DVI op de RP2040

vraaggesprek met Luke Wren, chipontwikkelaar bij Raspberry Pi

Mathias Claußen (Elektor Lab)

Luke Wren, een van de engineers van de Raspberry Pi RP2040-microcontroller, heeft verbazingwekkende dingen bereikt door deze kleine chip tot het uiterste te drijven. Elektor-engineer Mathias Claussen wilde meer weten, vooral over de trucs en hacks die nodig zijn om video-uitvoer uit zo'n kleine chip te persen.

De Raspberry Pi RP2040 is een 'sub-$1' MCU met verbazingwekkende mogelijkheden. Een van de ontwerpers is Luke Wren, die niet alleen aan de Raspberry Pi RP2040 heeft gewerkt, maar ook heeft laten zien dat de kleine RP2040 tot DVI in staat is (zie het artikel "Video-output met microcontrollers (2)" [1]. Als hij niet bezig is met geheime projecten voor Raspberry Pi, deelt hij enkele van zijn hobbyprojecten met de community op Twitter (@wren6991) en de code op GitHub [2].

Mathias Claussen: Kun je ons iets over jezelf vertellen?

Luke Wren: Je begint met de moeilijkste vraag! Ik ben engineer bij Raspberry Pi, en als ik dat niet doe, werk ik meestal aan mijn hobbyprojecten, speel ik niet erg goed gitaar, of (meer recent) steek ik tijd in het leren van talen. Ik woon in Cambridge in het Verenigd Koninkrijk, niet omdat het een bijzonder opwindende stad is, maar meer uit luiheid na mijn afstuderen aan de universiteit. Ik heb in Duitsland gewoond toen ik jonger was, maar mijn Duits is nu behoorlijk roestig, dus ik ben blij dat we dit gesprek in het Engels voeren.

Mathias: Hoe lang ben je al bij Raspberry Pi?

Luke: Ik ben in september 2018 begonnen als werknemer, maar ik was daarvoor stagiair bij Raspberry Pi.

Mathias: Wat was je rol bij de ontwikkeling van de RP2040?

Luke: Ik werkte aan een gedeelte van het digitale ontwerp – voornamelijk PIO, DMA, XIP-cache, busstructuur en PWM. Ik heb ook gewerkt aan de boot-ROM en de SDK, en natuurlijk moest ik ook mijn steentje aan de documentatie bijdragen.

Mathias: Video uit een MCU/CPU persen lukte met de Sinclair ZX81, maar DVI is iets nieuws voor een MCU die minder dan 1 dollar kost. De meeste MCU's kunnen VGA genereren, maar wat is de uitdaging als het om DVI gaat?

Luke: Er zijn twee dingen die DVI-D moeilijker maken dan VGA. Het eerste is het serialiseren van de gegevens: de minimale pixelklok voor DVI-D is 25 MHz, en de bitklok is 10 keer zo hoog, dus minimaal stuur je 3 × 250 Mbps differentiële seriële lijnen aan (rood/groen/blauw).
Het tweede is dat DVI-D niet slechts ruwe pixelgegevens verstuurt, maar deze eerst codeert. De codering is eenvoudig in hardware, maar ietwat lastig in de software, vooral wanneer die software de brute snelheid van de seriële uitgang moet bijhouden.
Al het andere is vergelijkbaar. Het is eigenlijk gewoon DPI door een snellere pijplijn. *(Noot van de redactie: Display Pixel Interface is een parallelle RGB-pixelinterface – inclusief pixelklok – om pixelgegevens naar een beeldscherm te transporteren.)*

Mathias: Waarom wilde je DVI bij de RP2040 proberen?

