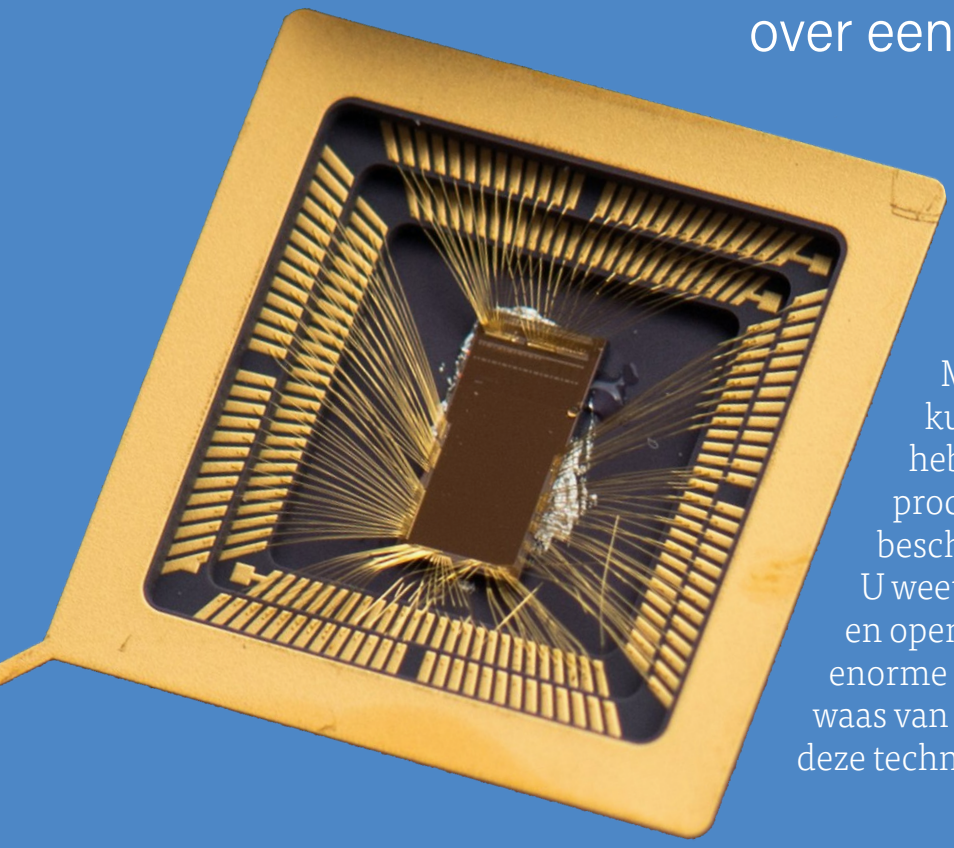
Wat Is **RISC-V**?

en waarom de industrie zo enthousiast is over een nieuwe processorkern!



Stuart Cording (Elektor)

De elektronica-industrie lijkt op z'n kop te staan vanwege RISC-V. Maar waarom? Wat is het? En hoe kun je eraan deelnemen? Misschien hebt u ergens gelezen dat het een type processor is en dat er een aantal chips beschikbaar zijn die erop zijn gebaseerd. U weet misschien ook dat het 'gratis en open' is, wat vooral de opwinding en enorme 'fanbase' verklaart. Laten we dit waas van hysterie oplossen en kijken waar deze technologie werkelijk om draait.

