Automatic generation of Infineon microcontroller product configurations

Prateek Chandra, Infineon Technologies India Pvt. Ltd, Bangalore, India
(*prateek.chandra@infineon.com*)

Leily Zafari, Infineon Technologies AG Neubiberg, Germany (leily.zafari@infineon.com)

Boyko Traykov, Infineon Technologies AG Neubiberg, Germany (boyko.traykov@infineon.com)

***Abstract*— The ConfigSector (CFS) flow is a methodology developed to automate the process of creating different product configuration for an Infineon Microcontroller. A microcontroller can be configured for its features by tweaking the firmware which goes into the embedded flash. This flow fetches data from cross-organizational sources like marketing requirements, concept engineering, design, firmware and test and then generates a structured output which can be stored in the respective place in the embedded flash of the chip. This methodology leverages full control over a device which needs to be delivered to a particular customer, drastically reducing time to market with an added advantage of tracking inherent discrepancies**

Keywords*— *Configuration Sector; CFS; single-source data; productivity gain; cross consistency of inputs

I. INTRODUCTION

The device Configuration Sector (CFS) is a dedicated embedded flash area which stores the information on how single SoC can be configured to serve different customers and address multiple markets.

Figure 1, shows an SoC which has multiple features. However, not all the features are requested by one particular customer. A varying set of customers will have requirements distinct from each other. In order to address all the requests, a microcontroller is designed keeping in mind the superset of all the features demanded. Later the microcontroller can be configured and offered to different customers as per their requirement. These configurations have a minuscule difference in their configuration sector data which finally goes in the embedded flash memory. Additionally, every configured product being delivered is individually stamped with unique chip identification, repair information and trimming values.

For each of the configured product, manual generation of configuration sector data is a redundant job. The manual process requires capturing information from different sources, interpreting it and finally formalizing into a consistent structure compatible with the firmware. Thus, the approach is prone to be inefficient and inconsistent because of the inherent complexity of collecting information from multiple sources.