Luke: Toen de stress van het silicium eenmaal voorbij was, wilden een paar van ons eens uitproberen hoe hoog we de klokfrequentie van het systeem konden opvoeren. In de praktijk is er enige marge boven de nominale frequentie van 133 MHz. Ik had met DVI-D op een FPGA gespeeld, als onderdeel van mijn RISCBoy-project [3], en toen ik merkte dat er een overlap was tussen de laagste DVI-bitklokfrequenties en de hoogste systeemklokfrequenties bij de RP2040, ging er bij mij een lampje branden. Ik vroeg me af of het mogelijk was – daarom dus.

Mathias: Wat was de grootste uitdaging om een DVI-uitgang (figuur 1) op de RP2040 te krijgen?

Figuur 1. Raspberry Pi Pico met de Pico DVI Sock (bron: Luke Wren).

Luke: TMDS coderen. Als je het algoritme in de DVI-specificatie volgt, lukt het nooit om het snel genoeg te krijgen op twee Cortex-M0+ kernen die op de bitklokfrequentie draaien. Er zijn dus wat trucs en sluipwegen nodig om het toch mogelijk te maken, en vervolgens wat zorgvuldig handgeschreven code om het voldoende snel te maken. De RP2040 heeft veel geheugen, maar niet genoeg om een compleet frame met TMDS-gecodeerde pixels op te slaan, dus het is een "wedloop tegen de elektronenstraal" bij het coderen.

Mathias: Je moest de RP2040 iets overklokken (van 133 MHz standaard naar 252 MHz). Is er een kritisch pad in de chip voor de DVI-signalen (je moet I/O-pinnen ook sneller aansturen dan standaard)?

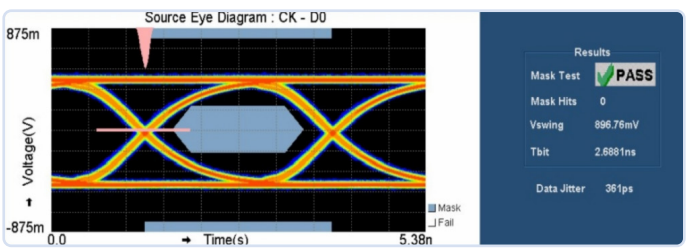
Luke: De eerste beperking waar je bij de RP2040 tegenaan loopt is dat de systeemklok 1:1 moet lopen met de bitklok, dus als je probeert hogere-resolutiemodi te bereiken, dan lopen de processoren gewoon vast. Het kritieke instellingspad voor het systeemklokdomein op de RP2040 wordt gevormd door de adresfase-signalen van de processor naar de SRAM's.
En afgezien daarvan zitten we ook vrij dicht bij de grenzen van wat je door die algemene 3V3-paden kunt sturen; als je kijkt naar het oogdiagram van **figuur 2** voor 720p30 (372 Mbps) op mijn GitHub [2], dan kan dat nèt, maar het is marginaal. Ik betwijfel of je 1080p30 zou zien zonder speciale hardware.

Mathias: Hoe cruciaal zijn, naast de hogere snelheid, de PIO's en de interpolator in de RP2040 om DVI aan de praat te krijgen?

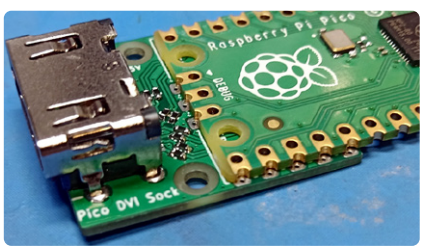
Luke: Je moet hoe dan ook 3 seriële databits plus hun differentiële complementen op GPIO's met minimaal 250 Mbps beschikbaar maken. De mogelijkheid om de single-ended/pseudo-differentiële conversie in de PIO te doen halveert de DMA-bandbreedte, en het opdelen van de TMDS-lanes in 3 FIFO's is handig als je de codering in software doet omdat je dan je code op maat kunt maken voor de codering van de rood/groen/blauw-componenten. Iets als een PIO is dus cruciaal als je geen speciale hardware hebt.
De interpolatoren helpen bij het genereren van de adresen bij de TMDS-codering, wat zeker belangrijk is voor sommige van de demo's die je hebt gezien, maar mijn pixel-verdubbelde TMDS-coderingstruc zou nog steeds passen op een enkele Cortex-M0+ kern zonder interpolatoren.