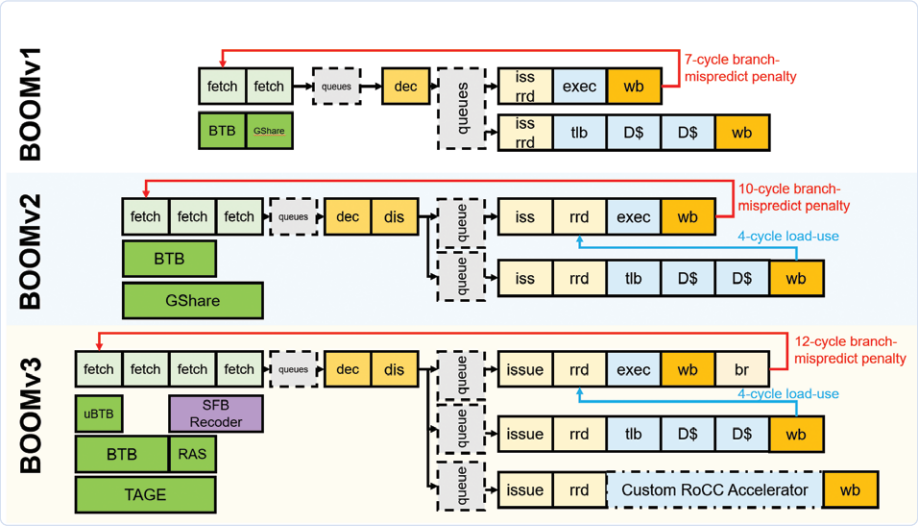
Om te beginnen moet u begrijpen dat RISC-V een Instruction Set Architecture (ISA) is [1] en geen processor. Dit betekent dat de gemeenschap achter RISC-V een beschrijving heeft opgesteld van hoe een processorontwerp zou moeten werken, mocht u besluiten dat te baseren op hun RISC-V ISA. En als we 'ontwerp' zeggen, bedoelen we echt de processor met al zijn registers, accumulatoren, wiskundige bewerkingen, geheugenbussen en alles erop en eraan.

De ISA documenteert de ondersteunde bewerkingen, de geheugenadresseringsmogelijkheden, hoe de stack functioneert en wat er gebeurt als er interrupts optreden, om maar een paar zaken te noemen. Met betrekking tot ondersteunde bewerkingen, wordt uitgelegd hoeveel bits worden gebruikt om een instructie te coderen en welke bits worden gebruikt om de bron van eventuele benodigde operanden te coderen.

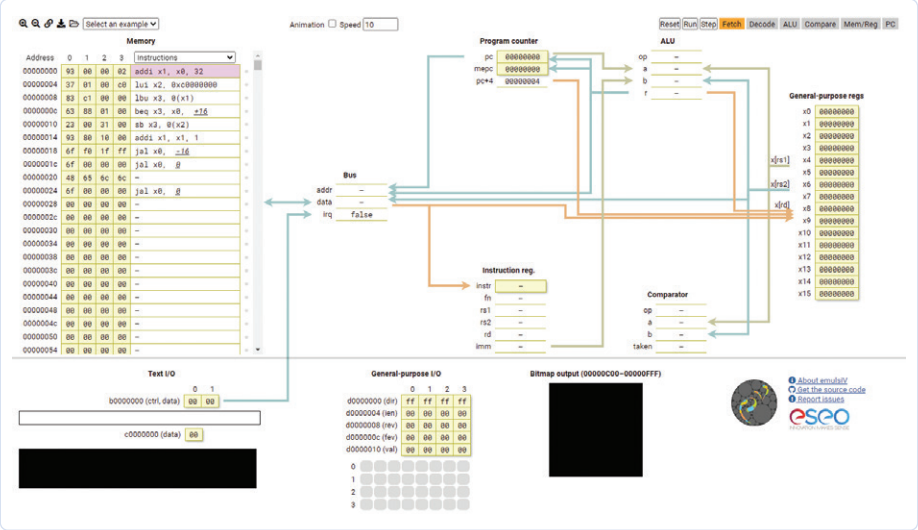
De reden voor alle opwinding is dat de ISA gratis en open is. Open betekent dat iedereen aan de ontwikkeling kan bijdragen, terwijl gratis betekent dat het geen cent kost om te gebruiken. Maar net zoals Arduino-boarddesigns open en gratis gebruikt kunnen worden, wil dat niet zeggen dat het geen geld kost om een board te kopen, en hetzelfde geldt voor het bouwen van uw droomontwerp op basis van RISC-V.

Waartegen neemt RISC-V het op?