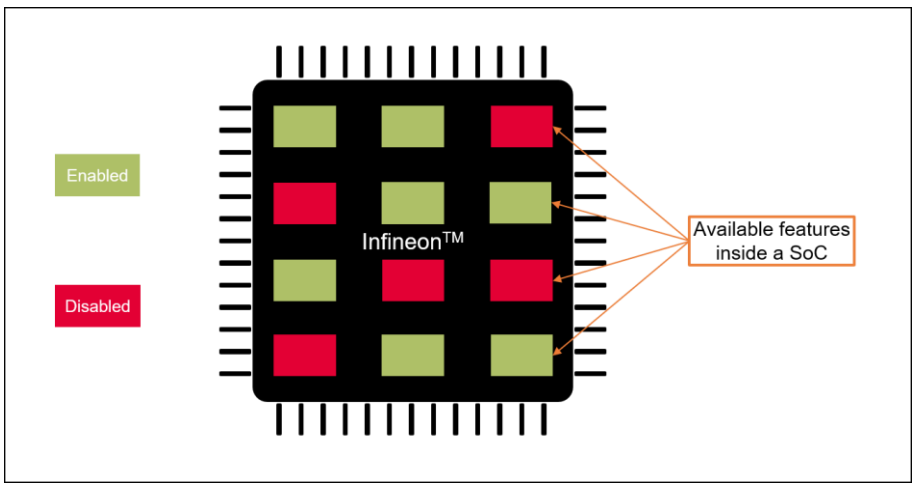

Figure 1: Configurable features inside a microcontroller

II. MOTIVATION

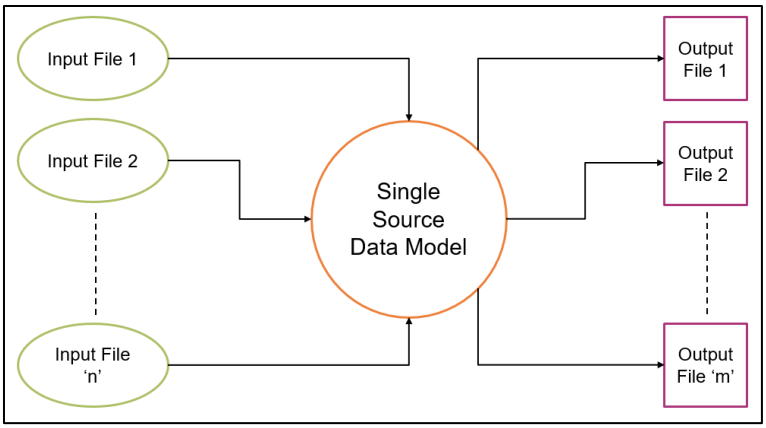

Figure 2 – Aim of the methodology

As it is established that the manual approach of generating configuration sector data is time-consuming, inefficient and prone to human errors, there are many to affirm its inconvenience. For example, the firmware engineer, who is responsible for delivering configuration sector data, has to spend extra effort in reviewing and then manually rework the errors and inconsistencies discovered in verification and test.

The configuration sector data which goes into the embedded flash is multifarious. It consists of data related to modules of a microcontroller, their instances inside the design and marketing options. Each of these data sets is owned by different groups within the organization which are based at different geographical sites. Thus, keeping the data consistent and updated to be used productively for manually generating a product configuration becomes agitating for the firmware engineer. Thus, it is a rational choice to have all these data sets available at a central location. Moreover, there has to be an intermediate stage to encapsulate all the relevant data fetched from the central location as a single source for subsequent processing.

With the ConfigSector flow, we aim to establish an automated flow to generate multiple product configurations using single-source data available from a central version-controlled system in a form compatible with the format of the firmware. At the same time, a human-readable format is also be generated to ensure ease of use. The flow should encapsulate the diverse range of data into a data model. Assuming this data model, virtually to be a single source, the flow also aims to ensure consistency of data between multiple input sources and the multiple output files produced.

III. CONFIGURATION SECTOR FLOW

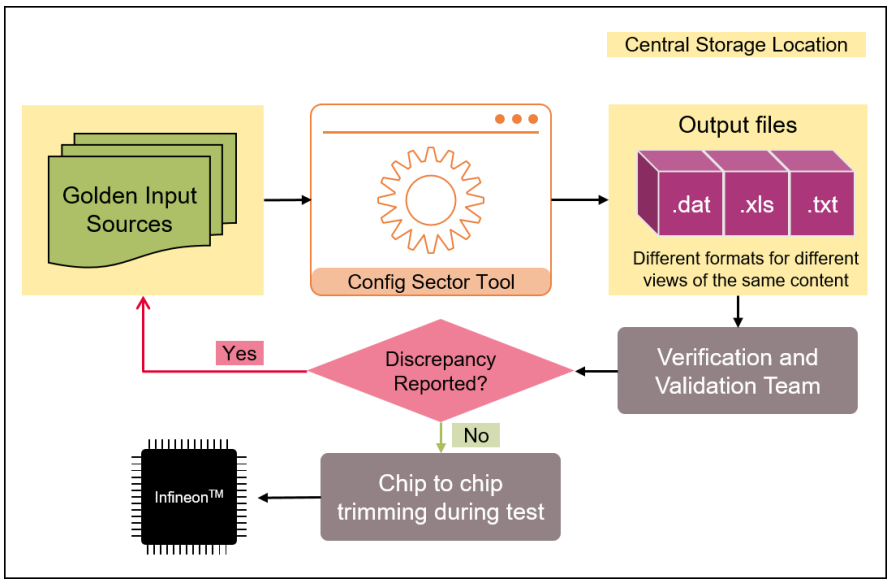

Figure 3 – Diagram of the ConfigSector Workflow

The foundation of any methodology flow is based on three major pillars – the input that is available, the way the input gets manipulated and the output that it renders at the end of execution. Similarly, the input for generating configuration sector data is captured from multiple golden sources, then it gets processed through the underlying algorithms and produces the expected output. The configuration sector values are generated in the form ".dat" files. These values are then loaded to the embedded flash after it has been verified and validated by the SoC verification and validation group. If here, it is found that the configuration sector values should be modified to have a better performance or because they are not correct, the golden sources will be changed accordingly and the flow gets repeated. Figure 3 reveals that the generated output which gets loaded in the microcontroller is completely free of anomalies.

