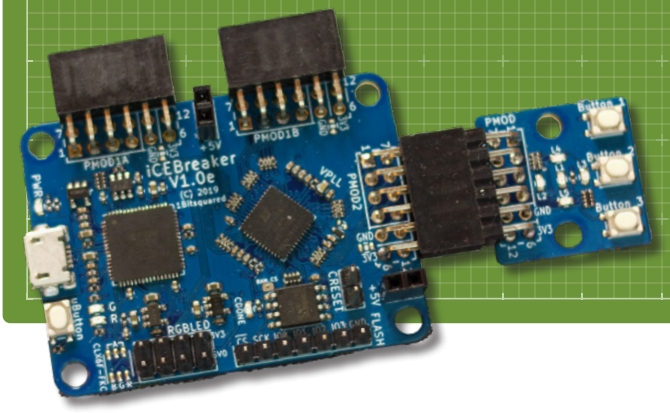
Bouw uw eigen RISC-V Controller

eerste stappen met de NEORV32 RISC-V softcore voor goedkope FPGA's



Mathias Claußen (Elektor)

Wilt u experimenteren met RISC-V? Dat gaat nu ook zonder hard-wired chip. Gebruik de NEORV32 RISC-V softcore voor goedkope FPGA's.

Wie wil experimenteren met RISC-V-microcontrollers, kan kiezen uit verscheidene processoren die deze open-standaard instructiesetarchitectuur gebruiken, zoals de nieuwe ESP32-C3. Maar u hoeft geen hard-wired chip te gebruiken: er zijn alternatieven. Het NEORV32-project biedt een RISC-V softcore-ontwerp dat u met behulp van een FPGA bouwt. De voltooide processor is ietwat minder krachtig dan een hard-wired exemplaar, maar biedt daarvoor veel meer flexibiliteit zodat verschillende ontwerpen kunnen worden getest en eventuele zelfontworpen periferie ook kan worden geïntegreerd in de hardware. En onderweg leert u veel over (bijvoorbeeld) de werking van een CPU.

Een praktische toepassing

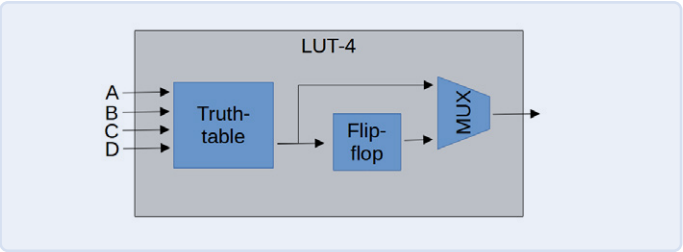
Zelfs wie al een beetje met FPGA's hebben gestoeid, zal snel problemen tegenkomen als het gaat om het configureren van het eigen kleine processorontwerp. Het hele proces kan zelfs voor ervaren ingenieurs een behoorlijke uitdaging zijn, zoals onze artikelreeks over het SCCC-project van Martin Oßmann [1] bewees. Gelukkig hoeft u niet van de grond af aan te beginnen. U kunt profiteren van enkele (bijna kant-en-klare) oplossingen die zijn ontwikkeld door toegewijde experts die hun ontwerpen gratis beschikbaar hebben gesteld.

Hier gebruiken we zo'n oplossing, die ook onder een open source-licentie valt. Dit artikel niet bedoeld als een uitgebreide cursus over RISC-V- of FPGA-technologie, maar het zou uw leercurve moeten helpen verkorten door te tonen hoe u snel uw eerste praktische toepassing kunt bouwen en gebruiken.

FPGA, synthese, softcore, RISC-V en de compiler

Wanneer u een 'algemene' microcontroller voor een specifieke toepassing moet kiezen, kan een hele reeks factoren uw beslissing beïnvloeden. Een van de meest fundamentele overwegingen is de verscheidenheid aan ingebouwde periferie die de controllerchip biedt. Al deze functies zijn vastgelegd in de hardware van de specifieke versie van de chip en kunnen niet worden gewijzigd. Deze aanpak stelt fabrikanten in staat om goedkope chips met geoptimaliseerde prestaties te produceren. Dat ligt anders als u uw eigen controller ontwerpt op basis van een FPGA.

Een FPGA zelf bestaat uit een aantal logische cellen, de opzoektabellen (LUT's, *Look-Up Tables*), die via een matrix flexibel aan elkaar kunnen worden gekoppeld. De blokken in zo'n LUT zijn weergegeven in **figuur 1**. Een voorbeeld is een LUT-4 element met vier ingangssignalen, een waarheidstabel, een flipflop en een multiplexer aan de uitgang. De waarheidstabel kan worden gebruikt om elke gewenste logische basispoort te vormen (AND, OR, NOT of EXCLUSIVE OR). In combinatie met de matrix binnen de FPGA kunnen deze componenten worden gebruikt om complexere structuren zoals geheugens, optellers of multiplexers te bouwen, die op hun beurt kunnen worden gecombineerd tot een nog complexer systeem zoals een processor of een compleet System-on-Chip. De FPGA kan worden vergeleken met een Lego-bouwdoos waarmee u bijvoorbeeld een kasteel kunt bouwen en dat weer afbreken om een brug of iets anders te maken – maar waarbij u steeds dezelfde stenen gebruikt.



Figuur 1. Signaalstroom in een LUT-4.

Om de FPGA een specifieke functie te laten uitvoeren, moet deze op de juiste manier worden geconfigureerd. Het is echter niet nodig om elke afzonderlijke LUT moeizaam met de hand te creëren en ze in de matrix met elkaar te verbinden. Dit is de taak van de FPGA-synthesetools. De gewenste functionaliteit kan worden beschreven in talen als Verilog of VHDL. De meeste synthesetools verstaan beide hardware-beschrijvingstalen. De synthesetool kent de eigenaardigheden van de FPGA en creëert een bitstroom uit de beschrijvingstaal, die vervolgens wordt gebruikt om de FPGA te configureren. **Figuur 2** toont ruwweg het syntheseproces voor een FPGA. De meeste FPGA-fabrikanten bieden gratis tools die meestal met Windows en Linux overweg kunnen. Sommige FPGA's worden ondersteund door open source-oplossingen die dit syntheseproces kunnen uitvoeren en bij voorkeur draaien onder een Unix-achtig besturingssysteem zoals Linux of macOS.

