evaluate the tools. For each function there is a built-in input.
- This work provides, for each function, the number of clock cycles required during the execution with Altera Stratix V and Xilinx Virtex-7 done with these tools.
- They used the following default target frequencies: 250 MHz for BAMBU, 150 MHz for DWARV, and 200 MHz for LEGUP. For the commercial tool, they decided to use a default frequency of 400 MHz.
- Its easy to obtain the CC4CS doing a profiling of the functions to get the number of executed C statements and next calculate the ratio.

Framework: CC4CS in HW Domain

- Two sets of experiments to evaluate the compilers has been done.
- In the first experiment, they executed each tool using all of its default settings, which they refer to as *standard-optimization*.
- In the second experiment, they manually optimized the programs and constraints for the specific tools (by using compiler flags and code annotations to enable various optimizations) to generate *performance-optimized* implementations.
- In the following tables are shown statistics calculated on the sample obtained by computing CC4CS for each function.

Analysis: CC4CS in HW Domain

- Results in *standard-optimization* case:

Method	Min	Max	AM[a]	GM[b]
Commercial	0,1173	4,0064	1,137464	0,75815
Bambu	0,0148	7,357	1,179243	0,46834
DWARV	0,0177	4,4854	1,253125	0,65008
LegUp	0,0007	7,404	1,339010	0,48307

[a]AM: Arithmetic Mean, [b]GM: Geometric Mean

Analysis: CC4CS in HW Domain

- Results in *performance-optimized* case:

Method	Min	Max	AM[a]	GM[b]
Commercial	0,1639	4,0064	0,85406	0,538482
Bambu	0,0148	3,3233	0,69188	0,334336
DWARV	0,0142	6,6672	1,6322	0,639132
LegUp	0,0007	1,5473	0,535986	0,281288

[a]AM: Arithmetic Mean, [b]GM: Geometric Mean

Analysis

- So, given a trace of execution, thanks to CC4CS it is possible to immediately estimate how much time will require 8051 (and other processors for which the metric has been already evaluated) to execute it
  - e. g. Given a function using only int8 and a specific input, and supposing that by means of a host-based profiling the number of executed C statements are 200, according to the 95^{h} percentile, it is possible to estimate that 8051 will require form 58*200 to 176*200 clock cycles
    - With a 20 Mhz clock it is an interval between 0.58 ms and 1.76 ms
- It is worth noting that estimation errors have to be still analyzed in details.
- But it is also worth noting that, having CC4CS for several processors, a comparison of estimated execution times is straightforward since it is based only on the trace provided by a host-based profiling.
  - This is the ultimate goal and it would be still more powerful considering also the oppotunity to directly compare HLS-based HW implementations.

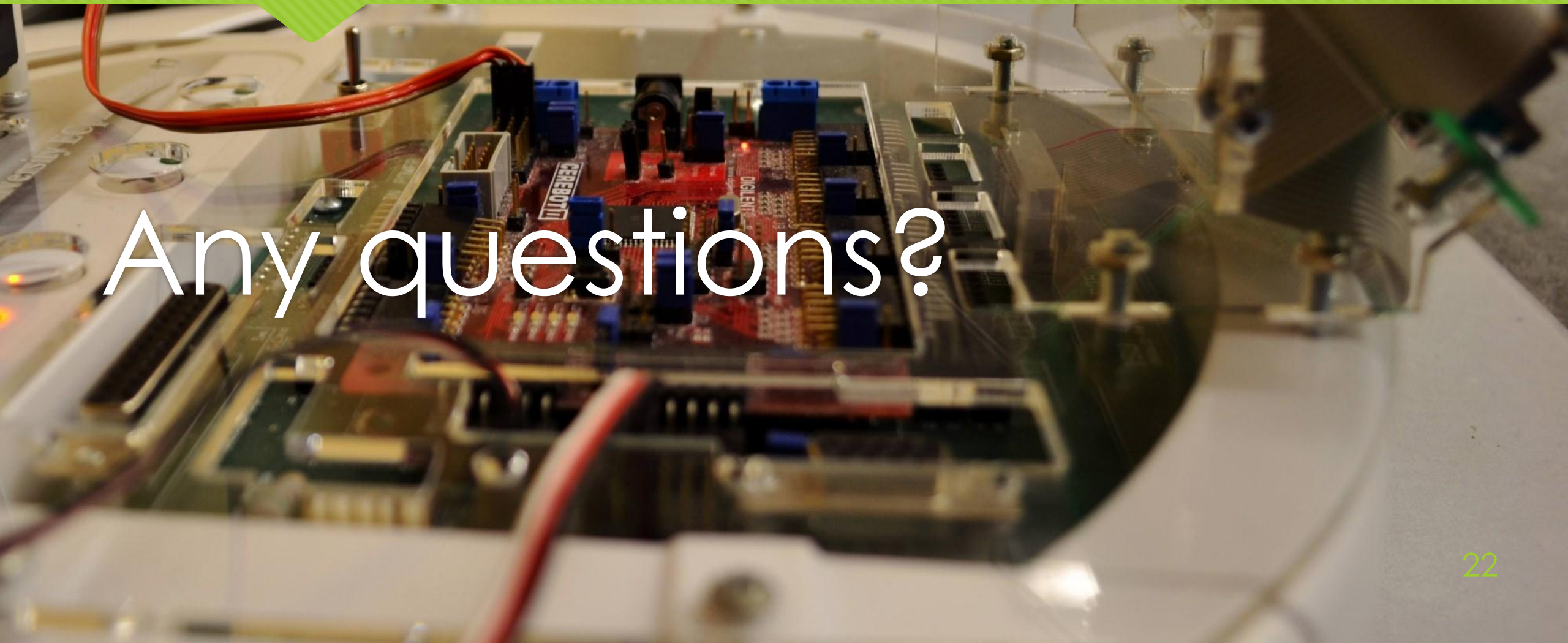
Conclusion and Future Works

- The metric seems to be good enough to allow reasoning about the suitability of a processor with respect to given timing constraints and a comparison among processors
  - Estimation errors have to be still analyzed in details.
- A more relevant testbech (maybe someone used internationally for similar purposes) should be adopted
- Some other analysis and considerations related to the HW characteristics of the processors and compiler optimization have to be done.
- Evaluate CC4CS also for C functions directly implemented in HW by means of High Level Synthesis techniques.
  - This work avoids reasoning about assembly code related to C statements so it is possible to use CC4CS for HLS

References

1. D.J. Lilja, Measuring Computer Performance, A Practitioner's Guide, Cambridge University Press, New York, USA, 2000.
2. Dalton Project: 8051 microcontroller, University of California, http://.ann.ece.ufl.edu/i8051/ , Accessed 26 April 2017.
3. GCOV Profiler , https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, Accessed 26 April 2017.
4. GCC GNU Compiler Collection, https://gcc.gnu.org/onlinedocs/gcc, Accessed 26 April 2017.
5. SDCC, http://sdcc.sourceforge.net/doc/sdccman.pdf, Accessed 26 April 2017.
6. BCC, http://www.gaisler.com/doc/bcc.pdf, Accessed 26 April 2017.
7. A Survey and Evaluation of FPGA High-Level Synthesis Tools, https://panda.dei.polimi.it/wp-content/papercite-data/pdf/TCADHLSEVAL2016.pdf, Accessed 5 September 2017.

THANKS!



Any questions?