Mathias: Je Pico DVI Sock voor de RP2040 gebruikt een fysieke HDMI-aansluiting (figuur 3), duidelijk gelabeld met DVI-only, dus we krijgen geen audio. Is dat 'alleen maar' een licentie-kwestie met het HDMI-consortium?

Luke: Er is niets dat je weerhoudt om HDMI data-eilanden toe te voegen en audio uit te voeren. Iemand heeft dat al gedaan gedaan met een NES-emulator-poort [4][5]! Er zijn geen extra fysieke verbindingen nodig voor de audiosignalen, hoewel je strikt genomen geen HDMI-functies mag gebruiken voordat je het display-datakanaal hebt opgevraagd, dat niet is geïmplementeerd op mijn Pico DVI Sock. De situatie met de HDMI-licentie bezorgt zeker hoofdbrekens, en ik heb mezelf al min of meer vastgezet door de repository "PicoDVI" te noemen, dus dit laat ik aan de community over.



Figuur 2. Oogdiagram voor RP2040 DVI bij 720p30 (bron: Luke Wren).



Figuur 3: Pico DVI Sock voor de Raspberry Pi Pico (bron: Luke Wren).

Figuur 4. Hoge-resolutiebeelden dankzij enkele trucs (bron: Luke Wren).

Mathias: Als je de DVI-uitgang gebruikt, hoeveel van de RP2040-resources worden dan daardoor in beslag genomen? Is er tijd over om andere code op de MCU te draaien?

Luke: Het hangt af van de videomodus. Voor een RGB565-uitvoer met dubbele pixels gebruik je uiteindelijk ongeveer 65% van een kern voor TMDS-codering en DMA-interrupts; de andere kern is dan compleet beschikbaar voor het genereren van de video en het draaien van de logica van je hoofdprogramma.

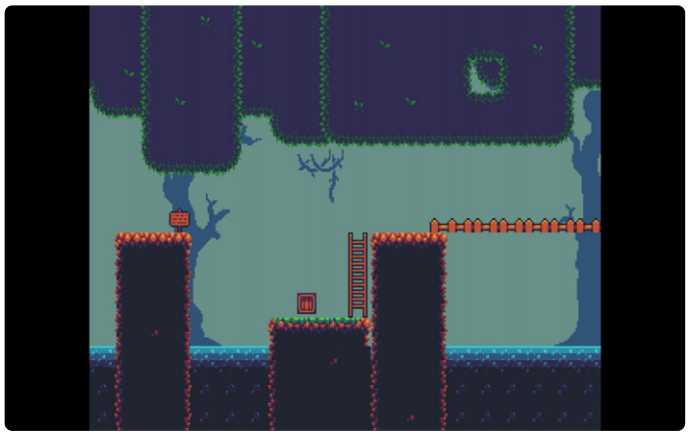
*Noot van de redactie: Naast het genereren van pure video, werden enkele toepassingen voor de Pico DVI Sock toegevoegd. Een daarvan is het bewegen van verschillende Eben Upton-gezichten over het scherm. Als we even rekenen, zou een beeld van 640×480 pixels dat als een volledig frame met 8-bit resolutie is opgeslagen ongeveer 308 KB RAM in beslag nemen (meer dan de RP2040 heeft), dus beperken we ons tot maximaal 320×240 met 16bit-kleur (154 KB) in RAM, maar de demo (**figuur 4**) is niet op die manier gepixeld. Er lijkt dus sprake te zijn van een softwarematige truc.*

Mathias: Bij de software om een DVI-signaal te genereren hoort een bibliotheek die ook sprites verwerkt. Kun je daar meer over vertellen?