Elke processor heeft een ISA; bijna alle zijn bedrijfseigen, en sommige kunnen in licentie worden gegeven. Microchip produceert controllers die 8-bit en 16-bit PIC-processoren gebruiken en ergens is er een ISA die ze beschrijft. Dit zijn bedrijfseigen kernen die toebehoren aan en worden verkocht door Microchip in hun microcontrollers. Als u uw eigen microcontroller wilt bouwen, dan kijkt u waarschijnlijk naar Arm en MIPS. Deze bedrijfseigen kernen kunnen als intellectueel eigendom (IP) in licentie worden gegeven. De bedrijven erachter doen al het werk om de ISA om te zetten in een goed processorontwerp, ondersteunende tools te ontwikkelen, bijbehorende infrastructuur te creëren en u een rekening te sturen als u ze gebruikt. Het wordt echter een uitdaging wanneer u met deze opties *net niet* kunt bereiken wat u wilt. Uw nieuwe toepassing moet wellicht één taak uitvoeren, bijvoorbeeld codering, heel snel maar met een minimaal stroomverbruik. Een potentieel processor-IP in licentie kan uw taak uitvoeren in 100 instructies. Als u nu het stroomverbruik wilt reduceren, moet u een chipsbakker (fab) vinden die gespecialiseerd is in low power. Dat kan duurder zijn dan een fab-proces 'voor algemeen gebruik,' waardoor uw fantastische product te duur wordt voor uw beoogde markt.



Figuur 1. Ontwikkelingsproces voor de RISC-V pipeline-implementatie, gebruikt door het BOOM-project [27] (Bron: Regents ot the University of California).



Figuur 2. Met de emulsiV-simulator kan iedereen RISC-V uitproberen in een webbrowser.

U kunt echter ook enkele slimme ingenieurs hebben die de uitvoeringstijd van uw code kunnen optimaliseren door nieuwe instructies voor de processor te maken, maar omdat de ISA intellectueel eigendom is, mag u die niet wijzigen. U loopt dus tegen een probleem met de processorprestaties aan dat u moet oplossen op fabricageniveau. Verderop meer hierover...

Wat doet RISC-V out-of-the-box?

In het kort: "niet veel maar genoeg". Kortom, u moet beginnen met de keuze van de precieze architectuur die u wilt. Momenteel zijn er 32-bit en 64-bit ISA's gedefinieerd, terwijl ook aan een 128-bit ISA wordt gewerkt. Deze basisdefinities worden RV32I en RV64I genoemd. Als u de RV32I kiest, hebt u 49 instructies [2] tot uw beschikking. De 'I' staat overigens voor 'Integer.' Inbegrepen zijn alle elementaire rekenkundige en logische instructies voor gehele getallen (ADD, SUB, AND, OR, XOR), shifts, compare, jump en link, en enkele systeeminstructies [3]. Als u compacte code wilt ondersteunen, bent u wellicht geïnteresseerd in de 'C'-optie. Deze biedt 16-bits instructiecodering, vergelijkbaar met de Arm Thumb-modus. Vermenigvuldigen en delen (M), atomaire (A) en drijvende-komma (F, D en Q) instructies kunnen ook worden toegevoegd.

De volgende stap is om uw processorkern te ontwerpen op basis van de specificaties van de gekozen opties in een *Hardware Description Language* [4] (HDL), zoals VHDL of Verilog. Omdat dit niet gemakkelijk is, komt de community hier op het toneel. Het ontwerpen van processoren vereist veel vaardigheid, dus er zijn een heleboel mensen en bedrijven die u kant-en-klare ontwerpen bieden. Als u de 'gratis' route wilt volgen, hoeft u niet verder te zoeken dan het PULP-platform [5], opgericht door de ETH Zürich en de Università di Bologna. Hun CV32E40P RV32IM [F] C-implementatie is beschikbaar op GitHub [6], en de instructiedecoder is hier [7] als u wilt zien hoe zoiets wordt geïmplementeerd. Een andere implementatie is het BOOM-project, een hoogwaardige en parametreerbare kern voor architectuuronderzoek, ontwikkeld door de University of California, Berkeley (**figuur 1**). Als u haast hebt en wat ondersteuning kunt gebruiken, moet u met wat geld over de brug komen en een licentie nemen op een implementatie van bijvoorbeeld SiFive [8]. Ze hebben een reeks 32- en 64-bit ontwerpen beschikbaar [9] die ook kunnen worden aangepast.