Een FPGA met voldoende LUT's kan niet alleen eenvoudige logische functies uitvoeren, maar ook als complete processor of processorsysteem fungeren. Deze kunnen ook worden beschreven met Verilog of VHDL. Aangezien deze processorkern niet 'hard' in het FPGA-silicium is bedraad, kan de functionaliteit of het gedrag kan worden aangepast door de hardwarebeschrijving te wijzigen. De processoreenheid wordt een softcore genoemd. Dergelijke softcores zijn beschikbaar voor verschillende processorarchitecturen. In sommige gevallen bevatten deze softcores ook periferie zoals businterfaces en dergelijke. De open source RISC-V processorarchitectuur wordt als softcore steeds populairder. De keuze aan hard-wired RISC-V MCU's is momenteel nog overzichtelijk, zodat u met een softcore eerste ervaringen met de architectuur kunt opdoen. U kunt uw eigen RISC-V MCU bouwen om deze te testen en te bestuderen.

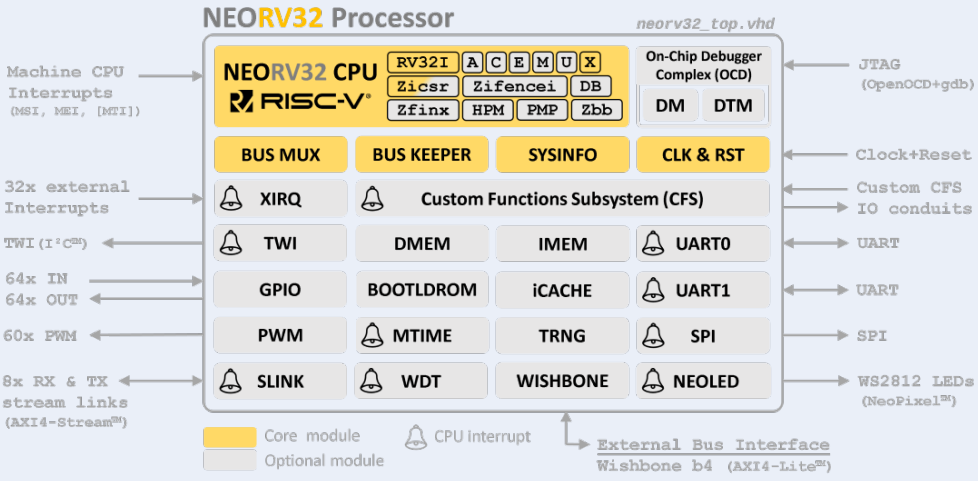
Als u RISC-V gebruikt, zijn er geen licentiekosten, NDA's of andere licentieovereenkomsten die gewoonlijk worden geassocieerd met het gebruik van andere propriëtaire architecturen. RISC-V betekent ook dat er al compilers voor het verwerken van C-broncode beschikbaar zijn, onder meer in de vorm van de GNU C-compiler (GCC). Dit betekent dat ook de basisbibliotheken voorhanden zijn.

NEORV32

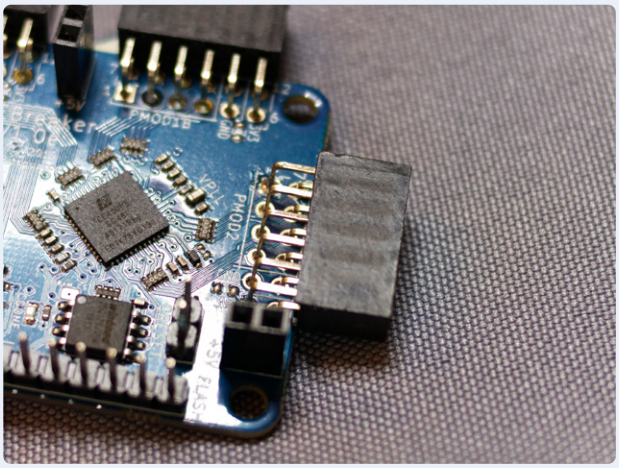
Het NEORV32-project van Stephan Nolting [4] bewijst dat geen dure FPGA hoeft te gebruiken om uw eigen System on Chip (SoC) te bouwen. De NEORV32 is een RISC-V-compatibele CPU met alle perifererie om als een microcontroller-achtige SoC op een FPGA te draaien. Het project is volledig uitgevoerd in platformneutrale VHDL en is dus niet gebonden aan individuele FPGA-fabrikanten. De NEORV32 is niet alleen volledig open source, hij wordt ook geleverd met uitgebreide documentatie, een software-framework en tools.

De geïmplementeerde periferiemodules zijn te zien in **figuur 3**. SPI, I²C en UART's zijn beschikbaar, evenals GPIO's, PWM-eenheden en een WS2812-interface. Dit geeft beginners en gevorderde gebruikers een compleet systeem inclusief een complete ontwikkelomgeving voor de NEORV32 met alle benodigde bibliotheken voor de hardware en periferie. Daarnaast zijn voor sommige FPGA-boards al voorbeeldconfiguraties beschikbaar, zodat u zonder al te veel problemen aan

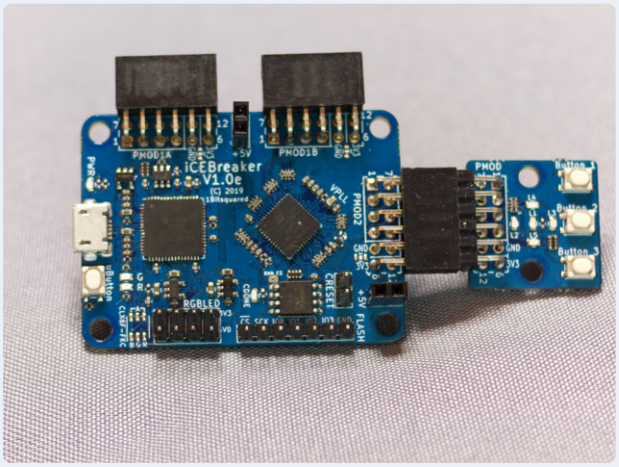
Figuur 2. De stappen in het FPGA-syntheseproces.



Figuur 3. Dit blokschema toont de NEORV32-functieblokken (bron: Github/Nolting, S. [20]).



Figuur 4. De iCE40UP5K in een QFN-behuizing van 7 x 7 mm.



Figuur 5. Het iCEBreaker-board met PMOD-header.

de slag kunt. Maar welke van deze boards werken "out-of-the-box"? En hoe moeilijk is het om de NEORV32 op een FPGA-board te krijgen dat niet direct wordt ondersteund?

Kies een FPGA

In principe kan elk bestaand FPGA-board met voldoende resources worden gebruikt, aangezien de NEORV32 geen fabrikantspecifieke extensies gebruikt. Een FPGA die op verschillende betaalbare boards is gemonteerd, is de Lattice iCE40UP5K [5].