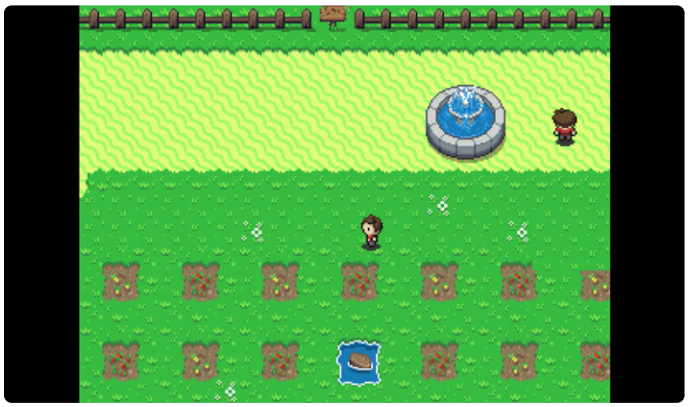
Luke: Zeker! Wanneer je een video-uitvoer schrijft, is het volgende probleem dat je tegenkomt dat je video moet hebben om daadwerkelijk uit te voeren, en terwijl ik aan PicoDVI werkte, heb ik precies voor dat doel de ARMv6-M sprite-bibliotheek in elkaar gehackt. Het belangrijkste kenmerk van deze bibliotheek is dat er geen framebuffer nodig is om in te renderen, alleen een lijnbuffer. Het renderen gebeurt net 'voor de straal', dus vlak voor de TMDS-codering. Dit betekent dat je video-uitvoerresoluties kunt ondersteunen die niet in het geheugen zouden passen als een gewone framebuffer, en het laat het grootste deel van het geheugen vrij voor de eigenlijke grafische activa. Er zijn enkele redelijk snelle 'blit&fill'-routines, een paar tiling-routines en enkele 'affine transformed sprite'-routines waarmee je geschaalde/geroteerde/gekantelde sprites kunt maken. Genoeg voor sommige spelletjes op het niveau van een Game Boy Advance of zo. (*Noot van de redactie: Je kunt een voorbeeld zien in **figuur 5**.*)

Mathias: Waar haalde je de inspiratie voor de bibliotheek vandaan – en de tijd om die te schrijven?

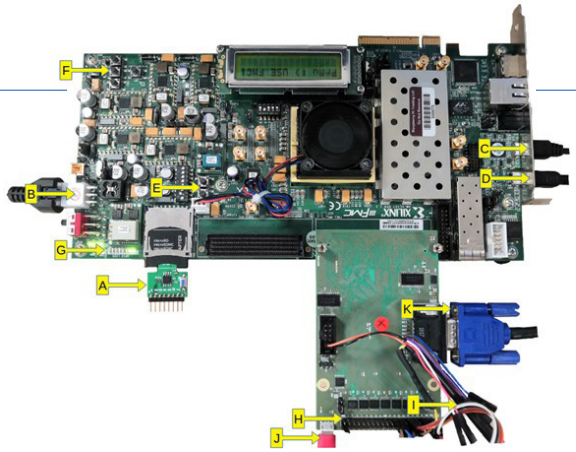
Luke: Ik heb enige tijd gewerkt aan scanline-gebaseerde grafische hardware voor RISCBoy, dus nadat ik dat in hardware had gedaan, was het vrij eenvoudig om het in software te repliceren. Alles in de PicoDVI-repository heb ik in mijn vrije tijd op mijn laptop gedaan, behalve de oogdiagrammen, waarvoor ik een 'scoop op mijn werk heb gebruikt.



Figuur 5. Sprite-demo voor de Raspberry Pi RP2040.



Figuur 6. Walker-demo met DVI-uitvoer.



Figuur 7: Raspberry Pi RP2040 FPGA-prototype (bron: Luke Wren).

