Flexible Processor Architecture Design

Dvivedi Rohan Vipulkumar
Mechanical Engineering
BITS-Pilani, Hyderabad Campus
Hyderabad, Telangana, India 500078
rohandvivedi@gmail.com

P. Veda Bhanu*, and Soumya J†
Electrical and Electronics Engineering
BITS-Pilani, Hyderabad Campus
Hyderabad, Telangana, India 500078
{vedabhanuiit2010*, soumyatkgp†}@gmail.com

***Abstract*—This paper provides an alternative approach to the modern processor architecture design using a Transport Triggered Architecture (TTA). The aim has been focused on flexible computer architecture design, that can allow execution of multiple Instruction Sets (Hardware) or Byte Codes (Virtual Machines) to emulate other Instruction Set Architectures (ISAs). The idea is to design a vertical microcoded processor, that can enable basic register to register data transfer from one address to another using TTA system. This results in execution of complex instructions in a multi-cycle fashion. The target then boils down to convert any currently known instructions to just a mere multiple register to register word transfers that would in-turn invoke other combinational or sequential hardware modules to execute the instruction.**

Keywords*—*Processor, Computer architecture design, Transport Triggered Architecture, Multi-cycle processor

I. Introduction

Instruction Set (IS) may have large number of architectural implementations. In the commercial market major IS which have own software infrastructure are x86, ARM, MIPS, and soft core instruction set architectures namely RISC-V, Virtual Machine (VM) based V8 engine, Python Virtual Machine (PVM) in CPYTHON etc. Every system programmer requires the most stable and built up software infrastructure to program the machine. Hence, there is a need for additional instructions to the machine which could make the task easier. For example, adding a more compatible atomic instruction to move large chunks of data more efficiently, to combine two repetitively occurring instructions on a RISC machine (example Load + Add), multiply add in FMA3 FMA4 by AMD in pile driver [1] and Intel in HASWELL [2]. With the proposed architecture design paradigm newer instructions may not require redesigning of the control logic, or adding more control signal lines. The Programmable Instruction Set Architecture (PISA) will allow IS emulation, extension of IS may lead to more flexibility and efficiency as they get discovered at system level [3]. PISA provides a solution to solve software compatibility issues at hardware level by dynamically loading the IS as and when needed, and also avoids cyber invasions by customizing the IS. A PISA would also enable emulation of VMs at hardware level. The proposed model can bring hardware level security on the processors, where availability of the computer program is not necessarily enough to execute the instructions. Customization of IS for server system companies would provide better efficiency by implementation of interpreted programming languages like JavaScript at hardware level. Moreover, newer embedded processor can be designed which have more complex and only useful instructions in their microcode. This results in decreasing unnecessary fetches of instructions. The proposed architecture design is based on Transport Triggered Architecture (TTA) [4], where multiple transfers of words of memory are just enough to execute any operation, by invoking other logic circuits [5].

A flexible openISA has been proposed in [6], that enables both software and hardware to evolve independently. Their work concentrated on efficient software emulation to overcome hardware lock-ins. TTAs possess many advantages, such as modularity, flexibility, and scalability. Low power and high code density architectures have been proposed in [5]. They have optimized the ISA, architecture, circuit and compiler level design to reduce the power consumption. An efficient instruction scheduling method for configurable processor based on TTA has been proposed in [7]. They have used the genetic algorithm to realize the instruction scheduling and tabu search is implemented to prevent from local optimal solution. In [8], a universal processor based on fully parallel fixed complexity sphere decoder algorithm has been proposed for TTA. They have used different functional units to accommodate antenna configurations. Unlike the approaches reported in the literature, our proposed architecture design can emulate any instruction set within single stage pipelining. The rest of the paper is organized as follows. Section II briefs about proposed architectural design. Section III recites the experimentation results followed by the conclusion and future scope.

II. Architectural design

The design consists of an execution unit that controls related internal peripherals. The prototype consists of only one execution unit but multiple peripherals which operates on a TTA [4]. Different types of peripherals can be specially designed for such an architecture. The proposed model has the internal peripherals with their addressing shown in Table I to Table III.