De Lattice iCE40UP5K FPGA is momenteel de grootste versie van de iCE40 Ultra-Plus varianten. In totaal heeft de chip 5280 LUT's, 120 kbit (15 kB) EBR RAM, 1024 kbit (128 kB) SPRAM en hard-wired functionele eenheden voor SPI en I²C die een solide funmament vormen om uw eigen projecten te bouwen. Het zijn niet alleen deze eigenschappen die deze FPGA interessant maken, maar ook de behuizing en de lage kosten. Een QFN-48 behuizing van 7 x 7 mm (**figuur 4**) is veel gemakkelijker om mee te werken dan een chip met een BGA-behuizing; hij kost ongeveer € 5 per stuk; de prijsklasse is dus zonder meer interessant. Op het moment van schrijven (oktober 2021) kost de FPGA bij alle distributeurs € 5...6 per stuk; hij is echter nergens op voorraad terwijl de levertijd maximaal 46 weken kan bedragen.

Voor onze doeleinden hoeft u zelf geen print voor de FPGA te ontwerpen. Het iCEBreaker FPGA-board (**figuur 5**) en de iCEBreaker Bitsy (**figuur 6**) van 1BitSquared [6] zijn twee open-hardware development boards waarop de iCE40UP5K FPGA al is gemonteerd. Een andere open hardware-optie is de UPduino V3.0 [7] van tinyVision.ai (**figuur 7**). Het NEORV32-project ondersteunt het UPduino V3.0-bord volledig en bevat enkele extra voorbeeldprojecten. Eventuele wijzigingen die nodig zijn voor het iCEBreaker FPGA-bord kunnen zeer snel worden uitgevoerd. Om te beginnen zullen we de UPduino V3.0 gebruiken en dan laten we zien wat er aan ons voorbeeldproject moet worden aangepast zodat het op het iCEBreaker-board draait.

Het gebruik van deze FPGA resulteert in een SoC met 64 kB ruimte voor applicaties, 64 kB RAM, SPI-interface, I²C-interface, 4 ingangs- en 4 uitgangspinnen, 3 PWM-pinnen en een UART, en dat in combinatie met de RV32IMAC-kern op 18 MHz.

De toolchain

Voor de toolchain voor de Lattice iCE40up5k staan u twee mogelijkheden ter beschikking. U kunt de tools van Lattice [8] installeren of de OSS CAD-suite van YosysHQ [9] gebruiken die volledig is gebaseerd op open source-tools. Voor dit project kiezen we voor die laatste optie: de open source-route. In dat geval is Ubuntu 20.04 LTS ons besturingssysteem en gebruiken we de OSS CAD Suite om de NEORV32 voor de iCE40UP5K te synthetiseren.

Naast de tools voor de FPGA compileren we later ook een demoprogramma dat op de gesynthetiseerde RISC-V-processor draait: een klassieke implementatie van de 'Hello World'-boodschap wordt verzonden met behulp van de UART. Ook hier worden open source-tools gebruikt om een bruikbare toolchain te genereren (vergelijkbaar met de Kendryte K210 [10]). Een GNU C-compiler (GCC) en extra bibliotheken voor de NEORV32-periferie zijn beschikbaar.

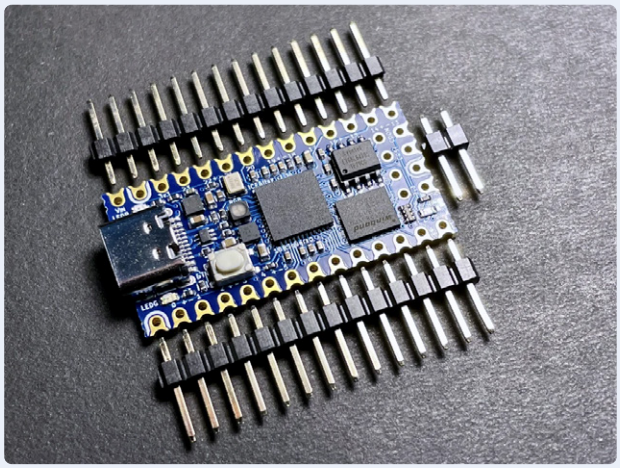
Mise en place

Zoals mijn collega Clemens Valens in zijn video [11] demonstreerde, lijkt het werken met FPGA's een beetje op het bereiden van een maaltijd. Als eerste moeten we er voor zorgen over alle ingrediënten en het benodigde keukengerei (gereedschap) klaarstaan. Als u geen risico wilt lopen dat de installatie van uw hoofdbesturingssysteem wordt beïnvloed, kunt u ook een virtuele machine implementeren. We veronderstellen hier een vers geïnstalleerde versie van Ubuntu 20.04 op een AMD64-systeem. Het zou ook mogelijk moeten zijn een Raspberry Pi te gebruiken, aangezien zowel Ubuntu 20.04 als het Raspberry Pi OS gebaseerd zijn op Debian, maar er kunnen kleine verschillen zijn vanwege de architectuur. Voor dit project gebruikte ik Ubuntu 20.04 die 'solo' op een AMD64-machine draaide.

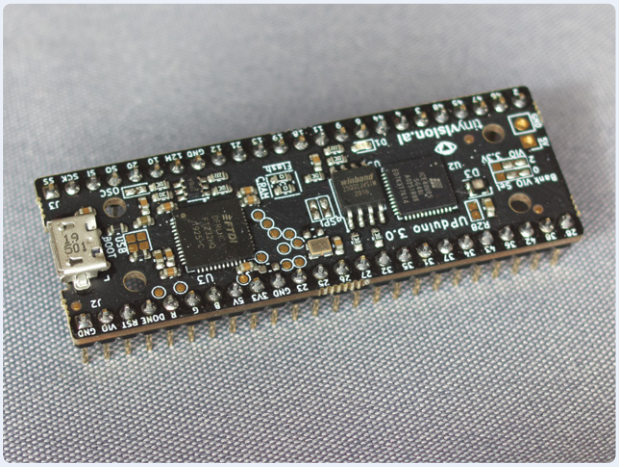
Om de "hardware" voor de FPGA te synthetiseren, moet de huidige versie van de OSS CAD Suite [12] worden gedownload naar de home-map. Dit bestand heet *oss-cad-suite-linux-x64-xxxxxxxx.tgz*. In een terminal wordt dit bestand nu uitgepakt met `tar -xvzf oss-cad-suite-linux-x64-xxxxxxxx.tgz` en vervolgens verplaatst naar */opt* met `sudo mv ~/oss-cad-suite /opt/`. Om later toegang tot die map te hebben, worden de rechten ingesteld met `chmod 777 /opt/oss-cad-suite -R`. De bibliotheek *libgnat-9* is ook vereist; installeer die met `sudo apt install libgnat-9`. Nu staan de FPGA-tools klaar.

Compiler

Vervolgens hebben we de bestanden voor de NEORV32 [4] uit de GitHub-repository nodig. Hiertoe installeren we *git* door `sudo apt install git` in te voeren . De NEORV32-repository wordt vervolgens



Figuur 6. iCEBreaker Bitsy (bron: https://cdn.shopify.com/s/files/1/1069/4424/products/IMG_3859_large.jpg / 1BitSquare).