In the centrally stored input sources, data for each microcontroller or it's to be configured derivatives are supposed to be version baselined. At the front end of the ConfigSector tool, a graphical user interface allows the users to specify the correct baseline version for their microcontroller. At the backend, the baseline will have the data for all possible configurations of a particular microcontroller indexed. Subsequently, the tool would browse through all the indexed items to fetch the relevant particulars and use it for the generation of configuration sector data. The generated output files can also be then released in the central location as one of the components.

In the following sub-sections each of these foundational bases has been described:

A. Input Files

As explained earlier, there are multiple input files which are fed into the tool and each has its own significance. There will be files consisting of the structure of the to be generated output, details about different module designs, number of their instances inside the microcontroller and configuration sector values for some special registers. These files which together form the bill of material for particular microcontroller contain the largest chunk of the input data. In addition to this, the marketing option sheet also contains some information about the different product configurations of a microcontroller. Ultimately, there will be some configuration sector values for which no hardware relevant golden sources can be found. These values will be provided in a separate file. Ultimately, comes the most important yet risky input file which can be used to override any existing configuration sector values. It is used in a scenario, where the files storing module design data has been frozen and can no longer be changed. In order to maintain consistency, this file will also be version controlled in the input baseline, if used.

B. Graphical User Interface

The Graphical User Interface (GUI) can be invoked from the command line and has all the necessary text fields to capture the name of the silicon and the corresponding baseline version to look for the input data.

The GUI provides buttons to fetch the lookup paths for different input files and to trigger the generation of the required output. There is also an option to prioritize the lookup from a particular item out of all indexed items present in the baseline.

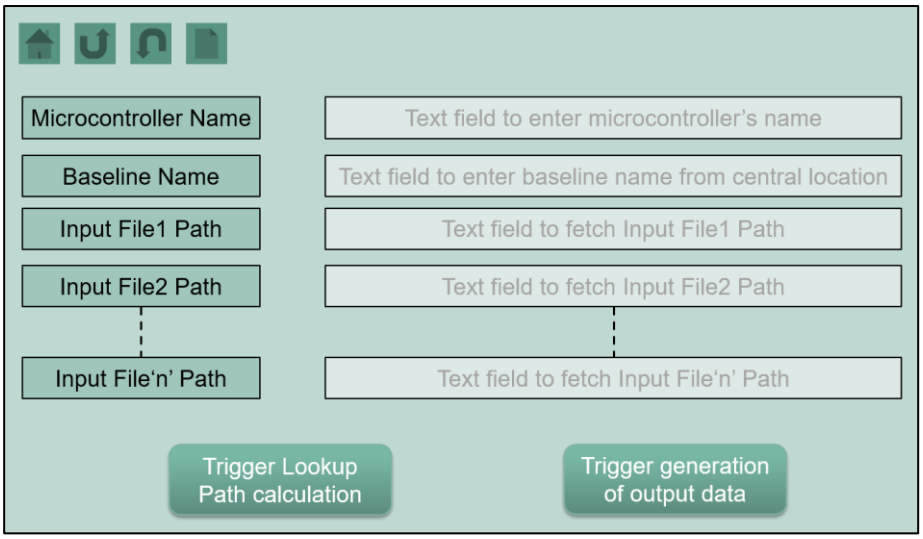

Figure 3 – GUI Wireframe

Figure 3 shows the wireframe of the interface available at the user's end. As proposed in the motivation to this paper, that all the gathered data from multiple sources to a single source data model, it is now that this operation gets executed i.e., just after the generation is triggered.

