applied sciences

MDPI

Article

Securing Network Information System Design: An Efficient Tool for DSP Undocumented Instruction Mining

Xingcan Zhang [1], Zhe Chen [1], Jiawen Ye [1], Huan Li [1], Jian Wang [1,*], Changlong Liu [2] and Bin Li [2]

[1] School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2] The 54th Research Institute of China Electronics Technology Group Corporation, Shijiazhuang 050081, China
* Correspondence: wangjian3630@uestc.edu.com.cn

Abstract: As recently studied, the undocumented instructions in embedded processors that may cause catastrophic results for devices have become one of the main threats to system security. To tackle this issue, in this paper, we propose an undocumented instruction mining tool for digital signal processors named DSPUIM that can find out the undocumented instructions from the frequently used Digital Signal Processors (DSP) in network information systems. First, we analyzed the characteristics of the DSP instruction format to compress the instruction search space and improve the instruction search speed. Second, according to the public instruction set of DSPs, we built an instruction disassembly framework that helped us to identify all the undefined instructions. Finally, by testing the executability of undefined instructions automatically, we obtained the undocumented instructions for target DSPs. To demonstrate the effectiveness of our tool, we applied it on ten DSP processors of Texas Instruments (TI) and mined 335 undocumented instructions from them within 5 min. Some undocumented instructions have malicious functions, such as changing registers and denial of service, posing a security threat to the network devices using DSPs.

Keywords: undocumented instruction; DSP; disassembly; instruction analysis

check for updates

Citation: Zhang, X.; Chen, Z.; Ye, J.; Li, H.; Wang, J.; Liu, C.; Li, B. Securing Network Information System Design: An Efficient Tool for DSP Undocumented Instruction Mining. *Appl. Sci.* **2023**, *13*, 3931. https://doi.org/10.3390/app13063931

Academic Editor: Luis Javier Garcia Villalba

Received: 19 January 2023
Revised: 12 March 2023
Accepted: 15 March 2023
Published: 20 March 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the continuous development of information technology and the increasing demands for data processing, processors have become an indispensable part of production in information systems [1,2]. As one of the most important processors, the digital signal processor (DSP) has powerful data processing capabilities and high operating speeds with low energy consumption [3,4]. Therefore, it is widely used in many fields such as communication [5,6], remote sensing [7], autopilot [8], medical equipment [9,10], environmental monitoring [11], Internet of Things (IoT) [12], and consumer electronics, as shown in Figure 1. As reported in [13], the global DSP chip market size is estimated to be USD 3872.6 million in 2022, and it is expected to be USD 5746.9 million by 2028. Such a huge market also attracts attackers to take advantage of the loopholes to achieve their illegal purposes.

At present, the instruction set which works as an important interface connecting processor hardware units and software is no longer trustworthy [14]. Duflot L. [15] first found that there are undocumented instructions in processors. Thereafter, increasingly more undocumented instructions have been mined, such as FDIV [16], F00F [17], and VIA C3 [18]. These undocumented instructions may lead to malfunction of the processors, such as change functions [16], stopping working [17], and allowing attackers to obtain illegal privileges [18], endangering the security of tens of millions of devices. With the increasing number of undocumented instruction security incidents, it is necessary to mine potential undocumented instructions from processors as a preventive measure to protect processor security.



Figure 1. DSP applications in various domains.

In this paper, we propose an undocumented instruction mining method for DSP that can quickly obtain DSP undocumented instructions within minutes, providing a defense tool for DSP security.

Our research motivations were as follows:

(1) As an important processor device in network information systems, the research on the security of DSP is relatively weak. Researchers have focused on applying DSP to security devices [19,20], not the security of DSP itself.

(2) There is no undocumented instruction mining method in the current literature that applies to the characteristics of the DSP instruction set. Many researchers have proposed undocumented instruction mining methods mainly for the architecture of general-purpose processors rather than DSP.

The main contributions of our paper are twofold:

- We propose an undocumented instruction mining method for the DSP, which can find out the undocumented instructions on a variety of DSP processors within several seconds. It provides a platform for subsequent researchers to study the security of DSP.
- We performed functional analysis and classification on these undocumented instructions. Furthermore, we discuss the affection of these instructions when they are executed, and we find that partially undocumented instructions indeed pose a security threat to the corresponding processors.

The rest of this paper is organized as follows. In Section 2, we briefly discuss the related work. In Section 3, we detail how we implement the DSPUIM tool. In Section 4, we conduct extensive experiments to demonstrate the effectiveness of our method, and we perform functional analysis and classification for mined undocumented instructions. Finally, we conclude in Section 5.

2. Related Work

Thus far, the research on processor security is mainly focused on general-purpose processors, touching upon two computing architectures, i.e., Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC).

2.1. The Security of CISC

Although processors are ubiquitous in our lives, the research on verifying the security of processor instruction set has just begun in recent years. The complex instruction set architecture represented by the x86 architecture has a history of more than 40 years. It has a large number of instructions and various lengths of instruction words [21]. Undocumented instruction, circuit bugs, and backdoor are major security issues for processors.

Undocumented instructions are caused by inconsistencies between official documentation and real instructions. An error raised by the processor or an exception handled incorrectly by system software indicates the presence of undocumented instructions. Some researchers have put many efforts on mining the undocumented instructions of x86 CPU [22,23]. In the past few years, several search tools for undocumented instruction in the x86 architecture have been proposed and gradually improved. As a pioneer work, Domas proposed an undocumented instruction search tool for x86 architecture CPU called Sandsifter [24], which is the first automatic search tool for undocumented instructions. However, it suffers limitations on a high false positive rate. In 2018, Jianping Zhu et al. [25] proposed a comprehensive CPU security benchmark to evaluate the hardware security of processors and develop a tool mining undocumented instruction with higher speed compared to Sandsifter. UISfuzz [26] further reduces the search space based on Sandsifter. UISfuzz improves the mining efficiency by 5.57 times and reduces the memory overhead by 18.46 times compared to Sandsifter. The instruction search method named HEAM [27] proposed by Jiatong Wu et al. reduces its search time to 30% compared with the Sandsifter. They also proposed a classification method based on the instruction format to identify the actual function of undocumented instructions. In addition, Ermolov et al. [28] and Guang Wang et al. [29] also have contributed to the mining of undocumented instructions in terms of processor microcode and instruction decoder, respectively.