Hoe kan ik RISC-V uitproberen?

Ondanks dat RISC-V al een tijdje bestaat, is er niet veel silicium beschikbaar om te testen. In industriële context is RISC-V nog relatief nieuw. Als u van microcontrollers houdt, hebt u gezien hoe het grootste deel van de industrie Arm heeft omarmd en afstand heeft genomen van hun eigen cores. Dat was een strategische investering voor de lange termijn. Een overstap naar RISC-V zou alleen maar besparen op de royalty's die aan Arm worden betaald en de gebruikers weinig voordelen opleveren. Ze zouden ook hun ontwikkelingsteams op de hoogte moeten brengen van RISC-V, het moeten integreren met al hun andere IP (analoog, timers, bussen, interfaces, geheugens) en de ontwikkel-IDE, compiler, debugger enzovoort moeten bijwerken.

Als u een harde schijf van Seagate of Western Digital bezit, gebruikt u mogelijk al RISC-V [10] [11]. Maar u wilt waarschijnlijk code op deze kern uitvoeren en niet slechts een product bezitten dat er gebruik van maakt. De snelste manier om dit te doen is met behulp van een simulator zoals emulsiV [12], geleverd door ESEO, die hun RISC-V-kernimplementatie

"Virgule" gebruikt (**figuur 2**). Naast de processor biedt de simulator wat tekst-in- en -uitvoer, een bitmapuitvoer en wat algemene I/O (GPIO). Zeven voorbeelden behandelen de basisprincipes, van optellen en uitvoeren van ASCII-tekst tot het besturen van de GPIO's. Een leuke bijkomstigheid is de 'animate'-optie (selectievakje midden boven) die laat zien waar alle gegevens vandaan komen en naar toe gaan terwijl de code wordt uitgevoerd. Als u wilt, kunt u de code van **listing 1** proberen door het naar een teksteditor te kopiëren, het bestand op te slaan als `program.hex` en het te uploaden naar de simulator.

Wanneer u RISC-V in Arduino-formaat wilt ervaren, is de HiFive1 Rev B momenteel beschikbaar via CrowdSupply [13]. Deze maakt gebruik van de SiFive FE310-G002-microcontroller [14]. Dit is een kaal apparaat met alleen digitale periferie (I²C, UART, SPI, PWM, GPIO) en wat SRAM dat vertrouwt op een off-chip QSPI-flash voor niet-vluchtige opslag. Het board bevat een WiFi- en Bluetooth-module samen met een Segger J-Link voor USB-debugging.

Aan de andere kant van het prestatiespectrum bevindt zich de Microchip PolarFire SoC [15] die vier 64-bit RISC-V-cores naast een FPGA heeft. Dit biedt een zeer configureerbaar platform waarop Linux kan draaien en dat tegelijk 'harde' real-time applicaties ondersteunt.

Hoe pas ik mijn RISC-V aan?

Eerder merkten we al op dat het praktische voordeel van RISC-V is dat u de instructieset kunt afstemmen op uw individuele applicatiebehoeften. Dat betekent dat als de processorimplementatie die u hebt gevonden 95% doet van wat u nodig hebt, u wat handige extra functionaliteit kunt toevoegen om de resterende 5% te implementeren. Laten we aannemen dat uw applicatie intensief gebruik maakt van ChaCha [16] stream cipher ('stroomvercijfering') [17], zoals beschreven in een applicatieblad [18] van Imperas, een andere RISC-V-speler die verificatie-, analyse- en profiling-tools biedt.

U hebt uw ChaCha-implementatie op een RISC-V-kern gedraaid en u hebt gemerkt dat dit een aanzienlijke hoeveelheid verwerkingstijd kost. U wilt niet alleen de uitvoeringstijd verbeteren, maar u wilt ook profiteren van de geringere vermogensopname die daar het gevolg van is, bijvoorbeeld door deze te gebruiken om naar een energiebesparende slaap-modus te gaan.