Mathias: Er is een door Zelda geïnspireerde Sprite-demo voor de RP2040 (figuur 6). Kun je ons vertellen over het idee daarachter? Een NES of SNES zou speciale hardware gebruiken om zulke beelden samen te stellen, en hier hebben we gewoon twee CPU's die pixels verplaatsen.

Luke: Dat is eigenlijk een geporte versie van een van de RISCBoy-demo's. Zoals je zegt is er veel overhead om dit alles in software te doen, en RISCBoy op 36 MHz kan evenveel sprites op het scherm zetten als de RP2040 op 252 MHz [6].

Mathias: In de genoemde documenteatie van de bibliotheek staat het idee van een Mario Kart-kloon voor de RP2040? Was het gewoon een idee, of ben je aan het sleutelen om het aan de praat te krijgen? Er wordt ook opgemerkt dat de interpolator daarvoor nuttig zou zijn.

Luke: Op dit punt moet ik dus toegeven dat de oorspronkelijke reden voor de interpolator in de chip was dat we texture- en tile-mapping in SNES MODE7-stijl wilden doen, hoewel we tussen dat moment en de tape-out wat tijd hebben besteed om er een bruikbaarder en krachtiger stuk hardware van te maken. We hebben nooit een echte MK-kloon gemaakt, hoewel je online genoeg voorbeelden kunt zien van mensen die soortgelijke technieken gebruiken; we hebben wel een texture-mapped 3D-kubus met Eben's gezicht erop op een FPGA laten draaien.

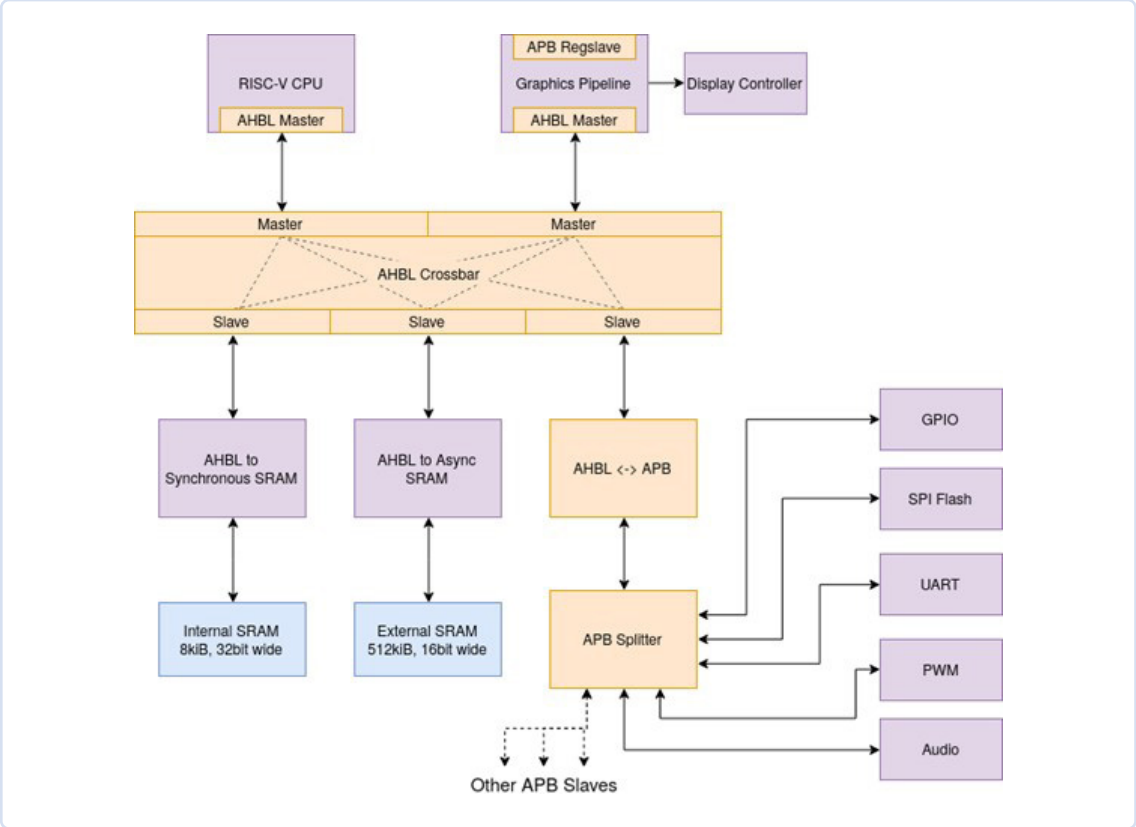
Mathias: In de documentatie van jullie Pico DVI Sock voor de RP2040 Pico, vermelden jullie een 48-MHz FPGA prototype. Kun je ons daar iets over vertellen?