Bugs in the processors mean that there are involuntary implementation mistakes in the component, and these bugs are scattered in various mechanisms and functions of the processor, making it difficult to implement automated and formal detection tools. Memory Sinkhole [30] in 2015; Meltdown [31], Spectre [32], TLBleed [33], PortSmash [34] in 2018; and MDS [35] in 2019 are all vulnerabilities that violate the security model of the processor design architecture. Some researchers have proposed some defense methods [36–38] against Specter, Meltdown, and their variants. SPECS [39] is a lightweight dynamic detection system for processor bugs with a defense efficiency of 86% and less than 5% additional hardware and no software run-time overhead.

The backdoor of the processor has the characteristics of high authority and strong concealment. If the backdoor is implanted in the processor, it will cause great disaster to the upper information system. The backdoor in VIA C3 [18] mentioned above is a typical backdoor using undocumented instructions. In terms of processor backdoor defense, researchers have focused on the microcode level of the processor [40,41].

2.2. The Security of RISC

With the development of processors, increasingly more special instructions are added to the CISC, which greatly increases the cost of hardware design [42,43]. As a result, the RISC processor has gradually become the mainstream in embedded systems due to its advantages of flexible instructions and fast operation speed [44,45]. In terms of research on security testing of the RISC instruction set, Reid [46] converted the ARM-v8 instruction set into a machine-readable format for formal verification of ARM processors. Further in depth, iScanU [47] is an undocumented instruction search tool for RISC processors, taking only a few hours to search on faster systems, but it may show false positives for privileged instructions. Strupe et al. [48] proposed an undocumented instruction search tool specifically for ARM-v8 processors named Armshaker, which is publicly available as an open-source software, enabling users to audit their systems for undocumented instructions. They conducted a detailed analysis of the undocumented instructions but did not analyze the performance of the tool.

In terms of research on the security mechanism of RISC processors, representative security mechanisms of RISC-V hardware and architecture include hardware and physical access security, ISA security extensions, memory protection, side-channel attack protection, etc. [49–51]. In 2020, Nils Wistoff et al. [52] proposed a low-cost method for preventing covert channels in the RISC-V architecture. The method has a time overhead of about 320 cycles and a hardware overhead of 1% or less, all of which are negligible. Escouteloup et al. [53] proposed a new security instruction set based on the RISC-V ISA, through which software attacks that exploit hardware bugs can be circumvented.

Table 1 shows a summary of current processor defense approaches, which we compare in terms of instruction set type and runtime. To the best of our knowledge, the current literature proposes undocumented instruction search methods only for the instruction set architectures such as x86, ARM, and RISC-V, which do not apply to the characteristics of the DSP instruction set. Therefore, the proposed undocumented instruction search method for DSP is a gap in the current literature.

Table 1. Comparison between defense approaches for processors.

Ref#	Year	Approach Name	Approach Target	Instruction Set Architecture	Running Time Overhead
[39]	2015	SPECS	bugs	OpenRISC 1200	-
[24]	2017	Sandsifter	undocumented instructions	x86	343 min
[26]	2019	UISfuzz	undocumented instructions	x86	60 min
[47]	2020	IScanU	undocumented instructions	ARM, RISC-V	24 h
[48]	2020	Armshaker	undocumented instructions	ARM	-
[52]	2020	fence.t	covert channels	RISC-V	0.32 s
[53]	2021	RV32S	secure instruction set	RISC-V	-
[28]	2021	Red Unlock	microarchitecture	x86	-
[14]	2021	IMSC	undocumented instructions	MIPS32	real-time
[29]	2021	DFSGen	instruction decoders	MIPS, ARM, RISC-V	83 min
[27]	2022	HEAM	undocumented instructions	X86	103 min

3. Method

3.1. Preliminary

Before presenting our method, we first provide the structure of the entire instruction space. As shown in Figure 2, I represents all possible instruction encodings, the number of which is determined by the maximum length of the instruction. I_p stands for the public instruction set, containing all executable instructions indicated in the official document. Typically, a public instruction set I_p only covers a fraction of possible instruction encodings I, and the rest of undefined instruction encodings are classified to another set I_u. I_u can be further divided into two subsets, one is an illegal non-executable instruction set I_i, and the other is the illegal executable instruction set I_h. We aim to search the undocumented instruction I_h hidden in the undefined instruction space.

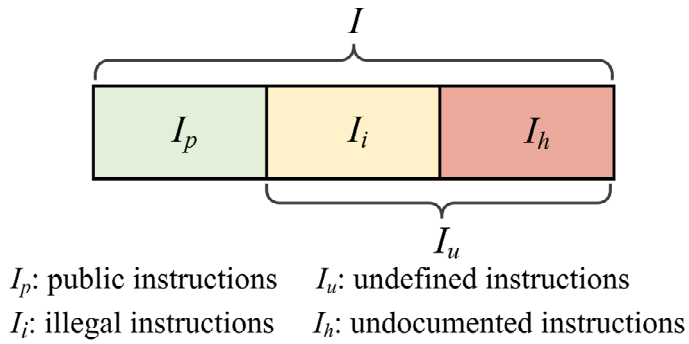


Figure 2. Instruction space description.

To search for undocumented instructions in DSP, we designed a tool named DSPUIM. The tool contains three parts. The first part is an instruction generation method based on valid instruction information bits; it is introduced in Section 3.2. The second part refers to the precise disassembly method; it is introduced in Section 3.3. Finally, we introduce the third part in Section 3.4, presenting the automatically executing method for tested instructions.

3.2. An Instruction Generation Method Based on Valid Instruction Information Bits

The straightforward way to generate instructions is brute force traversal instruction encodings. However, blind traversal consumes a large amount of time and resources. Taking 32-bit instructions as an example, walking through all encoding bits results in 4.2 billion instructions. To reduce the search space for instructions, we designed an instruction generation method based on the valid instruction information bits.