Figuur 7. De UPduino V3.0 met gemonteerde pinheader-strips.

gekloond met `git clone https://github.com/stnolting/neorv32.git ~/neorv32`.
Om later met de NEORV32 te kunnen communiceren, is een terminalprogramma nodig. Hiervoor gebruik ik normaal gesproken HTerm van Tobias Hammer, wat een handig hulpmiddel is gebleken. Dit programma kan worden gedownload van zijn homepage [13] met `wget https://www.der-hammer.info/terminal/hterm-linux-64.tgz`. Met `mkdir ~/hterm && tar -xvf hterm-linux-64.tgz -C ~/hterm` wordt de inhoud van het *tgz*-bestand uitgepakt naar de map *~/hterm*. Om ervoor te zorgen dat de seriële interfaces later door een gebruiker kunnen worden benaderd, moeten ze als lid aan de dialout-groep worden toegevoegd. Gebruik hiervoor de terminal en voer `sudo adduser $(whoami) dialout` in.
De C-compiler moet nu zelf gecompileerd worden. In de voorbeelden voor de iCE40up5k is de NEORV32 geconfigureerd als RV32IMAC, zodat commando's voor vermenigvuldigen en delen van integers beschikbaar zijn (zie ook het kader **RISC-V Naming Convention**). De compiler moet worden gecompileerd in overeenstemming met deze specifieke architectuur en de opdrachtextensies; waarom deze aanpassing belangrijk is, kunt u in de NEORV32-documentatie [14] nalezen. Ga in een terminal eerst naar de thuismap met `cd ~` en voer `git clone https://github.com/riscv/riscv-gnu-toolchain --recursive` in om de RISC-V Toolchain te klonen. We hebben nog een paar extra pakketten nodig die kunnen worden geïnstalleerd met behulp van:

```
sudo apt-get install autoconf automake autotools-
   dev curl python3 libmpc-dev libmpfr-dev libgmp-
   dev gawk build-essentiële bison flex texinfo gperf
   libtool patchutils bc zlib1g-dev libexpat-dev
```

Ten slotte kunnen de compiler en bibliotheken worden gecompileerd. Bij RISC-V CPU-modellen bestaat er een hiërarchie in termen van de ondersteunde commando's. Dit betekent dat code die is gecompileerd voor een RV32I-processormodel ook op een RV32IMAC-model kan draaien, maar niet andersom. Het advies is daarom om eerst de toolchain zo te compileren dat deze compatibel is met het 'kleinste gemene deler'-model. Met behulp van de terminal kunnen we overstappen naar de map met de gekloonde toolchain met `cd ~/riscv-gnu-toolchain` en dan `./configure --prefix=/opt/riscv --with-arch=rv32i --with-abi=ilp32` gebruiken om de toolchain te preconfigureren voor compilatie. Nu kunnen we het compilatieproces starten met `sudo make`. De toolchain is dan te vinden in */opt/riscv*. Om ervoor te zorgen dat iedereen toegang heeft tot de compiler, moeten de rechten voor *of/opt/riscv* worden gewijzigd. Met behulp van een terminal kunt u iedereen toegang verlenen met het commando `chmod 777 /opt/riscv -R`.
Voor de OSS CAD Suite en de RISC-V GCC-toolchain moet de padvariabele in het */etc/enviroment*- bestand worden gewijzigd. Met behulp van `sudo nano /etc/enviroment` kunnen we het bestand openen en na `PATH="` de string `/opt/oss-cad-suite/bin:/opt/riscv/bin:` invoeren; daarna wordt het bestand opgeslagen.
De meeste FPGA-boards gebruiken een FT232H-chip van FTDI voor de programmeerinterface. Om ervoor te zorgen dat een gebruiker er zonder root-rechten toegang toe krijgt, moet een passende *udev*-regel worden gemaakt voor de FTDI-chip. Gebruik de terminal en voer `sudo nano /etc/udev/rules.d/53-lattice-ftdi.rules` in om een nieuw bestand te openen waarin geschreven kan worden. In dit bestand moet het onderstaande worden ingevoerd:

```
ACTION=="add", ATTR=="0403", ATTR=="6010",
   MODE:="666"
ACTION=="add", ATTR=="0403", ATTR=="6014",
   MODE:="666"
```

Sla het bestand vervolgens op. De voorbereidingen zijn nu afgerond en de synthese en daaropvolgende upload van ons eerste testprogramma kan beginnen. Eerst moet opnieuw worden opgestart, zodat alle instellingen van kracht worden. Een andere aanbeveling is om een editor te installeren met syntax highlighting voor gebruik met VHDL en Verilog.

NEORV32 voor de FPGA

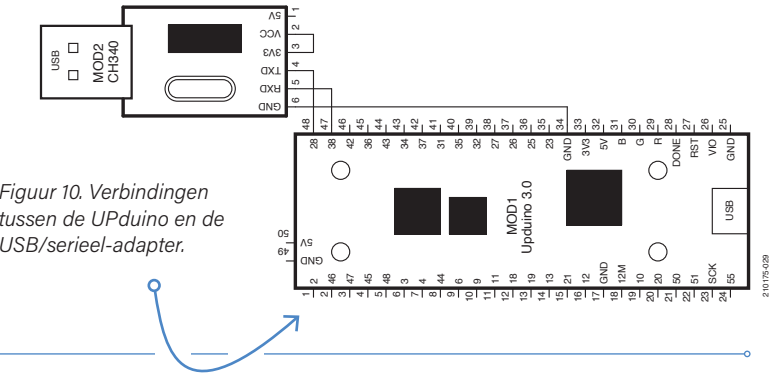
In spanningsloze toestand is de FPGA niet geconfigureerd. Bij het opstarten leest de iCE40UP5K de verbindingsbeschrijving uit een extern SPI-flashgeheugen. Deze beschrijving van de interne verbindingen moet eerst worden opgeslagen in de SPI-flash op de UPDuino V3.0 of de iCEBreaker.
In dit artikel maken we een voorbeeldproject voor het UPduino V3.0-board, dat de processor en periferie bevat. Hierdoor ontstaat een systeem dat direct op de FPGA moet kunnen draaien.

```
MEMORY
{
/* section base addresses and sizes have to be a multiple of 4 bytes */
/* ram section: first value of LENGTH => data memory used by bootloader (fixed!); second value of LENGTH => *physical* size of data memory */
/* adapt the right-most value to match the *total physical data memory size* of your setup */

  ram  (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED(make_bootloader) ? 512 : 8*1024

/* rom and iodev sections should NOT be modified by the user at all! */
/* rom section: first value of ORIGIN/LENGTH => bootloader ROM; second value of ORIGIN/LENGTH => maximum *logical* size of instruction memory */

  rom   (rx) : ORIGIN = DEFINED(make_bootloader) ? 0xFFFF0000 : 0x00000000, LENGTH = DEFINED(make_bootloader) ? 32K : 2048M
  iodev (rw) : ORIGIN = 0xFFFFFE00, LENGTH = 512

}
/* ************************************************************************** */
```

Figuur 8. Om de RAM-grootte te configureren moeten deze wijzigingen worden doorgevoerd.

```
user@user-VirtualBox:~/neorv32/sw/example/hello_world$ make exe
Memory utilization:
   text    data     bss     dec     hex filename
   5576       0     116    5692    163c main.elf
Executable (neorv32_exe.bin) size in bytes:
5588
user@user-VirtualBox:~/neorv32/sw/example/hello_world$
```

Figuur 9. NEORV32_exe.bin wordt gemaakt.