Luke: We moeste 's nachts een FPGA-image maken van de laatste RP2040-broncode zodat we die de volgende dag konden gebruiken voor software-ontwikkeling. Wij hebben gewoon een standaard Virtex 7-ontwikkelboard gebruikt, met een dochterboard om de IO's van de FPGA naar 3,3 V te brengen (**figuur 7**), en een ander board om een QSPI-flash in de SD-aansluiting te steken, die is aangesloten op de RP2040 XIP-interface [7].
De FPGA is vrijwel een volwaardige RP2040 [8] – de klok/reset-schakeling is vereenvoudigd en de ADC is weggelaten, maar verder is alles aanwezig en functioneel. Dit maakt het een ideaal platform voor software-ontwikkeling, hoewel we voor de eigenlijke verificatie van de chip gebruik maakten van conventionele simulaties en formele modelcontrole.

Mathias: Naast de DVI-uitgang zijn/waren jullie bezig met enkele andere projecten. Een daarvan is de PicoStation 3D. Kun je daar iets over vertellen? Als je alle onderdelen bij elkaar kon scharrelen, zou het dan nog steeds hetzelfde ontwerp zijn?

Luke: PicoStation 3D (**figuur 8**) is één van de vele hobby-PCB's die ik in de aanloop naar de lancering van de RP2040 had. Het is een board met een RP2040, een iCE40UP5K FPGA, microSD, audio out, DVI-D out via HDMI, en twee SNES controller-aansluitingen. Ik las op dat moment veel over grafische 3D-hardware en wilde een platform om daarmee te spelen, in de context van een soort kleine spelcomputer.
Het doet me pijn dat ik dat project zo lang op een laag pitje moest zetten, maar naast de problemen met de onderdelen heb ik gewoon te veel andere projecten lopen. Het is allemaal open source, dus ik zou het leuk vinden als iemand anders het idee oppikt en ermee aan de slag gaat.
Ik denk dat de keuze van de FPGA precies goed is – hij is klein en traag genoeg om je hard te laten werken voor je demo's, net geen DVI-D-capaciteit, en hij heeft royaal intern geheugen en een handvol 16-bit DSP-tiles, dus het is een briljant platform om te spelen met grafische spelletjes-hardware. Hij is ook goed te

Figuur 8. Prototype van de PicoStation3D (bron: Luke Wren).

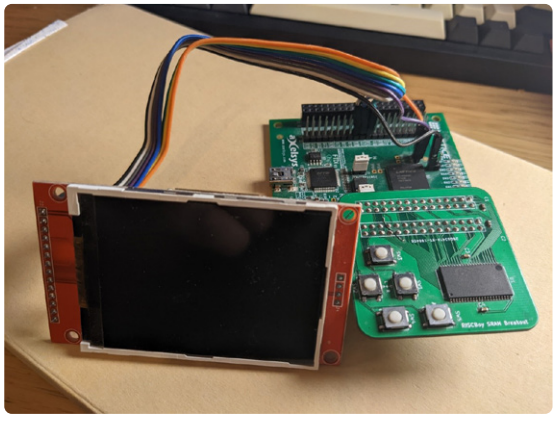


Figuur 9. Architectuur van de RISCBoy (bron: Luke Wren).

combineren met de RP2040. Waar ik over na zou willen denken is de IO. Wat als je bijvoorbeeld de DVI naar de microcontroller verplaatst en de SNES-controllers naar de FPGA, en wat als je het audio-circuit wat beter maakt, dat soort dingen.
Ik denk dat de fysieke vormfactor ongeveer goed is, aangezien deze wordt bepaald door de twee SNES controller-aansluitingen.