We performed a detailed analysis of the DSP instruction format. As a special-purpose microprocessor, DSP architecture and instruction set are optimized for signal processing-type algorithms. In general, the instruction format consists of conditional fields, operands, opcodes, and fixed values, etc., as shown in Figure 3. The instructions with the same function have a fixed value field, which is the judgment basis when a machine code is transformed to be the corresponding assembly instruction. The opcode and the fixed fields work together to determine the function of an instruction. The resource allocation and condition fields affect how instruction is executed. In detail, the condition field determines whether an instruction can be executed or not, while the resource allocation field decides which part of the hardware is used for the instruction execution.

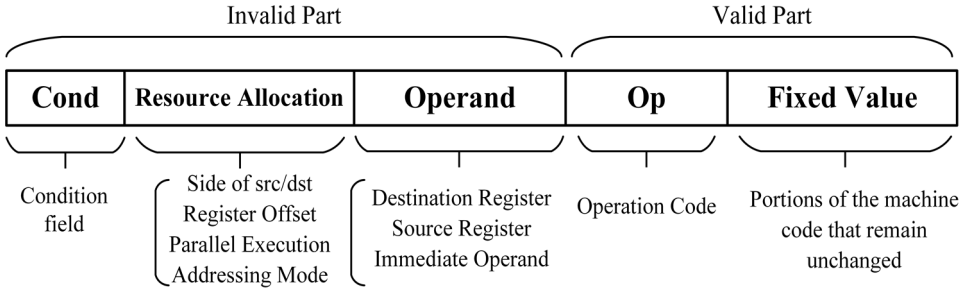


Figure 3. DSP general instruction format.

On the basis of the instruction format, the instruction generation method is as follows. First, we divided the bits in instructions into valid parts and invalid parts for the instruction searching algorithm. The valid part contains the operation code and fixed value, and the invalid part involves the operand field, condition field, and resource allocation field. Second, we traversed each bit in the valid part in a one-by-one manner while keeping the invalid part unchanged. As such, the number of search instructions was reduced from 2^m to 2^n. Here, m is the length of instruction and n is the number of valid bits. Note that the method

of reducing search space is based on the assumption that the undocumented instructions follow the format of public instructions.

3.3. Precise Disassembly Method

This subsection introduces a precise disassembly method for DSP processors. The purpose of disassembly is to translate machine code into corresponding assembly instructions. The set of instructions under test can further be reduced by removing the public instructions which can be disassembled.

Currently, there are two tools that can be used to disassembly and analyzing the DSP instruction set—one is CCS and another is IDA. However, both two tools have a significant limitation, i.e., they cannot disassemble all instruction codes exactly as the same as the rules in the official reference manual. Hence, we designed a disassembly tool specifically for DSP to accurately distinguish public instructions from undefined instructions.

As shown in Figure 4, our disassembly tool consisted of two parts. Firstly, the database based on the DSP instruction system was established. Secondly, the machine instructions were translated to corresponding assembly instructions according to a linear matching algorithm.

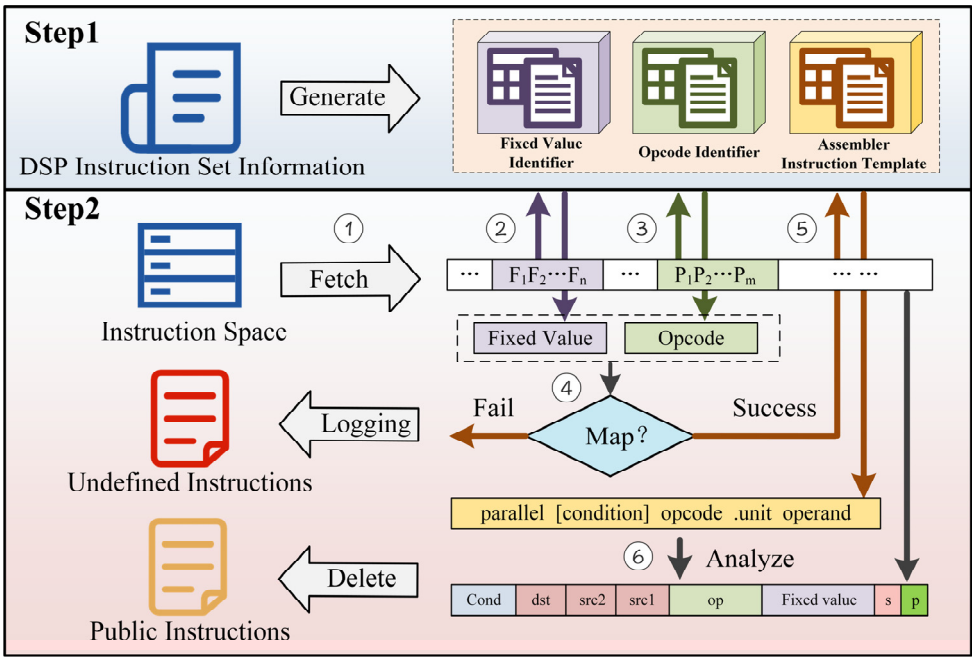


Figure 4. Disassembly method flow.

1. Establishment of Binary Map Database

The establishment of the database is the primary work for precise disassembling. The different databases correspond to different DSP instruction sets. As introduced in Section 3.2, each field of a machine code has a specific meaning. The fixed value determines the format of an instruction, and the opcode field indicates which operation the assembly instruction is. Therefore, in the database, we stored all the standard templates corresponding to public instructions and filled their fixed value field and opcode field.

2. Instruction Translation

Instruction translation is a process that translates machine codes to corresponding assembly instructions, which contains two steps. First, assuming that the machine code for disassembly analysis is h_s, we matched it with the fixed value and opcode in the database. In detail, if any fraction of h_s successfully matches with the fixed value in the database, we will further extract and compare the opcode field between the tested machine code and the

database. If they match with each other, the machine code is legal and can be disassembled. Otherwise, the machine code is marked to be an undefined instruction I_u. Similarly, if any fraction of h_s cannot be matched to the fixed value in the database, the machine code is marked to be an undefined instruction I_u.