Table I shows the address range, size and name of the registers used for internal peripherals which are both readable and writable. Table II shows address range, size and name of the registers used for internal peripherals which are writable only. Similarly, Table III shows the address, and operations of the instructions. For the prototype machine, register A has the same address as GPR 0x0D, register B, C has an address of 0x0E and 0x0F. The operations on these registers can be a combinational logic of any of these registers. For the prototype, 0x0D, 0x0E and 0x0F registers are chosen randomly. The

TABLE I: Internal Peripheral Addressing

Address range	Size (bits)	Name of the register
0x00 - 0x0F	16	General Purpose Registers (GPR)
0x10 - 0x13	04	Input Output Registers (IOR)
0x14 - 0x14	01	Input Output Control Register (IOC)
0x15 - 0x16	02	General Purpose Registers (GPR)
0x17 - 0x17	01	Immediate Register (IMM)
0x18 - 0x19	02	Counter1 (A01:A11) for Memory bank1 (M1)
0x1A - 0x1A	01	register in Memory bank 1 (M1) addressed by Counter1
0x1B - 0x1C	02	Counter2 (A02:A12) for Memory bank2 (M2)
0x1D - 0x1D	01	register in Memory bank 2 (M2) addressed by Counter2
0x1E - 0x1E	01	Signal register for C1 and C2 counters
0x1F - 0x1F	01	NULL

TABLE II: Internal peripheral addressing for write-only registers

Address range	Size (bits)	Name of the register
0x20 - 0x22	03	Microcode Jump Address registers UCJA (J2:J1:J0) J0 is addressed 0x22, J1 as 0x21 and J2 as 0x20
0x23 - 0x23	01	Write Address Register (WAR)
0x24 - 0x24	01	Read Address Register (RAR)
0x25 - 0x25	01	Jump Register

TABLE III: Internal peripheral addressing for read-only registers

Address	Operation of instructions
0x20	Carry out from addition A + B + (c & 0x01)
0x21	A lesser than B
0x22	A equals B
0x23	A greater than B
0x24	A + B + (C & 0x01)
0x25	A & B (bitwise and)
0x26	A | B (bitwise or)
0x27	A $\oplus$ B (bitwise ex-or)

addressing of internal peripherals in the proposed model has been randomly selected. However, user can give any specific address range. The suitable GPR addressing for the the prototype has been decided keeping in mind the register addresses starting from 0 in ARM architectures.

A. Execution Unit

As in any TTA architecture [4], a word is read from source address and written to destination address, where read and write addresses are generated by the execution unit, according to the microcode. The microcode has to be programmed with a fetch, decode, and execute loop for execution of an instruction. The microcode instruction format is 16 bits wide, 15^{th} bit represents operate Write Address Register (opWAR) and 14^{th} bit represents operate Read Address Register (opRAR), the 6 bits from 13^{th} bit to 8^{th} bit represent the Write Address and 5^{th} bit to 0^{th} bit represent the Read Address. The prototype design has only 8 bit internal data bus, all the registers used in our prototype are also 8 bit wide. If the opWAR bit is set, in the microcode, the WAR is used as the Write Address field from the microcode, else it is taken from the microcode instruction field. Similarly, if the opRAR bit is set in the microcode, then RAR is used as the Read Address field from the microcode or else it is taken from the microcode instruction field.

Fig. 1 shows the block diagram of execution unit, functionality of the execution unit provides a way/ a walk into systematic manipulation of the microcode execution. This enables to take operands for the execution instruction from the decoded counterparts. A generic instruction would have format like, opcode operand 1, operand 2 considering a standard register to register move instruction. The operands would be instruction level addresses, which would be translated and loaded to RAR and WAR register to execute the instruction. In this way, an instruction would itself be accounted as a function for the microcode.

A conditional microcode JUMP can be executed by writing a value to 0x25 address. If the value has bit 0 set, a microcode jump is executed and the next instruction corresponds to the UCJA (microcode counter jump address) register. It is a combination of J2, J1, J0 registers and from the next cycle microcode address increments as usual. It has to be noted that the JUMP addresses have to be pre-loaded using the microcode before writing to the 0x25 address.