Figuur 10. Verbindingen tussen de UPduino en de USB/serieel-adapter.

Ga nu in een terminal naar de voorbeelddirectory voor de open source-tools met `cd ~/neorv32/setups/osflow/`. Om de synthese te starten en daarmee de bitstream voor de FPGA te genereren, hoeven we slechts `make BOARD=UPduino UP5KDemo` als commando in te voeren; daarna kan het even duren voordat het syntheseproces is voltooid. Dan staat in de map *~/neorv32/setups/osflow/* een bestand met de naam *neorv32_UPduino_v3_UP5KDemo.bit*. De bitstream in dit bestand beschrijft hoe de logische basisblokken in de FPGA met elkaar moeten worden verbonden. Deze bitstream moet nu in de SPI-flash op het FPGA-board worden geschreven.

Nu kunnen we het UPduino-bord via een USB-kabel op de PC aansluiten. In de terminal voeren we `iceprog ~/neorv32/setups/osflow/neorv32_UPduino_v3_UP5KDemo.bit` in. Dit start het programmeren van de externe SPI-flash; de configuratie wordt vervolgens in de FPGA geladen. Dit betekent dat we nu een RISC-V systeem hebben geconfigureerd in de UPduino V3.0, die we nu kunnen voorzien van software. Zoals we al schreven, is 64 KB geheugen beschikbaar voor applicaties en 64 KB als RAM. Voor de randapparatuur hebben we een SPI-interface, I²C, een UART, vier ingangen en vier uitgangen, evenals drie PWM-uitgangen. De hier gesynthetiseerde CPU is een RV32IMAC-model met een klok van 18 MHz. De SoC in de FPGA heeft ook een kleine bootloader die toegankelijk is via de UART.

Hello World

De FPGA is nu voorzien van de NEORV32 en de eerste *Hello World*-demo kan worden gecompileerd en worden geladen. De NEORV32 is configureerbaar, dus de actuele grootte van het RAM-geheugen moet op de juiste manier worden gedefinieerd in het linker-script. Open de terminal en voer `nano ~/neorv32/sw/common/neorv32.ld` in regel 62 in, en ook

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED
   (make_bootloader) ? 512: 8*1024
```

in plaats van (zie **figuur 8**)

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED
   (make_bootloader) ? 512: 64*1024
```

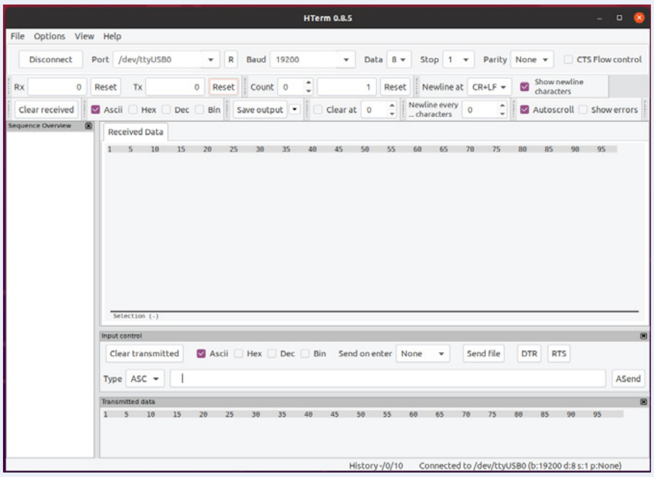
De RISC-V-compiler is nu klaar voor gebruik. Ga in een open terminal naar de map met het *Hello World*-programma met:

```
cd ~/neorv32/sw/voorbeeld/hello_world
```

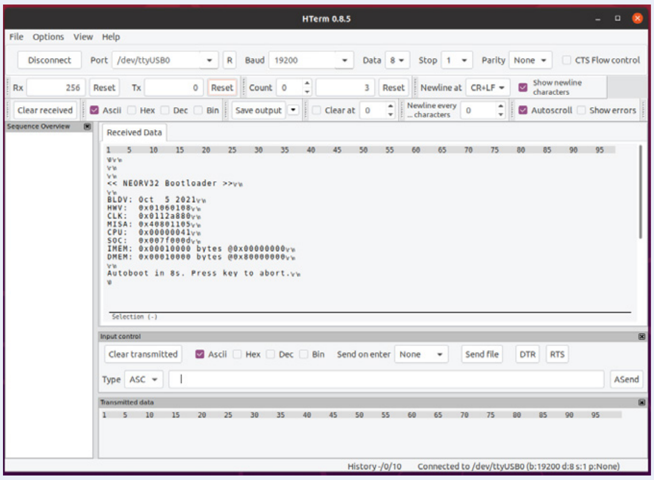
Om een uitvoerbaar bestand te genereren dat in de NEORV32 kan worden geladen, hoeft u alleen maar `make exe` in te voeren om het bestand *neorv32_exe.bin* te genereren (**figuur 9**). Om ervoor te zorgen dat de NEORV32 kan worden uitgevoerd, moet deze nu in het board worden geladen. Hiervoor wordt de geïntegreerde bootloader gebruikt die data ontvangt via de UART (19200 baud, 8 databits, 1 stopbit, geen pariteit, geen flow control). Als een UPduino V3.0-board wordt gebruikt, is een externe USB/serieel-converter vereist, zoals de op de CH340 gebaseerde versie die in de Elektor-shop verkrijgbaar is (zie het kader **Gerelateerde producten**). Deze moet worden aangesloten zoals geschetst in **figuur 10**.

Voor het eigenlijke uploaden gebruiken we HTerm. Dit kan worden gestart vanaf een terminal met `~/hterm/hterm`, er moet dan een venster zoals in **figuur 11** verschijnen. De USB/serieel-converter moet nu als poort worden geselecteerd; gewoonlijk verschijnt die als */dev/ttyUSB0*; afhankelijk van de hardwareconfiguratie en de geselecteerde adapter kan dit echter afwijken.