Once the h_s matches the fixed value and the operation code identifier in the database, the corresponding assembly instruction template and machine code structure template in the database are obtained. According to the template, h_s can be segmented according to the instruction structure, and the instruction condition judgment, instruction unit, operand, and other information are obtained. Ultimately, the machine code h_s is translated into an assembly instruction.

After the instructions are disassembled, they can be divided into two categories according to the disassembly results. As shown in Formula (1), assume that the instruction set sent to the disassembly tool is H. H is just a part of I, because our method only traverses the effective fields in the instruction, rather than the whole instruction.

$$H = \begin{cases} H_p, H_p \in I_p(\text{public instructions}) \\ H_u, H_u \in I_u(\text{undefined instructions}) \end{cases} \quad (1)$$

3.4. A Fast Testing Method for DSP Undocumented Instructions

In this section, we provide a method to automatically test the undefined instruction H_u one by one. As shown in Formula (2), it is assumed that the undefined instruction set H_u contains M undefined instructions to be tested. In general, according to the result of instruction execution, H_{ui} can be further attributed to the undocumented instruction set H_h or the illegal instruction set H_i, as shown in Formula (3).

$$H_u = \{H_{u1}, H_{u2}, H_{u3}, \ldots, H_{uM}\} \ (1 \leq i \leq M) \quad (2)$$

$$H_u = \begin{cases} H_i, \ H_i \in I_i(\text{illegal instructions}) \\ H_h, \ H_h \in I_h(\text{ undocumented instructions}) \end{cases} \quad (3)$$

Our approach to distinguishing the undocumented instruction H_h from the illegal instruction H_i is as follows. Firstly, we make use of unused memory space to store an instruction to be tested H_{ui} followed by a jump instruction, so that the original program address can be returned after the instruction under test is executed. Then, we observe whether the processor can recognize the instructions under test and execute them. If the instruction under test is executable, the instruction belongs to undocumented instruction. Otherwise, the instruction is an illegal instruction, and an "illegal opcode" error will be reported on the development software or an unexcepted interrupt will happen on the DSP chip. In this way, the undocumented instructions and the illegal instructions can be quickly distinguished.

4. Experiment and Discussion

4.1. Experimental Setup

The experimental setup includes software and hardware. As shown in Figure 5, the hardware device consists of the target DSP processor and a PC. The software contains the DSPUIM tool written in JAVA and the DSP Electronic Design Automation (EDA) tool. The software runs on a PC with an AMD Core Ryzen7-5800 CPU operating at 3.20 GHz, with a memory of 16 GB and an operating system of Windows 10 64-bit. Texas Instruments (TI) is the DSP manufacturer with the highest market share; it is reasonable to add it to the research object list. We selected 14 different TI DSPs processors picking up from six series, i.e., C28x, C54x, C64x, C64x+, C66x, and C674x.

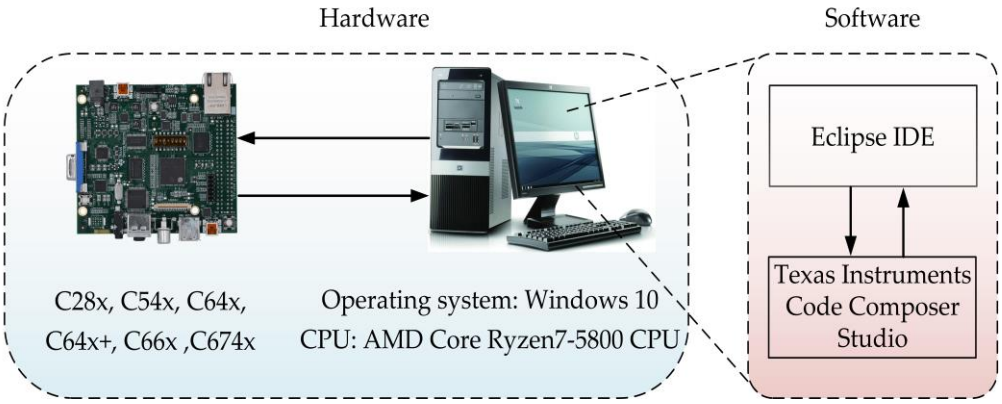


Figure 5. Experimental Setup of DSPUIM.

4.2. Undocumented Instruction Search Results

We applied our tool to each TI DSP and gathered the undocumented instructions. The experiment results are shown in Table 2. It can be seen that our DSPUIM tool successfully excavated 335 undocumented instructions in total—61 instructions were with 16-bit operation codes (TMS320F28335 and TMS320C5416), while the others were all instructions with 32-bit operation codes. Therefore, this method is valid. In addition, we found that more than 50% of the undocumented instructions excavated can be identified and disassembled by the CCS disassembly tool such as DDBG, EDBG, and PLD. However, these instructions are not recorded in the public instruction set documents of the corresponding processors. This means that the built-in instruction set of the CCS disassembly tool contains these undocumented instructions, indicating that the merchant has concealed information about these undocumented instructions.

Table 2. Undocumented instruction found.

DSP Processor Series	Processor Model	Number of Undocumented Instructions	
		OP-16	OP-32
C28x	TMS320F28335	52	49
C54x	TMS320C5416	5	0
C64x	TMS320C6416	-	15
C64x+	TMS320DM6437	0	19
	TMS320DM6443	0	19
	TMS320DM6446	0	19
	TMS320C6421	0	19
	TMS320C6424	0	19
	TMS320C6454	0	19
	TMS320C6455	0	19
	TMS320C6472	0	20
	TMS320C6474	0	20
C66x	TMS320C6678	2	18
C674x	TMS320C6748	2	19

4.3. Undocumented Instruction Function Analysis

According to the purpose of each undocumented instruction, we divided them into four categories: test, change function, denial of service, and temporarily unclear. Taking TMS320C6478 and TMS320C6678 processors as an example, we list their function of undocumented instructions in Table 3.

Table 3. Classification of undocumented instruction functions.