C. Output Files:

As already discussed, once the required data is gathered in a singsle source, the tool allows generating different views of the configuration sector data without redundant effort. It takes a minimal effort to develop generators for each required view. In addition to the previously mentioned XML file containing the paths, three other views are generated. Files in ".dat" format are used by testers and SoC verification group. They are converted to hex format through a compiler and loaded on the chip. Files in ".xls" and ".txt" are also generated as user-friendly formats for reviewing the data and also tracing the origin of the configuration sector data amongst multiple input files.

IV. FEATURES

The ConfigSector tool is a platform-independent tool and can be used on both Windows and Linux as the entire methodology has been implemented in Python programming language. At the algorithmic level, the tool not just fetches input values and restructures it for the correct output format but also performs some automatic operations based on given rules in the input files. For example, the calculation of a unique chip ID is done for each configured derivative using the rules defined in the marketing option sheet. The tool also calculates cyclic redundancy check values to ensure that the correct configuration sector data has been loaded.

The marketing option sheet, as mentioned before, describes some of the configuration sector data for each microcontroller derivative. However, not all derivatives in this sheet will go to production, but it gives the marketing team the independence to make different possible configurations and discuss it with their customers. Before the configuration sector data is generated, the desired derivatives should be identified as productive in the sheet. For each of the productive derivatives, all three views (".dat", ".txt", ".xls" files) mentioned above will be generated. As a bonus, a file containing paths to all the required inputs is also generated for the user to backtrack the origin of each value in the generated configuration sector data.

V. RESULTS

Earlier to the inception of ConfigSector flow, the embedded flash data from different groups located in different places were collected by a single person which certainly is the sluggish and fallible approach. For making the first shot of data for each derivative, the time spent would be more than a week, and then each change request for new silicon would also need two hours to two days effort to update the data. With the automated ConfigSector flow, the overall effort is reduced to a single execution of approximately sixty minutes. Needless to mention that the consistency of the generated data only adds to the robustness of the ConfigSector flow. Furthermore, it is possible to keep track of the products being ordered by the customers as each product is assigned with a unique ID.

VI. BENEFITS

There are numerous benefits of using the ConfigSector flow. The most pronounced of all is the reduced development time. Since the data owners are situated at a geographically different location, it is extremely difficult for a firmware engineer to have an interface with all of them. The approach alleviates this problem as the data is now being fetched from a central location where individual owners are responsible to keep the data updated. Another advantage of the approach is that no matter how many products are chosen, the user should just run the flow once for a given list of desired derivatives. So any synchronization between the marketing group and the configuration sector users is not necessary anymore, which avoids any inconsistencies.

VII. CONCLUSION

There is a basic necessity that redundant jobs need not be done manually if a solution to automate it can be curated. The ConfigSector flow is one such example. The ConfigSector flow has been designed by following "correct-by-construction" methodology which actually means that while building the flow a major part of the total effort was spent in conceptualising the solution. The flow ensured that there is a steep cut down on the manual work of reading all input files then creating the ".dat" files. Also, the output is free of human errors creeping in at any stage.

The flow also addresses the verification gap owing to the single-source methodology, which otherwise would have been reported by customers. Thus, it also instils a factor of trust in the customers. In a scenario where there is a need for late changes of values in the module specification, the user need not start creating the ".dat" file from the first word. The user only needs to re-trigger the tool. At present, the tool is productively being used by the firmware engineering group and has resulted in an appreciable gain of productivity for the gain of organization.

REFERENCES

[1] Deeksha, "Boot sequence for an arm based embedded system", https://www.embeddedrelated.com/showarticle/118.phpecb

[2] Ecker Wolfgang, Velten Michael, Zafari Leily, Goyal Ajay (2014), "The metamodeling approach to system level synthesis". Dresden, Germany. 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), 24-28 March 2014.

[3] Python Software Foundation, "Python 2.7.16 documentation", https://docs.python.org/2/index.html

[4] Python Software Foundation, "Other Graphical User Interface Packages", https://docs.python.org/2/library/othergui.html

[5] Hermann Michael, "About PyQt", https://wiki.python.org/moin/PyQt