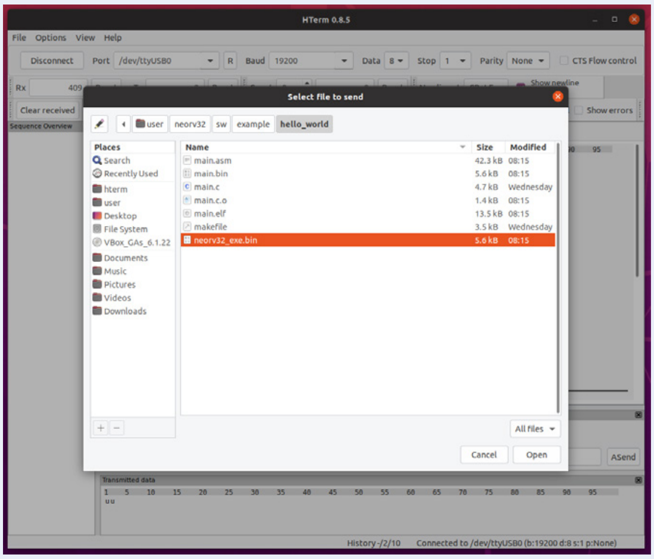
Na aansluiting op de USB/serieel-converter kan de UPduino van spanning worden voorzien en moet de bootloader zich melden (**figuur 12**). Als er niet snel genoeg een karakter naar de bootloader wordt gestuurd, probeert die automatisch op te starten vanaf de SPI-flash, die momenteel geen software bevat. Elk teken dat binnen 8 seconden na het starten van de bootloader wordt verzonden, schakelt deze over naar de commando-modus. Er moet dan een `u` worden verzonden om het uploaden in de bootloader te activeren; in HTerm moet op de knop *Select File* worden geklikt. Selecteer het bestand *neorv32_exe.bin* in de map *~/neorv32/sw/example/hello_world* zoals in **figuur 13** en laad het. Als alles klaar is, retourneert de bootloader



Figuur 11. Een geopend HTerm-venster.



Figuur 12. De NEORV32-bootloader.



Figuur 13. Dialoogvenster voor het uploaden van neorv32_exe.bin.

```
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:>
```

Figuur 14. Uploaden geslaagd.

```
<< NEORV32 Bootloader >>\r\n
\r\n
BLDV: Oct  5 2021\r\n
HWV:  0x01060108\r\n
CLK:  0x0112a880\r\n
MISA: 0x40801105\r\n
CPU:  0x00000041\r\n
SOC:  0x007f000d\r\n
IMEM: 0x00010000 bytes @0x00000000\r\n
DMEM: 0x00010000 bytes @0x80000000\r\n
\r\n
Autoboot in 8s. Press key to abort.\r\n
Aborted.\r\n
\r\n
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:> e\r\n
Booting...\r\n
\r\n
\r\n
                                                                                      ##                                \r\n
                                                                                      ##         ##   ##   ##           \r\n
 ##     ##   #########   ########    ########   ##      ##   ########    ########     ##       ################         \r\n
####    ##  ##          ##      ##  ##      ##  ##      ##  ##      ##  ##      ##    ##     ####            ####       \r\n
## ##   ##  ##          ##      ##  ##      ##  ##      ##          ##         ##     ##       ##   ######   ##         \r\n
##  ##  ##  #########   ##      ##  #########   ##      ##      #####        ##       ##     ####   ######   ####       \r\n
##   ## ##  ##          ##      ##  ##    ##     ##    ##           ##     ##         ##       ##   ######   ##         \r\n
##    ####  ##          ##      ##  ##     ##     ##  ##    ##      ##   ##           ##     ####            ####       \r\n
##     ##    #########   ########   ##      ##      ##       ########   ##########    ##       ################         \r\n
                                                                                      ##         ##   ##   ##           \r\n
                                                                                      ##                                \r\n

Hello world! :)\r\n
```

Figuur 15. 'Hello World' van de NEORV32.

RISC-V-naamgevingsconventie

RISC-V is een verzamelnaam voor de verschillende architectuurvarianten. Onze Elektor-collega Stuart Cording schreef een informatief artikel ("Wat is RISC-V", Elektor juli/augustus 2021, [2]) waarin de details nader worden toegelicht. RISC-V beschrijft een ISA waarin de processoren zijn ingedeeld in 32-, 64- of 128-bit-versies. Een 32-bit processor heeft een naam die begint met RV32 voor 32 bit; RV64 betreft dus een 64-bit processor. Daarnaast komt er na RV32 of RV64 een aantal letters (van A tot Z) die aangeven welke commando's en extensies de processor aankan; hun betekenis kan worden opgezocht in de actuele RISC-V-specificatie [3]. De prestaties van de processoren verschillen afhankelijk van de ondersteunde opdrachten en extensies.

een `OK` zoals in **figuur 14** en kan het programma worden uitgevoerd door `e` in te voeren.
Het resultaat is te zien in **figuur 15**. Het programma is niet opgeslagen in de SPI-flash, maar in het RAM-gebaseerde "ROM"-geheugen van de NEORV32. Dit verlengt de levensduur van de SPI-flash door het aantal schrijfcycli te verminderen. Het nadeel van deze configuratie is dat elke keer dat de NEORV32 wordt opgestart, het programma opnieuw moet worden geladen. Als het programma automatisch moet worden geladen, moet de bootloader het uploaden naar de SPI-flash. Naast de basisdemo "Hello World" zijn er nog andere voorbeelden voorhanden, waaronder een complete FreeRTOS, die al in Elektor is beschreven [15]. Het is de moeite waard om in de map *~/neorv32/sw/example* te kijken, waar verschillende andere voorbeelden in separate mappen te vinden zijn.

Een nieuwe iCE40UP5K FPGA-omgeving

Aan de hand van het iCEBreaker-board laten we zien hoe de NEORV32 kan worden aangepast aan andere iCE40up5k-boards. De features van het SoC zelf zijn hier niet gewijzigd, maar de pintoewijzing op de FPGA is aangepast aan het iCEBreaker-board zodat een passende een geschikte bitstream wordt gegenereerd.
Om dit te doen, moet de Makefile in *~/neorv32/setup/osflow* worden aangepast. Open het bestand in een teksteditor naar keuze; het nieuwe board wordt als target toegevoegd. Voor het iCEBreaker-bord schrijven we het volgende in regel 72:

```
iCEBreaker:
$(MAKE) \
BITSTREAM=neorv32_$(BOARD)_$(DESIGN).bit \
NEORV32_MEM_SRC="devices/ice40/neorv32_imem.ice40up_
   spram.vhd devices/ice40/neorv32_dmem.ice40up_
   spram.vhd" \
run
```

Dit zorgt ervoor dat naar het board wordt verwezen in de primaire Makefile. Ook in *~/neorv32/setup/osflow/boards* is een *iCEBreaker.mk*-bestand met de volgende inhoud nodig:

```
.PHONY: all

all: bit
echo "! Built $(IMPL) for $(BOARD)"
```