Classification of Instruction Functions	Disassembly Information in CCS	C6748 Processor Functional Description	C6678 Processor Functional Description
Test	DDBG	related to debugging, set the DBGM bit of the TSR register to 1	
	EDBG	related to debugging, set the DBGM bit of the TSR register to 0	
Change function	EFRW *	receive a word from EFI	-
	EFRDW *	receive double word from EFI	-
	B.S2 IFR	jump instruction, forced to jump to the address in the ARP register	
	B.S2 IER	jump instruction, forced to jump to the address in the ARP register	
	B.S2 ICR	jump instruction, forced to jump to the address in the ARP register	
	B.S2 CSR	jump instruction, forced to jump to the address in the ARP register	
	B.S2 ISTP	jump instruction, forced to jump to the address in the ARP register	
	B.S2 AMR	jump instruction, forced to jump to the address in the ARP register	
	B.S2 PCE1	jump instruction, forced to jump to the address in the ARP register	
	PLD	change the value of the base register to the sum of the value of the base register and offset/ucst5 shifted left by three bits	
	STP.S1	zero the destination register	-
	STWP	no obvious phenomenon	rewrite memory
Denial of service	EFRW *	-	core is hung
	EFRDW *	-	core is hung
Temporarily unclear	SWBP	no obvious phenomenon	
	MARK	no obvious phenomenon	
	EFSDW.L1	no obvious phenomenon	
	EFSW.L1	no obvious phenomenon	
	EFCMD	no obvious phenomenon	

* Note: The instruction format of EFRW and EFRDW are the same, but the execution effects of the C6748 and C6678 processors are different.

First, the test function instructions such as DDBG and EDBG were related to debugging. Since they can set the DBGM bit of the TSR register to be one or zero, we judged that the two instructions were built-in debugging instructions of the manufacturer that are not disclosed to users.

Second, the function-changing instructions included jump instructions and the instructions changing register. The jump instructions force the pointer to change to the address in the ARP register. If such instructions are maliciously used, the execution sequence can be disordered. On the other hand, PLD, STP, and STWP instructions will change the value of the target register. If they are used to maliciously modify register conditions, some important instructions will be invalidated. Therefore, these function-changing instructions pose a security threat to the DSP processor.

Third, the denial-of-service instructions involve EFRW and EFRDW instructions. The execution of the two instructions on the C6678 processor will cause the processor to crash, and a "Device core is hung" error will be reported. Once a processor crashes, it cannot work until it is rebooted. In addition, we find that the undocumented instructions with the same instruction format have different execution effects on different series of processors. For example, the execution of EFRW and EFRDW on the C6748 processor will rewrite the target register, which is completely different from the execution effect of the C6678.

Finally, there are some instructions whose functions are temporarily unclear, and we did not find any obvious phenomenon or register changes after the execution of these instructions. In addition, we found that although the format and function information

of some undocumented instructions cannot be queried in an instruction set document, it is described in another instruction set document, for example, the instructions related to the EFI interface, such as EFCMD, EFRW, EFRDW, EFSW, and EFSDW. These instructions are specified in the OMAP35x instruction set [54], rather than the C6000 series instruction set [55,56].

4.4. Quantitative Analysis

To quantitatively evaluate the efficiency of DSPUIM, we analyzed three aspects: undocumented instruction mining time, instruction search space compression effect, and hardware overhead.

4.4.1. Undocumented Instruction Mining Time Analysis

Measuring the mining time metrics of our tools is essential because we need to confirm that our method implementation is feasible in time. For instance, brute-force mining of undocumented instructions, although theoretically feasible, cannot be applied to practical scenarios due to the excessive time cost. Hence, time is an important metric for the algorithm.

Figure 6 shows the number of mining instructions changing with the running time of DSPUIM on three experimental DSPs: TMS320C6748@456MHz, TMS320DM6437@ 594MHz, and TMS320C6678@1000MHz. As can be seen in Figure 6, the slopes of the three curves varied little during the search. The reason was that the main frequency of the DSP was fixed, and the number of instructions tested per unit of time was also fixed. Therefore, we can conclude that the tool mining time T_m for the same model of DSP chip was positively related to the number of search instructions C_I, as shown in Formula (4). E represents the error parameter, and α is a proportional coefficient that is affected by parameters such as the test code and the main frequency of the DSP processor. Furthermore, by conducting more than five undocumented instruction mining experiments on each of the 14 DSP chips, it was found that the DSPUIM tool will mine the undocumented instructions of various DSP processor models in less than five minutes.

$$T_m = \alpha C_I + E \tag{4}$$

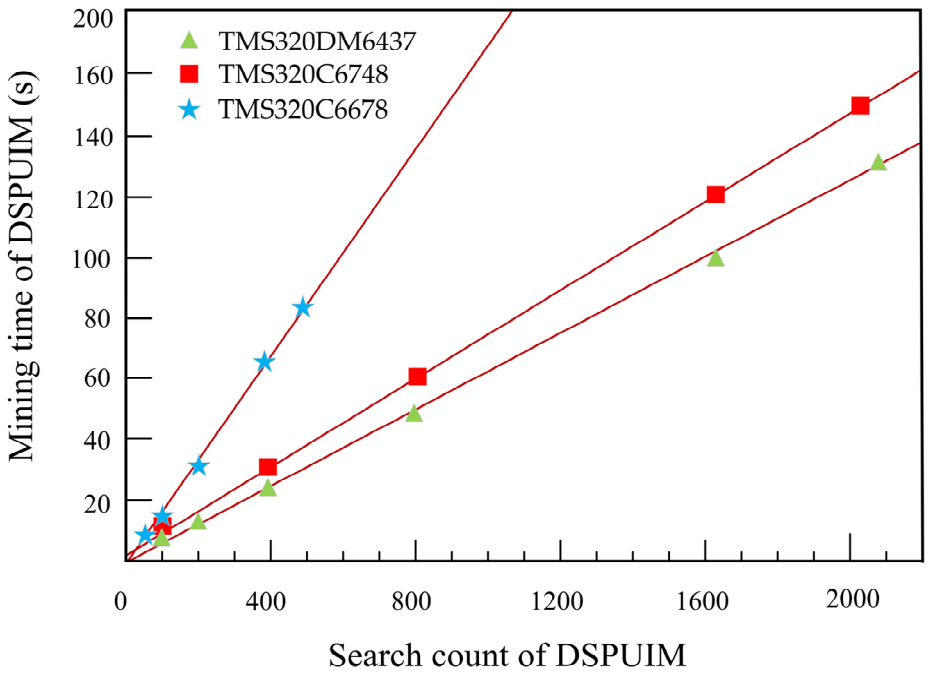


Figure 6. Count of instructions to be tested and mining time of DSPUIM.