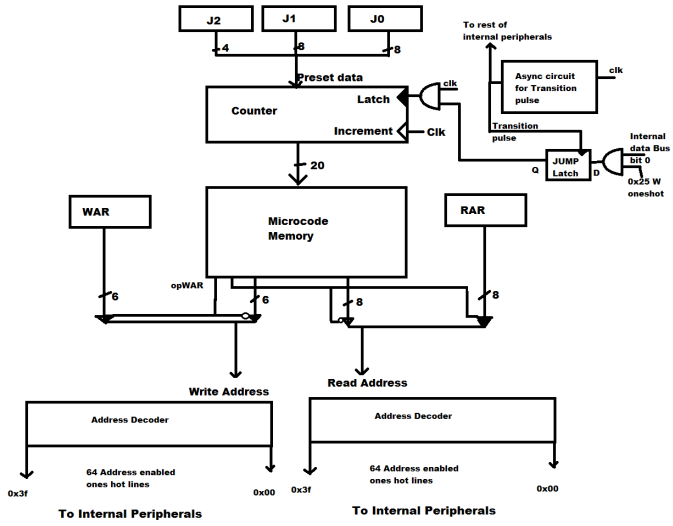


Fig. 1: Block diagram of execution Unit

B. Internal Peripherals

• **GPR - General Purpose Register**: A GPR can be written or read as a generic register. Fig. 2 shows the functional schematic of GPR. For an ISA, it has to be appointed as a generic operand storage register or a special purpose register like Program counter, Stack pointer, Queue pointer, Instruction register and other pointers.

• **IOR - Input Output Register**: IORs are the registers that provide input and output functionality to the processor for any instruction set that has to be ported on the hardware. IOR can be set as Input or Output using the specific bits in IOC register. If IOR register is set as an output, it can be both read and written with a new value, on the contrary if IOR is set as

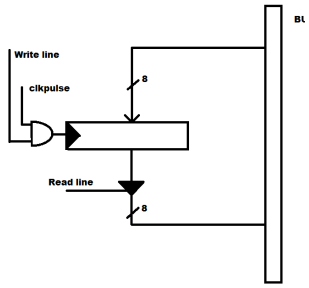


Fig. 2: Functional schematic of GPR

an input, it can be read only. If written they are not visible on the port unless the port gets set to output using the IOC bits. The Input/Output ports can be programmed to set as data bus, multiplex address and data bus, for addressing more memory, and get higher data width. Functional schematic of IOC and IOR is shown in Fig. 3.

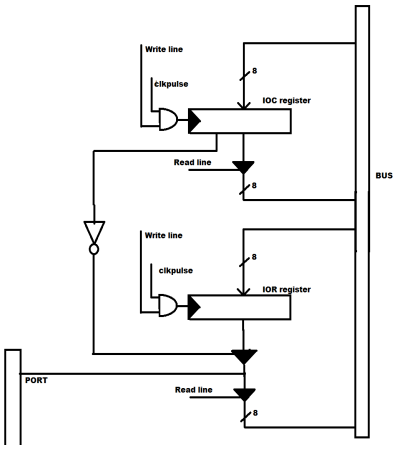


Fig. 3: Functional schematic of IOR and IOC registers

• **IOC - Input Output Control Register**: IOC register can be used to control the Input/Output of the IOR registers, to register interrupt and output Read/Write signal. To make the functionality more clear, consider the prototype having 4 IORs namely IOR0, IOR1, IOR2, IOR3 addressed as 0x10, 0x11, 0x12, 0x13 respectively.

• **IMM - Immediate Register**: This register is provided to generate Immediate values for the instruction execution. On writing a value to the IMM it does not get overwritten with the value of internal data bus, instead it gets written with value of the Read Address. For instance, the prototype addresses IMM as 0x17, so to generate immediate value 0x53, the microcode Read Address = 0x53 and Write Address = 0x17. Now this value can be written to any other register that needs to be initialized to 0x53. The read operation of the IMM is similar to any other generic GPR register.