De Makefiles worden zo voorbereid, maar twee VHDL-bestanden ontbreken nog. In *~/neorv32/setup/osflow/board_tops/* moeten de bestanden *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* en *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* worden gegenereerd. Aangezien hun inhoud bijna identiek is aan die van de bestanden *neorv32_UPduino_BoardTop_UP5KDemo.vhd* en *neorv32_UPduino_BoardTop_MinimalBoot.vhd*, kunnen ze gewoon gekopieerd en hernoemd worden. Dit moet worden gedaan in een terminal met:

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_
   UPduino_BoardTop_MinimalBoot.vhd
~/neorv32/setups/osflow/board_tops/neorv32_
   iCEBreaker_BoardTop_MinimalBoot.vhd
```

en

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_
   UPduino_BoardTop_UP5KDemo.vhd
~/neorv32/setups/osflow/board_tops/neorv32_
   iCEBreaker_BoardTop_UP5KDemo.vhd
```

Er moeten een paar wijzigingen worden aangebracht in de twee bestanden *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* en *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd*. In het bestand *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* moet regel 42 worden gewijzigd in `entity neorv32_iCEBreaker_BoardTop_UP5KDemo is` en regel 68 in `architecture neorv32_iCEBreaker_BoardTop_UP5KDemo_rtl of neorv32_iCEBreaker_BoardTop_UP5KDemo is` worden gewijzigd. In het bestand *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* wordt regel 42 veranderd in `entity neorv32_iCEBreaker_BoardTop_MinimalBoot is` en regel 54 in `architecture neorv32_iCEBreaker_BoardTop_MinimalBoot_rtl of neorv32_iCEBreaker_BoardTop_MinimalBoot is`. De laatste stap is dat het *iCEBreaker.pcf* constraints-bestand in *~/neorv32/setups/osflow/constraints/* wordt geplaatst. De inhoud van het bestand is te zien in **listing 1**. *ICEBreaker.pcf* definieert welke functie naar welke pin van de FPGA moet worden geleid. Dit integreert ook rechtstreeks de USB/serieel-converter op het iCEBreaker-bord, en leidt de knoppen en LED's naar de I/O-pinnen van de NEORV32. Wat nog ontbreekt is een resetknop. Ook al is die gedefinieerd in het constraints-bestand met beperkingen, wordt er niet naar zijn functie verwezen.
Na alle noodzakelijke aanpassingen kunnen we een bitstream genereren, net als voor het UPduino-board. In een terminal voeren we `cd ~/neorv32/setups/osflow` in om naar de *osflow*-map te gaan. Met `make BOARD=iCEBeaker UP5KDemo` kunnen we het make-proces starten. Om de bitstreams naar de iCEBreaker te sturen (net als voor de UPduino) gebruiken we `iceprog ~/neorv32/setups/osflow/neorv32_iCEBreaker_UP5KDemo.bit`.
Het laden van de software voor de NEORV32 wordt ook verzorgd door de geïntegreerde bootloader. Bij gebruik van dit board hebben we geen externe USB/serieel-converter nodig; in plaats daarvan gebruiken we het tweede kanaal van de converter die op het iCEBreaker-bord is geïntegreerd.

Een resetknop voor de NEORV32

Om de NEORV32 opnieuw op te starten, moet de voeding even worden losgekoppeld en vervolgens weer worden aangesloten. Dat wordt op den duur vervelend; aangezien de iCEBreaker genoeg knoppen heeft, kan een ervan als reset worden gebruikt. We kunnen de uButton

Listing 1. iCEBreaker.pcf

```
#UART (uart0)
ldc_set_location -site {9} [get_ports uart_txd_o]
ldc_set_location -site {6} [get_ports uart_rxd_i]

#SPI - on-board flash
ldc_set_location -site {14} [get_ports flash_sdo_o]
ldc_set_location -site {15} [get_ports flash_sck_o]
ldc_set_location -site {16} [get_ports flash_csn_o]
ldc_set_location -site {17} [get_ports flash_sdi_i]

#SPI - user port
ldc_set_location -site {43} [get_ports spi_sdo_o]
ldc_set_location -site {38} [get_ports spi_sck_o]
ldc_set_location -site {34} [get_ports spi_csn_o]
ldc_set_location -site {31} [get_ports spi_sdi_i]

#TWI
ldc_set_location -site {2} [get_ports twi_sda_io]
ldc_set_location -site {4} [get_ports twi_scl_io]

#GPIO - input
ldc_set_location -site {18} [get_ports {gpio_i[0]}]
ldc_set_location -site {19} [get_ports {gpio_i[1]}]
ldc_set_location -site {20} [get_ports {gpio_i[2]}]
ldc_set_location -site {28} [get_ports {gpio_i[3]}]

#GPIO - output
ldc_set_location -site {25} [get_ports {gpio_o[0]}]
ldc_set_location -site {26} [get_ports {gpio_o[1]}]
ldc_set_location -site {27} [get_ports {gpio_o[2]}]
ldc_set_location -site {23} [get_ports {gpio_o[3]}]

#RGB power LED
ldc_set_location -site {39} [get_ports {pwm_o[0]}]
ldc_set_location -site {40} [get_ports {pwm_o[1]}]
ldc_set_location -site {41} [get_ports {pwm_o[2]}]

#Reset
ldc_set_location -site {10} [get_ports
   {user_reset_btn}]
```

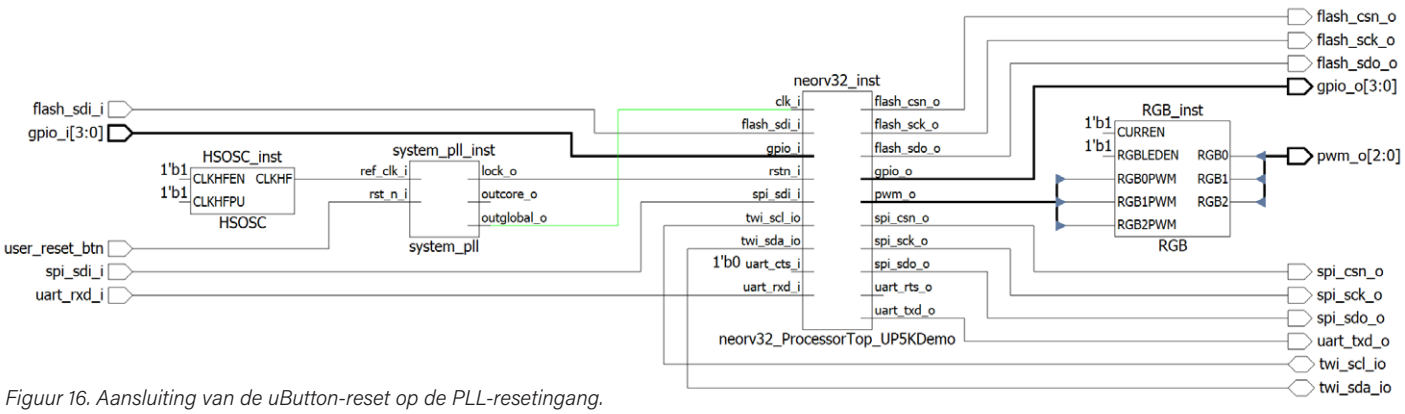
gebruiken in de buurt van de micro-USB-aansluiting op het board, die is aangesloten op pin 10 van de FPGA.