4.4.2. Instruction Space Compression Analysis

From Section 4.4.1, it is clear that the total undocumented instruction mining time shrunk as the instruction search space decreased. Table 4 shows the original instruction search space of the four DSPs and the instruction space after the three steps of our method. According to Table 4, we can conclude that the instruction search space was reduced from 10^9 to 10^3 after the first step of our method. Moreover, after the disassembly method, the number of instructions to be tested was further reduced by 50%. This meant that the instruction generation method based on valid instruction information bits compressed the instruction space by a factor of 10^6, which was also the key to the fast search speed.

Table 4. Statistics on the number of instructions at each step.

DSP Processor Model	Original Instruction Search Space	Instruction Search Space after the Instruction Generation Method	The Undefined Instruction Space	The Undocumented Instruction Space
TMS320C6748	4294967296	4272	2054	21
TMS320C6416	4294967296	4240	2250	15
TMS320DM6437	4294967296	4272	2143	19
TMS320C6678	4294967296	1784	513	20

4.4.3. Hardware Overhead

To quantitatively evaluate the hardware overhead of DSPUIM, we observed the hardware memory occupied by the target DSP processor through the CCS software when executing the instruction to be tested. Through the test, it was found that the maximum hardware overhead occupied by the DSPUIM tool on the DSP processor was within 50 KB, and the occupied hardware size had nothing to do with the number of instructions to be tested. Taking the TMS320C6678 DSP processor as an example, the DSPUIM tool occupied 1.07% of SRAM memory in DSP. The space complexity of the DSPUIM tool was O(1). In this method, an unused memory space is opened, and each time the instruction to be tested is executed, it will jump to the initial address of the memory, so the memory space will be used repeatedly. Therefore, the DSPUIM tool greatly saves hardware overhead.

4.5. Discussion

4.5.1. Comparison with the Existing Methods

We compared our tool DSPUIM with existing undocumented instruction mining tools in terms of the applicable processor types of the tool. As shown in Table 5, existing undocumented instruction mining tools are only applicable to general-purpose processors such as x86, ARM, and RISC-V, while only our tool applies to DSP processors.

Table 5. Undocumented instruction mining tools for processor types.

Tool \ Processor Type	DSP	ARM	RISC-V	x86
Sandsifter [24]				√
CPU security benchmark [25]				√
UISfuzz [26]				√
iScanU [47]		√	√	
Armshaker [48]		√		
HEAM [27]				√
Our DSPUIM	√			

4.5.2. Limitation

Although we have demonstrated our tool DSPUIM can successfully find out undocumented instructions on 14 DSPs in an efficient way, there are several limitations of our tool.

First, our tool needs to obtain feedback on instruction execution from the DSP processor when distinguishing undocumented instructions from illegal instructions. Second, establishing a physical connection to the target DSP processor is necessary for the validity of our tool. The DSPUIM tool is invalid without a physical connection to the test interface of the target DSP. Third, our undocumented instruction search method is only applicable to the DSP instruction set and cannot be used for variable-length instruction sets such as the x86 instruction set. The reason for this is that we design the instruction generation method based on the analysis of the structural characteristics of the DSP instruction set. Fourth, our tool can only record the format and quantity of undocumented instructions for the target DSP chip. In addition, the execution effect of undocumented instructions can only be analyzed by running each undocumented instruction on the processor in debug mode.

5. Conclusions

In this paper, we designed and implemented an efficient undocumented instruction search method for DSP processors that can successfully mine undocumented instructions on a variety of DSP chips. We briefly introduced related work on undocumented instruction search, detailed DSP undocumented instruction search methods, and proposed an instruction search tool named DSPUIM. Experiments were conducted on 14 DSP processors to evaluate the effectiveness of our tool, and the experimental results showed that DSPUIM successfully mined 335 undocumented instructions. We analyzed the functions and characteristics of undocumented instructions. We found that the undocumented instructions in function-changing and denial-of-service classes were potential threats to the security of the DSPs. Furthermore, by quantitatively analyzing the experimental process, our undocumented instruction mining processes were completed in less than 5 min with less than 50 KB of memory, which means that the DSPUIM tool has a low time and hardware overhead.

After analyzing the functions of these undocumented instructions, the way in which to build defense measures against these undocumented instructions will be the focus of our future work. It is hoped that this article will generate more interest from researchers in DSP undocumented instructions and contribute more strongly to promoting DSP security.

Author Contributions: Methodology, X.Z.; validation, X.Z. and J.Y.; writing—original draft preparation, X.Z.; writing—review and editing, J.W., Z.C., H.L., C.L. and B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key Research and Development Program of Sichuan Province, grant numbers 2021YFG0008 and 2021YFG0382.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DSP	Digital signal processor
DSPUIM	Undocumented instruction mining tool for digital signal processor
TI	Texas Instruments
IoT	Internet of Things
USD	United States dollar
CISC	Complex instruction set computer
RISC	Reduced instruction set computer

CPU	Central processing unit
RICS-V	Reduced instruction set computer five
ISA	Instruction set architecture
MIPS	Microprocessor without interlocked pipeline stages
CCS	Code composer studio
IDA	Interactive disassembler
EDA	Electronic design automation