De code (**listing 2**) maakt uitgebreid gebruik van XOR- en rotate-instructies in een stap die bekend staat als 'quarter rounds' waarvoor vier C `qrX_c()` functies zijn geschreven. Een `processLine()`-functie roept deze vier functies op om de codering uit te voeren. Uit analyse van uitvoeringstijden blijkt dat de processor ongeveer 55% van zijn tijd aan deze taak besteedt, waarvan ongeveer 32% is verdeeld over de vier quarter round-functies (**figuur 3**).

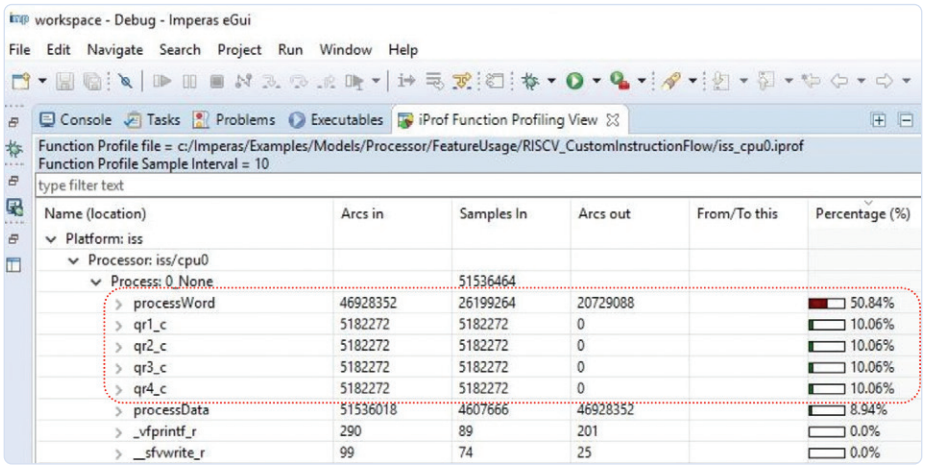
Met RISC-V kunnen we eenvoudig vier speciale quarter round-instructies implementeren die in een enkele cyclus worden uitgevoerd in plaats van te vertrouwen op de code die de C-compiler moet genereren. Dit komt doordat er in de ISA een sectie is gereserveerd voor instructies 'op maat'. In eerste instantie kunnen de instructies worden toegevoegd aan een RISC-V-ontwerp met de implementatie van de instructie gecodeerd in C. Zo kunnen de nieuwe instructies worden gesimuleerd om hun functionaliteit te testen en te controleren of een prestatieverbetering haalbaar is. In dit geval, met speciale quarter round-instructies die beschikbaar zijn op de

Listing 1. Raw HEX-code voor emulsiV-simulator om op te slaan en te uploaden als program.hex.

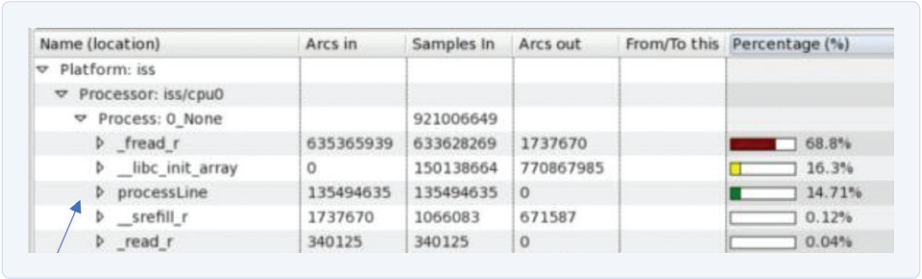
```
:10000000930000023701 00c083c100006388010033
:10001000230031009380 10006ff01fff6f0000007d
:1000200048656c6c6f20456c656b746f72210000c5
:00000001FF
```

Listing 2. C-code om ChaCha stream cipher te implementeren.

```
unsigned int processLine(unsigned int res, unsigned int word) {
  res = qr1_c(res, word);
  res = qr2_c(res, word);
  res = qr3_c(res, word);
  res = qr4_c(res, word);
  res = qr1_c(res, word);
  res = qr2_c(res, word);
  res = qr3_c(res, word);
  res = qr4_c(res, word);
  return res;
}
```



Figuur 3: De ChaCha stream cipher vergde ongeveer 55% van de processortijd met behulp van code die was gecompileerd in standaard C (bron: Imperas Software Limited).