Mathias: Naast de Raspberry Pi-gebaseerde items heb je ook een RISCBoy gemaakt met onder de motorkap een zelfontwikkelde RISC-V Core en andere periferie, zoals een grafische engine (2D-sprite-gebaseerd?). Wat kun je ons daarover vertellen?

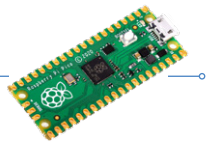
Luke: RISCBoy is mijn ietwat vertraagde concurrent voor de Game Boy Advance. De volledige hardware past in een iCE40 HX8K met wat extern parallel SRAM (**figuur 9**). Uiteindelijk zal er een fysieke versie komen, maar nu is het nog een HX8K ontwikkelboard met wat SRAM en knoppen, en een SPI-display dat er los bij hangt (**figuur 10**). Ik begon eraan te werken rond de tijd dat ik de universiteit verliet.
Alles is zonder uitzondering vanaf nul geschreven – niet de beste engineering-werkwijze, maar het is geweldig om te leren, en het maakt debuggen leuker wanneer er geen enkel deel van de hardware/software-stack is dat je kunt vertrouwen. Er is een 32-bit processor (Hazard5), wat programmeerbare 2D grafische hardware, en alle infrastructuur om het met elkaar te koppelen.
De grafische hardware doet alle gebruikelijke sprites, tiles, affine-transformed sprites/tiles enzovoort, maar doet dit door vanuit het geheugen commandolijsten uit te voeren die beperkte ondersteuning bieden voor control flow en vertakkingen naar subroutines. Tijdens elk frame schrijft de processor de commandolijst voor het volgende frame uit. Er zijn een paar scanline-buffers in de hardware, een waarnaar wordt gerenderd en een die naar het scherm wordt gestuurd, zodat je de bandbreedte, latentie en geheugenruimte van renderen in een framebuffer vermijdt.
Het project staat voorlopig op een laag pitje, maar ik wil het in elk geval ooit voltooien.



Figuur 10. Prototype van de RISCBoy (bron: Luke Wren).

Mathias: Je RISC-V Core wordt met de dag volwassener. Wat was de oorsprong ervan?

Luke: Ik heb momenteel een paar processoren in verschillende stadia van ontwikkeling, maar ik denk dat je de Hazard3 bedoelt. Het is een 3-traps in-order scalaire processor, oorspronkelijk afgeleid van de



Hazard5-broncode (de RISCBoy-processor). Hazard3 is werkelijk een leerplatform voor mij – het is allemaal goed en wel om RISC-V specificaties te lezen, maar dat is niet hetzelfde als ze daadwerkelijk te implementeren; en door een eenvoudige 3-traps als basis te gebruiken kan ik me concentreren op de periferie omdat de eigenlijke core-pijplijn gewoon werkt.
Toch zijn de prestaties tegenwoordig redelijk competitief (rond de 3,2 CoreMark/MHz) en ik heb veel tijd gestoken in verificatie en documentatie. Het op gang brengen van hardware-debugging en het uitvoeren van end-to-end tests met GDB, OpenOCD en mijn eigen JTAG-transportmodule is waarschijnlijk het hoogtepunt van het project tot nu toe, maar ik heb de hele tijd veel geleerd.
Ik denk dat Hazard3 in de toekomst een componenten-goudmijn zal zijn voor alle toekomstige 32-bit embedded-klasse processoren waaraan ik werk. Ik heb al iemand op Twitter mijn debug-module zien gebruiken om debug-ondersteuning toe te voegen aan de core van iemand anders. De broncode is volledig zelfstandig en zou behoorlijk porteerbaar moeten zijn. Hij heeft ook een Apache-2.0 licentie.