References

1. Lu, Y.; He, Y.; Gong, J.; Xu, G. Design and implementation of ATM interface module based on high speed DSP. In Proceedings of the 2022 IEEE 5th International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 23–25 September 2022; pp. 230–234.
2. Abdelkareem, A.E. Hardware considerations of a DSP based wireless coded receiver under limited resources. In Proceedings of the 2022 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE), Hadhramaut, Yemen, 3–5 December 2022; pp. 1–5.
3. Eyre, J.; Bier, J. The evolution of DSP processors. *IEEE Signal Process. Mag.* **2000**, *17*, 43–51. [CrossRef]
4. Eyre, J. The digital signal processor Derby. *IEEE Spectr.* **2001**, *38*, 62–68. [CrossRef]
5. Yu, J.; Wei, Y. Digital Signal Processing for High-Speed THz Communications. *Chin. J. Electron.* **2022**, *31*, 534–546. [CrossRef]
6. Fu, Y.; Chen, K.; Song, W.; He, G.; Shen, S.; Wang, H.; Zhang, C.; Li, L. A DSP-Purposed REconfigurable Acceleration Machine (DREAM) for High Energy Efficiency MIMO Signal Processing. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 952–965. [CrossRef]
7. Song, Y.; Feng, W.; Quan, Y.; Liu, Y.; Li, Q.; Dauphin, G.; Wang, Y.; Xing, M. Remote Sensing Image Fusion Technology Based on DSP. In Proceedings of the IGARSS 2022—2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur, Malaysia, 17–22 July 2022; pp. 3359–3362.
8. Vaz, D.J.F.; Joao, F.A.V.; Serralheiro, A.J.; Gerald, J.A.B. Autopilot—An autonomous navigation system. In Proceedings of the 2016 Conference on Design of Circuits and Integrated Systems (DCIS), Granada, Spain, 23–25 November 2016; pp. 1–5.
9. Zhang, Q.; Xie, Q.; Duan, K.; Liang, B.; Wang, M.; Wang, G. A digital signal processor (DSP)-based system for embedded continuous-time cuffless blood pressure monitoring using single-channel PPG signal. *Sci. China Inf. Sci.* **2020**, *63*, 149402:1–149402:3. [CrossRef]
10. Swamy, K.A.; Alex, Z.C.; Ramachandran, P.; Mathew, T.L.; Sushma, C.; Padmaja, N. Real-time Implementation of Delay Efficient DCT Based Hearing Aid Algorithm Using TMS320C5505 DSP Processor. In Proceedings of the 2021 Innovations in Power and Advanced Computing Technologies (i-PACT), Kuala Lumpur, Malaysia, 27–29 November 2021; pp. 1–8.
11. López, J.M.; Alonso, J.; Asensio, C.; Pavón, I.; Gascó, L.; de Arcas, G. A Digital Signal Processor Based Acoustic Sensor for Outdoor Noise Monitoring in Smart Cities. *Sensors* **2020**, *20*, 605. [CrossRef]
12. Fawzy, D.; Moussa, S.M.; Badr, N.L. The Internet of Things and Architectures of Big Data Analytics: Challenges of Intersection at Different Domains. *IEEE Access* **2022**, *10*, 4969–4992. [CrossRef]
13. Global DSP Chips Market Research Report 2022. Available online: https://www.marketgrowthreports.com/global-dsp-chips-market-21064939 (accessed on 15 November 2022).
14. Wang, Y.; Liu, P.; Jiang, Y. IMSC: Instruction set architecture monitor and secure cache for protecting processor systems from undocumented instructions. *IET Inf. Secur.* **2022**, *16*, 314–319. [CrossRef]
15. Duflot, L. CPU bugs, CPU backdoors and consequences on security. *J. Comput. Virol.* **2009**, *5*, 91–104. [CrossRef]
16. Price, D. Pentium FDIV flaw-lessons learned. *IEEE Micro* **1995**, *15*, 86–88. [CrossRef]
17. Pentium Processor Specification Update, Invalid Operation with Locked CMPXCHG8B Instruction. Available online: http://www.cpuzone.com/Pentium/Pentium%20processor%20specifcation.pdf (accessed on 15 November 2022).
18. Domas, C. Hardware backdoors in x86 CPUs. In Proceedings of the Black Hat, Las Vegas, NV, USA; 2018; pp. 1–14.
19. Sengupta, A.; Chaurasia, R.; Reddy, T. Contact-Less Palmprint Biometric for Securing DSP Coprocessors Used in CE Systems. *IEEE Trans. Consum. Electron.* **2021**, *67*, 202–213. [CrossRef]
20. Chaurasia, R.; Sengupta, A. Security Vs Design Cost of Signature Driven Security Methodologies for Reusable Hardware IP Core. In Proceedings of the 2022 IEEE International Symposium on Smart Electronic Systems (iSES), Warangal, India, 18–22 December 2022; pp. 283–288.
21. Shanley, T. *x86 Instruction Set Architecture*; MindShare Press: Austin, TX, USA, 2010.
22. Collins, R. Undocumented Opcodes: Salc. 1999. Available online: http://www.rcollins.org/secrets/opcodes/SALC.html (accessed on 14 March 2023).
23. Easdon, C. Undocumented CPU behavior: Analyzing undocumented opcodes on Intel x86-64. *Talk* **2018**.
24. Domas, C. Breaking the x86 ISA. In Proceedings of the Black Hat, Las Vegas, NV, USA, 22–27 July 2017.
25. Zhu, J.; Song, W.; Zhu, Z.; Ying, J.; Li, B.; Tu, B.; Shi, G.; Hou, R.; Meng, D. CPU Security Benchmark. In Proceedings of the Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors, Toronto, ON, Canada, 15 October 2018; pp. 8–14.