Name (location)	Arcs in	Samples in	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		921006649			
_fread_r	635365939	633628269	1737670		68.8%
__libc_init_array	0	150138664	770867985		16.3%
processLine	135494635	135494635	0		14.71%
__srefill_r	1737670	1066083	671587		0.12%
_read_r	340125	340125	0		0.04%

Figuur 4. Met speciale, nieuw ontwikkelde instructies daalt de belasting van de ChaCha cipher stream-processor tot minder dan 15% (bron: Imperas Software Limited).

aangepaste RISC-V-kern, verbruikt de functie `processLine()` minder dan 15% van de beschikbare processortijd (**figuur 4**). Als dit goed genoeg is, kan het ontwikkelteam de hardware-implementatie van de instructies in Verilog ontwikkelen.

Het gebruik van de nieuwe instructies is helaas niet zo simpel als het opnieuw compileren van de C-code (**listing 3**). Het aanpassen van een RISC-V-compiler om de nieuwe instructies te gebruiken, is een enorme inspanning. In plaats daarvan worden de hex-gecodeerde instructies aangeroepen met inline assembler op dezelfde manier als bij hand-geoptimaliseerde code.

Hoe kan ik bijdragen aan RISC-V?

Als u zou willen bijdragen aan de voortdurende ontwikkeling van RISC-V, dan hebt u geluk! RISC-V International [19] is de organisatie waaraan de ontwikkeling en de promotie van alles wat met RISC-V te maken heeft is toevertrouwd (**figuur 5**). Individuen kunnen deelnemen als Community Members [20]. Als u er een carrière mee wilt opbouwen, zijn er tal van bedrijven en universiteiten [21] die actief betrokken zijn.

Als u verwacht dat een groot assortiment RISC-V-microcontrollers op de markt zal komen, staat u wellicht een teleurstelling te wachten. GigaDevice heeft er een aantal in de aanbieding [22], en een Russische fabrikant levert er een die gericht is op de markt voor slimme meters [23]. Arm is echter te diep verankerd bij de grote spelers, en start-ups zullen moeite hebben om te concurreren op deze verzadigde markt, zelfs met het financiële voordeel van een processor waarvoor geen royalty's hoeven te worden betaald.

In plaats daarvan is de kans groter dat RISC-V wordt gebruikt in gespecialiseerde toepassingen waarbij de mogelijkheid om de kern aan te passen enorme voordelen oplevert, zoals ultra low power [24]. Met problemen rond het licentiëren van technologie aan China, blijkt RISC-V een populair alternatief te zijn voor het verwerven van IP uit de Verenigde Staten. Alibaba heeft een 16-core, 2 GHz, 64-bit RISC-V aangekondigd die is gebouwd in een 12nm-proces [25] en geeft aan dat de kern wordt overwogen voor gebruik in server-infrastructuur. Ten slotte heeft het European Processor Initiative [26]

Figuur 5. Officieel logo van RISC-V International dat de ontwikkeling van de ISA promoot en ondersteunt.

Listing 3. Gebruik van de nieuwe RISC-V-instructies.

Voor het gebruik van de nieuwe RISC-V-instructies is inline assembler vereist. De instructies zijn hexadecimaal gecodeerd omdat de assembler de namen van deze nieuwe instructies niet kent.

```
unsigned int processLine(unsigned int res, unsigned int word) {
  asm __volatile__("mv x10, %0" :: "r"(res));
  asm __volatile__("mv x11, %0" :: "r"(word));
  asm __volatile__(".word 0x00B5050B\n" ::: "x10"); // equivalent to qr1_c()
  asm __volatile__(".word 0x00B5150B\n" ::: "x10"); // equivalent to qr2_c()
  asm __volatile__(".word 0x00B5250B\n" ::: "x10"); // equivalent to qr3_c()
  asm __volatile__(".word 0x00B5350B\n" ::: "x10"); // equivalent to qr4_c()
  asm __volatile__(".word 0x00B5050B\n" ::: "x10"); // equivalent to qr1_c()
  asm __volatile__(".word 0x00B5150B\n" ::: "x10"); // equivalent to qr2_c()
  asm __volatile__(".word 0x00B5250B\n" ::: "x10"); // equivalent to qr3_c()
  asm __volatile__(".word 0x00B5350B\n" ::: "x10"); // equivalent to qr4_c()
  return res;
}
```