Mathias: Momenteel is het ontwerp een 32-Bit RISC-V, volledig als open-source. Heb je op een bepaald moment middelen voor de ontwikkeling gekregen van Raspberry Pi (tijd, hardware, software-tools)?

Luke: Ik doe al mijn thuisprojecten in mijn eigen tijd en met mijn eigen hardware, met open-source tools. Op dit moment heb ik thuis een Ryzen 7 5800X, wat helpt bij het uitvoeren van batch jobs. Ik gebruik Yosys voor FPGA-synthese, nextpnr voor place en route en CXXRTL voor simuleren.
Ik gebruik een iCEBreaker (iCE40UP5K) en een ULX3S (ECP5 85F) als referentieplatforms voor Hazard3, hoewel ik nog een behoorlijk aantal andere FPGA-ontwikkelboards heb die ik eigenlijk vaker zou moeten gebruiken. Als ik iets moet debuggen op een FPGA, kan ik prima uit de voeten met mijn Saleae Logic 8 die ik kocht toen ik nog studeerde, maar meestal kan ik vertrouwen op simulaties.

Mathias: Momenteel ga je met je core van 32 bit naar 64 bit. Wat was (tot nu toe) het lastigste obstakel tijdens de ontwikkeling?

Luke: Ik heb tot nu toe slechts een weekend met RV64 gespeeld, en dat was genoeg om de RV64IC compliance en de debug compliance tests te doen slagen, dus er is geen enorme leercurve – het wordt gewoon groter en verloopt langzamer.
Ik hackte op de Hazard3-codebase voor het gemak, maar als ik serieus werk wilde maken van een RV64-implementatie, dan zouden er enkele micro-architecturele veranderingen nodig zijn. Er is een reden waarom je niet veel drietraps 64-bit processoren ziet, en zeker niet die met de branche decision in trap 2. Hazard3 blijft een 32-bit embedded-klasse processor.

Mathias: Zijn er plannen om de core in de toekomst op echt silicium uit te proberen? Of zelfs om de core te combineren met een 2D-engine om een RISCBoy64 te krijgen?

Luke: Ik zou graag eens een SkyWater PDK tape-out willen proberen, maar voorlopig concentreer ik me op andere projecten. Ik wil iets interessanters bouwen dan 'gewoon een ander RISC-V systeem' en ik weet nog niet precies wat dat gaat worden. Voor iets als RISCBoy biedt het gebruik van een 64-bit processor niet veel voordelen.

Mathias: Bedankt voor je tijd, Luke Wren.

220575-03

Gerelateerde producten

- **Raspberry Pi Pico (SKU 19562)**
  www.elektor.nl/19562
- **DVI Sock for Raspberry Pi Pico (SKU 19925)**
  www.elektor.nl/19925

WEBLINKS

[1] Mathias Claußen, "Video-output met microcontrollers (2)", Elektor maart/april 2023: http://www.elektormagazine.nl/220614-03
[2] Luke Wren's GitHub-repository: https://github.com/Wren6991
[3] RISCBoy @ GitHub: https://github.com/Wren6991/RISCBoy
[4] pico-infones @ GitHub: https://github.com/shuichitakano/pico-infones
[5] Video @ Twitter: https://twitter.com/shuichi_takano/status/1477702448907419649
[6] Sprite-demo op de RISCBoy-hardware bij 36 MHz: https://twitter.com/wren6991/status/1333708886956707840
[7] Foto van het QSPI SD-board: https://twitter.com/wren6991/status/1134719550027632640
[8] RP2040-microcontroller: https://raspberrypi.com/products/rp2040/