26. Li, X.; Wu, Z.; Wei, Q.; Wu, H. UISFuzz: An Efficient Fuzzing Method for CPU Undocumented Instruction Searching. *IEEE Access* **2019**, *7*, 149224–149236. [CrossRef]
27. Wu, J.; Cui, B.; Chen, C.; Long, X. A High Efficiency and Accuracy Method for x86 Undocumented Instruction Detection and Classification. In Proceedings of the Innovative Mobile and Internet Services in Ubiquitous Computing, Asan, Republic of Korea, 1–3 July 2021; Springer: Cham, Switzerland, 2022; pp. 295–303.
28. Ermolov, M.; Sklyarov, D.; Goryachy, M. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors. *J. Comput. Virol. Hacking Tech.* **2022**, *29*, 27–41.
29. Wang, G.; Zhu, Z.; Li, S.; Cheng, X.; Meng, D. Differential Testing of x86 Instruction Decoders with Instruction Operand Inferring Algorithm. In Proceedings of the 2021 IEEE 39th International Conference on Computer Design (ICCD), Storrs, CT, USA, 24–27 October 2021; pp. 196–203.
30. Domas, C. The memory sinkhole. In Proceedings of the Black Hat, Las Vegas, NV, USA, 1–6 August 2015.
31. Lipp, M.; Schwarz, M.; Gruss, D.; Prescher, T.; Haas, W.; Mangard, S.; Kocher, P.; Genkin, D.; Yarom, Y.; Hamburg, M. Meltdown. *arXiv* **2018**, arXiv:1801.01207. [CrossRef]
32. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T. Spectre attacks: Exploiting speculative execution. *Commun. ACM* **2020**, *63*, 93–101. [CrossRef]
33. Gras, B.; Razavi, K.; Bos, H.; Giuffrida, C. Translation leak-aside buffer: Defeating cache side-channel protections with {TLB} attacks. In Proceedings of the USENIX Security Symposium, Baltimore, MD, USA, 15–17 August 2018; pp. 955–972.
34. Aldaya, A.C.; Brumley, B.B.; Hassan, S.u.; García, C.P.; Tuveri, N. Port Contention for Fun and Profit. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 870–887.
35. Schaik, S.v.; Milburn, A.; Österlund, S.; Frigo, P.; Maisuradze, G.; Razavi, K.; Bos, H.; Giuffrida, C. RIDL: Rogue In-Flight Data Load. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 88–105.
36. Löw, M. Overview of meltdown and spectre patches and their impacts. *Adv. Microkernel Oper. Syst.* **2018**, 53–61.
37. Stuart, B. Current state of mitigations for spectre within operating systems. *Adv. Microkernel Oper. Syst.* **2018**, 47–51.
38. Müller, L. Kpti a mitigation method against meltdown. *Adv. Microkernel Oper. Syst.* **2018**, 41–46.
39. Hicks, M.; Sturton, C.; King, S.T.; Smith, J.M. SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, Istanbul, Turkey, 14 March 2015; pp. 517–529.
40. Chen, D.D.; Ahn, G.-J. Security analysis of x86 processor microcode. *Ariz. State Univ. Tempe AZ USA Tech. Rep.* **2014**, 1–18.
41. Koppe, P.; Kollenda, B.; Fyrbiak, M.; Kison, C.; Gawlik, R.; Paar, C.; Holz, T. Reverse engineering x86 processor microcode. In Proceedings of the USENIX Security Symposium, Vancouver, BC, Canada, 16–18 August 2017; pp. 1163–1180.
42. Taheri, F.; Bayat-Sarmadi, S.; Hadayeghparast, S. RISC-HD: Lightweight RISC-V Processor for Efficient Hyperdimensional Computing Inference. *IEEE Internet Things J.* **2022**, *9*, 24030–24037. [CrossRef]
43. Meng, Z.; Zhang, Y.; Zhou, J.; Guo, Z. Design of 64-Bit High-Performance Embedded Processor Supporting RISC-V B-Extension. In Proceedings of the 2022 IEEE 16th International Conference on Anti-counterfeiting, Security, and Identification (ASID), Xiamen, China, 2–4 December 2022; pp. 28–32.
44. Patterson, D.A.; Sequin, C.H. RISC I: A reduced instruction set VLSI computer. In Proceedings of the 25 Years of the International Symposia on Computer Architecture (Selected Papers), Minneapolis, MN, USA, 12–14 May 1981; pp. 216–230.
45. Heath, S. *Microprocessor Architectures: RISC, CISC and DSP*, 2nd ed.; Butterworth-Heinemann Ltd.: Oxford, UK, 1995.
46. Reid, A. Trustworthy specifications of ARM® v8-A and v8-M system level architecture. In Proceedings of the 2016 Formal Methods in Computer-Aided Design (FMCAD), Mountain View, CA, USA, 3–6 October 2016; pp. 161–168.
47. Dofferhoff, R.; Göebel, M.; Rietveld, K.; Kouwe, E.V.D. iScanU: A Portable Scanner for Undocumented Instructions on RISC Processors. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 29 June–2 July 2020; pp. 306–317.
48. Strupe, F.; Kumar, R. Uncovering hidden instructions in Armv8-A implementations. In *Hardware and Architectural Support for Security and Privacy (HASP '20)*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1–9.
49. Lu, T. A survey on risc-v security: Hardware and architecture. *arXiv* **2021**, arXiv:2107.04175.
50. Cassano, L.; Mascio, S.D.; Palumbo, A.; Menicucci, A.; Furano, G.; Bianchi, G.; Ottavi, M. Is RISC-V ready for Space? A Security Perspective. In Proceedings of the 2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Austin, TX, USA, 19–21 October 2022; pp. 1–6.
51. Ahmadi, M.M.; Khalid, F.; Shafique, M. Side-channel attacks on RISC-V processors: Current progress, challenges, and opportunities. *arXiv* **2021**, arXiv:2106.08877.
52. Wistoff, N.; Schneider, M.; Gürkaynak, F.K.; Benini, L.; Heiser, G. Prevention of microarchitectural covert channels on an open-source 64-bit RISC-V core. *arXiv* **2020**, arXiv:2005.02193.
53. Escouteloup, M.; Lashermes, R.; Lanet, J.-L.; Fournier, J.J.-A. Recommendations for a radically secure ISA. In Proceedings of the CARRV 2020—Workshop on Computer Architecture Research with RISC-V, Valence (virtual), Valencia, Spain, 30 May 2020; pp. 1–22.
54. Texas. OMAP35x applications processor technical reference manual. In *User Manual SPRUF98Y*; Bharathidasan Engineering College: Vellore, India, April 2010.

55. Texas. TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide. In *User Manual SPRU732J*; Texas Instruments: Dallas, TX, USA, July 2010.
56. Texas. TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide. In *User Manual SPRU733A*; Texas Instruments: Dallas, TX, USA, November 2006.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.