• **C1 or C2 - Counter Register**: This registers can be read or written similar to any-other generic GPR registers. But the registers can be incremented, decremented or cleared by writing appropriate signals to the signal register. For instance,

TABLE IV: Bitwise Description of IOC register

Bit	Name	Functionality
7	Int1	it gets set using external signal this bit is cleared after IOC is read
6	Int0	it gets set using external signal this bit is cleared after IOC is read
5	Write bit	to signal write operation to external peripherals
4	Read bit	to signal read operation to external peripherals
3	IOC3	if set the IOR3 port is Input else it is Output
2	IOC2	if set the IOR2 port is Input else it is Output
1	IOC1	if set the IOR1 port is Input else it is Output
0	IOC0	if set the IOR0 port is Input else it is Output

in the prototype design C1, instead of a single register a 13 bit register composed of two registers addressed at 0x18 and 0x19 is used to address a memory location in the memory bank.

• M**1 or M2 - Memory Bank Register**: The address location corresponds to the address pointed out by the contents of the counter register C1 or C2 respectively. Such a configuration can be used to access memory banks sequentially. The specific task that can be accomplished by such a memory organization, would be to queue instructions or to stack microcode function calls.

• **Signal Register**: The signal register can be written with appropriate signal values. Once the signal register has been written with appropriate values in a particular cycle, it gets written with default values at the rising edge of the next clock. For instance, the bit functionality of the signal register in the prototype design is shown in Table. V.

TABLE V: Bit wise description of signal register

Bit	Name	Activated by	Default value
5	C2 clr	if set value gets cleared	0
4	C2 dec	rising edge	1
3	C2 inc	rising edge	1
2	C1 clr	if set value gets cleared	0
1	C1 dec	rising edge	1
0	C1 inc	rising edge	1

• **UCJA - Microcode Jump Address Register**: It is a concatenation of J2, J1, and J0 registers. This is the address to which microcode jumps, when a Microcode Jump Address is written with an odd value (with the 0th bit value set).

• **JUMP register**: If the JUMP register is written with an odd value, The UCJA value is used as the address for fetching of the next microcode instruction.

• **RAR, WAR - Read Address Register, Write Address Register**: The following registers RAR and WAR are more dominantly a part of execution unit. The contents of RAR registers can substitute Read Address, when opRAR bit of microcode instruction is set. Similarly, the contents of WAR registers can substitute Write Address, when opWAR bit of microcode instruction is set, for executing the micro instruction.

• **ALU operation registers**: The ALU operation register get latched during the high level of the clock pulse, the data transport from one register to another on a transition pulse only during the low edge of the pulse. Hence, due to such a configuration, the data can be easily read from ALU operations and written to ALU operand registers. Even though the memory element used for ALU operands are high level transparent D-latches and not edge triggered flip-flops. Such a configuration of latching the operation registers (latches) save a data transition, for data transport from operation register to any one of the operand registers. Fig. 4 shows the ALU functional schematic diagram.

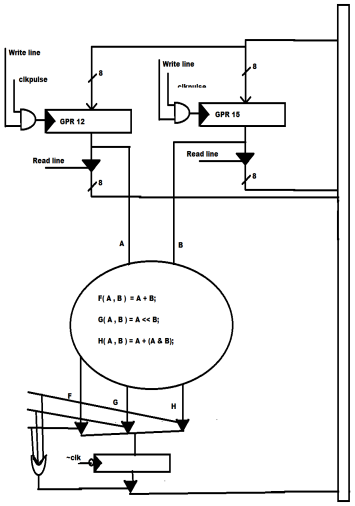


Fig. 4: Arithmetic and Logic functional schematic diagram

C. Transition pulse

Fig. 5 shows the clock and transition pulse used in our experimentation. The clock for prototype system requires specific high time and specific low time. The transition pulse has to be occurred only during the low level of the clock. The clock to prototype a design is used to generate the transition pulse, by using an asynchronous logic circuit that gives a pulse of just the right width. The following terminology is used in calculating specific high and low time of the clock pulse.