De NEORV32 heeft een interne reset-ingang die is aangesloten op de "lock"-uitgang van de systeem-PLL. Wanneer de PLL uit vergrendeling gaat, wordt de NEORV32 gereset. De fraaiste oplossing zou zijn om het gebruikers-resetsignaal van de knop samen met het vergrendelings-signaal door een poort te sturen, zodat een van beide de NEORV32 zou resetten. De snelle (en minder fraaie) oplossing is om gewoon de reset-ingang van de PLL te gebruiken. **Figuur 16** toont de aansluiting van de uButton op deze ingang. Wanneer nu de PLL wordt gereset, wordt ook de NEORV32 gereset, aangezien het *lock_o*-signaal laag wordt wanneer de PLL wordt gereset.

Om deze wijziging in het UP5K-demoproject op te nemen, is het nodig om één regel in te voegen en een andere regel te wijzigen in het bestand *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* (in de map *~/neorv32/setups/board_tops*). Voeg na regel 44 de nieuwe regel `user_reset_btn : in std_ulogic;` in. Regel 130 moet dan worden gewijzigd in `RESETB => user_reset_btn,`. Pas op: er zal een syntaxfout optreden als u de komma aan het einde vergeet. De uButton is nu geconfigureerd als reset. Om alle wijzigingen te effectueren moet ten slotte een nieuwe bitstream worden gegenereerd in de UP5K-demo en in de iCEBreaker-FPGA worden geladen.

Vooruitblik

We kunnen gemakkelijk een heel boek schrijven over RISC-V, FPGA en de NEORV32. De iCE40UP5K is een goedkope keuze voor beginners, ware daar niet het probleem met de leverbaarheid. De twee boards die in het artikel worden gebruikt, zijn niet de enige die deze FPGA



Figuur 16. Aansluiting van de uButton-reset op de PLL-resetingang.

gebruiken. Er bestaat op zijn minst een handvol andere platforms. Van de Raspberry Pi-add-on tot de complete handheld-console zijn er veel opties beschikbaar. Ook de verschillende tools en projecten die van toepassing zijn op deze FPGA zijn het onderzoeken waard. Voorbeelden zijn LiteX [16], waarmee u uw eigen SoC kunt samenstellen als in een bouwpakket en dat niet noodzakelijkerwijs beperkt is tot RISC-V, de RISCboy [17] van Luke Wren die een Gameboy-achtig systeem in de FPGA maakt, en de OK-ice40-PRO gamepad-console [18].
Als de problemen met de leverbaarheid tot het verleden behoren, komen er zeker nog meer projecten en ideeën voor de iCE40UP5K. Een kleine game-console rond een iCE40UP5K in combinatie met een Raspberry Pi RP2040 lijkt al in de pijplijn te zitten [19].

210175-03

Een bijdrage van

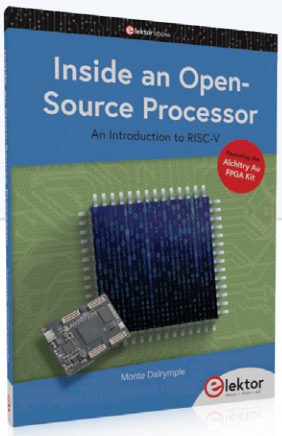
Tekst en illustraties: **Mathias Claußen**
Redactie: **Jens Nickel**
Vertaling: **Eric Bogers**
Layout: **Harmen Heida**

Vragen of opmerkingen?

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

GERELATEERDE PRODUCTEN

- **Alchitry Cu FPGA Development Board (Lattice iCE40 HX) (SKU 19640)**
  www.elektor.nl/19640
- **CH340 USB to TTL Converter UART Module CH340G (3.3 V/5.5 V) (SKU 19151)**
  www.elektor.nl/19151
- **M. Dalrymple, *Inside an Open-Source Processor* (Elektor 2021) (SKU 19826)**
  www.elektor.nl/19826



WEBLINKS

[1] M. Oßmann, "Het SCCC-project (1)", Elektor maart/april 2019: www.elektormagazine.nl/magazine/elektor-90/42537
[2] S. Cording, "Wat is RISC-V", Elektor juli/augustus 2021: www.elektormagazine.nl/magazine/elektor-181/59809
[3] RISC-V-specificatie: http://riscv.org/technical/specifications/
[4] NEORV32 GitHub-repository: http://github.com/stnolting/neorv32/
[5] Lattice iCE40UltraPlus productpagina: www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus
[6] iCEBreaker GitHub-repository: http://github.com/icebreaker-fpga/icebreaker
[7] UPduino GitHub-repository: http://github.com/tinyvision-ai-inc/UPduino-v3.0
[8] Lattice Radiant: www.latticesemi.com/LatticeRadiant
[9] YosysHQ oss-cad-suite-build : http://github.com/YosysHQ/oss-cad-suite-build
[10] Opstellen van een toolchain voor de Kendryte K210, Elektor Labs: www.elektormagazine.nl/labs/setup-a-toolchain-for-the-kendryte-k210-1
[11] Linux-flavored Snickerdoodles with Zynq, ElektorTV: www.youtube.com/watch?v=EE4yYZ-FEoQ
[12] YosysHQ oss-cad-suite-build releases: http://github.com/YosysHQ/oss-cad-suite-build/releases
[13] HTerm: www.der-hammer.info/
[14] NEORV32 - Build the Toolchain from scratch: http://stnolting.github.io/neorv32/ug/#_building_the_toolchain_from_scratch
[15] W. Gay, "Multitasking met de ESP32", Elektor januari/februari 2020: www.elektormagazine.nl/magazine/elektor-144/57032
[16] LiteX: http://github.com/enjoy-digital/litex
[17] RISCBoy : http://github.com/Wren6991/RISCBoy
[18] OK-iCE40Pro Handheld: http://github.com/WiFiBoy/OK-iCE40Pro
[19] PicoStation3D: http://github.com/Wren6991/PicoStation3D
[20] Nolting S., The NEORV32 RISC-V-Processor, GitHub-repository 2020: http://raw.githubusercontent.com/stnolting/neorv32/master/docs/figures/neorv32_processor.png