gekeken naar heterogene architecturen die ARM en RISC-V (of andere kernen) naast elkaar kunnen bezitten. Het doel is hier om het beste van twee werelden te combineren door de beste processor te gebruiken voor elke taak in multicore-ontwerpen.
RISC-V is niet de eerste gratis en open poging tot processor-IP, maar het is tot nu toe de meest succesvolle. Gezien zijn erfgoed, flexibiliteit, open benadering, interesse vanuit de academische wereld en uitgebreide ondersteuning door de industrie, is het een technologie die een generatie of meer ingenieurs gedurende hun hele loopbaan zal begeleiden.

210223-03

Vragen of opmerkingen?
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via stuart.cording@elektor.com, of naar de redactie van Elektor via redactie@elektor.com.

Een bijdrage van
Tekst en afbeeldingen: **Stuart Cording**
Redactie: **Jens Nickel, C.J. Abate**
Vertaling: **Eric Bogers**
Layout: **Giel Dols**

WEBLINKS

[1] **"Microarchitecture and Instruction Set Architecture," GeeksforGeeks, Oct 2019**: http://bit.ly/3bBKAy2
[2] **"RISC-V," Wikipedia**: https://en.wikipedia.org/wiki/RISC-V
[3] **J. Zhu, "RISC-V Reference"**: http://bit.ly/30IlCXj
[4] **"Hardware Description Language," Wikipedia**: https://en.wikipedia.org/wiki/Hardware_description_language
[5] **PULP Platform-website**: https://pulp-platform.org/
[6] **GitHub-repository - CV32E40P RISC-V core**: https://github.com/openhwgroup/cv32e40p
[7] **SystemVerilog-implementatie van de CV32E40P instructie-decoder**: https://github.com/openhwgroup/cv32e40p/blob/master/rtl/cv32e40p_id_stage.sv
[8] **SiFive-website**: https://www.sifive.com
[9] **SiFive Core Designer**: http://bit.ly/3rKnkU7
[10] **Seagate RISC-V pagina**: http://bit.ly/3etS0oU
[11] **Western Digital RISC-V pagina**: http://bit.ly/3eyldyP
[12] **emulsiV-simulator voor de RISC-V Virgule-processor**: http://bit.ly/30AzqmG
[13] **CrowdSupply-pagina voor HiFive1 Rev B 'Arduino'-board**: http://bit.ly/3eww7Fj
[14] **Datasheet van de SiFive FE310-G002 microcontroller**: https://bit.ly/3bFANak
[15] **PolarFire SoC-productpagina**: http://bit.ly/3bFAU5K
[16] **"ChaCha-variant [van Salsa20 stream cipher]", Wikipedia**: http://bit.ly/3rHfZES
[17] **"Stream Cipher," Wikipedia**: https://en.wikipedia.org/wiki/Stream_cipher
[18] **"Imperas RISCV Custom Instruction Flow Application Note," Imperas Software Limited, October 2019**: http://bit.ly/3vguDVK
[19] **RISC-V - over**: https://riscv.org/about/
[20] **RISC-V - lidmaatschap**: https://riscv.org/membership/
[21] **RISC-V - leden**: https://riscv.org/members/
[22] **GigaDevice GD32 RISC-V microcontrollers**: http://bit.ly/2NbSgO9
[23] **Сергей, "Russian microcontroller K1986BK025 based on the RISC-V processor core for smart electricity meters," november/december 2020**: http://bit.ly/3qGPY7o
[24] **MicroMagic-website**: www.micromagic.com
[25] **J. Aufranc, "More Details about Alibaba XT910 64-bit RISC-V Core," CNX Software, August 2020**: http://bit.ly/3eync6f
[26] **Website van het European Processor Initiative**: www.european-processor-initiative.eu/
[27] **GitHub-repository - BOOM RISC-V core**: https://github.com/riscv-boom/riscv-boom