$$t_{\text{high}} = High\ Pulse\ of\ Clock$$

$$t_{\text{low}} = Low\ Pulse\ of\ Clock$$

$$t_{\text{C}} = Propogation\ delay\ for\ Counter$$

$$t_{\text{MM}} = Propogation\ delay\ of\ Microcode\ Memory$$

$$t_{\text{D}} = Propogation\ delay\ of\ Decoder$$

$$t_{\text{SU}} = Setup\ time\ for\ Memory\ Element$$

$$t_{\text{MP}} = Minimum\ pulse\ for\ latching\ data$$

$$t_{\text{HD}} = Data\ Hold\ time\ for\ Memory\ Element$$

$$t_{\text{high}} = t_{\text{C}} + t_{\text{MM}} + t_{\text{D}}$$

$$t_{\text{low}} = t_{\text{SU}} + t_{\text{MP}} + t_{\text{HD}}$$

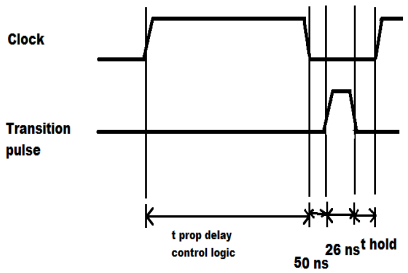


Fig. 5: Clock and Transition pulse signals

III. Experimental Results

The Prototype has been designed from commercial High Speed CMOS logic ICs. The ICs incorporated into the prototype together are, counters (74HC193) [9], transparent octal D-latches (74HC573) [10], rising edge triggered flip-flops (74HC374) [11], decoders (74HC138, 74HC238 and 74HC154) [12], [13], [14], AND gates (74HC08) [15], OR gates (74HC32) [16], EX-OR gates (74HC86) [17], Not gates (74HC04) [18], 4 to 1 Mux (74HC153) [18], shift registers (74HC595) [19]. All of the components put together for prototyping the designed architecture.

A. Experiment setup

The experiment has been setup to check for the complete functional working of the architecture. Fig. 6 shows the experiment setup used in our prototype. Fig. 6(a) shows peripheral interfacing, Fig. 6(b) shows top view of processor architecture, Fig. 6(c) shows processor bus back plane where different boards are stacked onto one single board and Fig. 6(d) shows about IO and Memory modules connected to processor. The I/O has been provided to ensure responsiveness and visual functionality of the hardware. The input was taken from ps2 Keyboard as shown in Fig. 6(a), and output was printed to a 16 x 2 LCD Display as shown in Fig. 7.

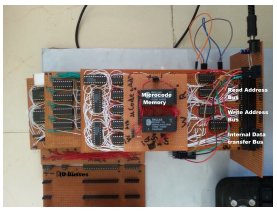
A simple program has been devised to ensure the complete functionality check on the designed prototype. IOR0 has been used as Data bus and IOR1 to IOR3(first 4 bits only) has been used for addressing memory and I/O. Table VI shows the memory addressing scheme carried out for the experimentation.

TABLE VI: Memory addressing scheme

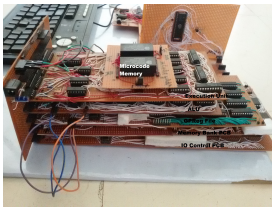
Address range	Peripheral
0x00000 - 0x37fff	RAM
0x38000	LCD data
0x38001	LCD command
0x38002	keyboard / SPI data count
0x38003	SPI read / write data
0x80000 - 0xfffff	ROM

1) Functionality of experimental microcode: It takes the input from keyboard when an interrupt occurs from the keyboard or scancode, and checks if it was a key press or key released. By checking the prefix of scancode (0xF0), if it was a key release it decodes the scancode to ASCII lower case characters. If the key released was enter key, then it will print

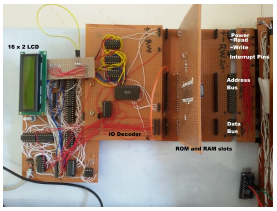
(a) Peripheral interfacing



(b) Top view of prototype



(c) Processor bus back plane



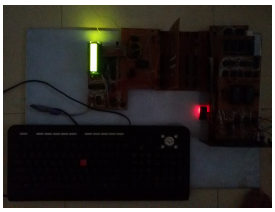
(d) IO and Memory modules

Fig. 6: Experimental Setup

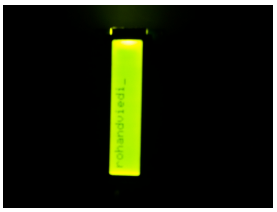
all the characters collected up-till now or else it will queue the new character and wait for next interrupt. The ps2 Keyboard to scancode table is preloaded in the ROM at address 0x801XX containing ASCII code 0xYY. Here, XX is the scancode from keyboard and YY is the corresponding ASCII value that we read from the ROM.

B. Results

Fig. 7(a) and Fig. 7(b) shows the working snapshot of our prototype where the input characters are displayed on 16x2 LCD. Using the experimental microcode, the prototype has been able to work appropriately at 120kHz to 280kHz with 75% to 98% duty-cycle clock. It consumes 0.37 Ampere at 5.03 volts, effectively drawing 1.86 W of power. The power consumption includes the power drawn by the 16 x 2 LCD display and Keyboard. The results show that functionality of our proposed design is working successfully.



(a) Displaying the input characters



(b) Input entries are displayed

Fig. 7: Experimental results

IV. Conclusion

The prototype designed so far has been successfully working and as an extension to this work newer block modules are added which would allow compact microcode programs. On the basis of the work carried out till date, no commercial IS has been ported to its microcode. Our future work includes porting of multiple ISAs to the existing prototype, and to keep on adding more blocks of combinational/sequential circuits to the design as the necessity arises.

Acknowledgment

The authors sincerely thank to Prof. Jeevan Jaidi, Department of Mechanical Engineering, BITS-Pilani, Hyderabad Campus for his constant support and guidance received towards the research work carried out in this paper.

References

[1] S. Nussbaum, "Hot chips 2012 amd ;trinity ; apu," in *2012 IEEE Hot Chips 24 Symposium (HCS)*, Aug 2012, pp. 1–40.

[2] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, Mar 2014.

[3] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. V. der Perre, and F. Catthoor, "A unified instruction set programmable architecture for multi-standard advanced forward error correction," in *2008 IEEE Workshop on Signal Processing Systems*, Oct 2008, pp. 31–36.

[4] H. Corporaal, "Design of transport triggered architectures," in *Proceedings of 4th Great Lakes Symposium on VLSI*, March 1994, pp. 130–135.

[5] Y. He, D. She, B. Mesman, and H. Corporaal, "Move-pro: A low power and high code density tta architecture," in *2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, July 2011, pp. 294–301.

[6] R. Auler and E. Borin, "The case for flexible isas: Unleashing hardware and software," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct 2017, pp. 65–72.

[7] S. Zhong, J. Wei, W. Guo, and Z. Wang, "Instruction scheduling using genetic algorithm with taboo search for tta-like processors," in *2010 International Conference On Computer Design and Applications*, vol. 2, June 2010, pp. V2–413–V2–417.

[8] Z. Xu, Z. Bie, C. Chen, and X. Jiao, "Tta processor design for fpfsd-mimo detector," in *2013 15th IEEE International Conference on Communication Technology*, Nov 2013, pp. 620–624.

[9] *Presettable synchronous 4-bit binary up/down counter*, Nexperia B V, 01 2016, rev. 5.

[10] *3-state Octal D-type transparent latch*, Nexperia B V, 03 2016, rev. 7.

[11] *High-Speed CMOS Logic Octal D-Type Flip-Flop, 3-State Positive-Edge Triggered*, Texas Instruments, 02 1998, revised May 2004.

[12] *3 TO 8 LINE DECODER DEMULTIPLEXER*, Diodes Incorporated, 06 2013, rev.3 - 2.

[13] *3-to-8 line decoder/demultiplexer*, Nexperia B V, 06 2018, rev. 5.

[14] *4-to-16 line decoder/demultiplexer*, Nexperia B V, 02 2016, rev. 7.

[15] *16-bit even/odd parity generator/checker*, Phillips Semiconductors, 12 1990.

[16] *Quad 2-input OR gate*, Nexperia B V, 12 2015, rev 6.

[17] *Quad 2-input EXCLUSIVE-OR gate*, Nexperia B V, 12 2015, rev 4.

[18] *Hex inverter*, Nexperia B V, 12 2015, rev 5.

[19] *SNx4HC595 8-Bit Shift Registers With 3-State Output Registers*, Texas Instruments, 9 2015, revised from November 2009.