Simulador complet d'un processador de codi obert

Oriol Casas Brugués

Resum—Aquest treball implementa un model de UART bàsic com a perifèric sèrie d'un simulador de processador RISC-V. Aquesta relativament moderna arquitectura (ISA) de codi obert representa un canvi de paradigma per al disseny personalitzat de processadors, obrint un ventall de possibilitats per a diverses aplicacions i especificacions. En el cas d'aquest treball, s'ha optat per integrar un nou mòdul UART com a perifèric al simulador de processador ja existent, per tal de que serveixi com a base per a futurs augments de les prestacions de la comunicació sèrie amb altres dispositius.

Paraules clau— *RISC-V, UART, Comuniació sèrie, SystemC, Emuladors Virtuals*.

Abstract—This work presents a basic UART model as a serial port for a simulation of a RISC-V microcontroller. This relatively new architecture (ISA), which is open-source, represents a paradigm shift towards making custom processor design available to everyone, and also offering an endless array of possibilities for several applications and specifications. In this case, a new UART module has been integrated as a peripheral to the existing virtual microcontroller, so that it may serve as a reference for future upgraded solutions of the serial communication with other devices.

Index Terms— *RISC-V, UART, Serial communications, SystemC, Machine Emulators*

1 Introducció - Context del treball

La proposta consisteix en desenvolupar un model de UART 16450 i integrar-lo com a perifèric en un simulador d'un processador RISC-V. El processador forma part d'un projecte ja existent [1], que està escrit en SystemC i dissenyat seguint TLM-2 (Transaction-Level Modeling), en el qual es facilita la expansió i l'addició de nous mòduls.

A través de la simulació d'un procesador RISC-V des de l'ordinador personal, es pot emular una màquina virtual (amb la possibilitat d'incloure-hi un sistema operatiu) que executi programes dissenyats per a un microxip. D'aquesta manera, s'obre un ventall de possibilitats per a futures aplicacions, sobretot fonamentades en la flexibilitat de fer modificacions i optimitzacions en un simulador (software), en comptes d'un microxip (hardware).

En el cas d'aquest treball, s'ha optat per afegir un perifèric al processador, en concret, el model de UART 16450. Aquesta és una versió simple i fonamental per a implementar la interfície de la comunicació en sèrie UART (veure apartat 2.2). D'aquesta forma, es dotaria al processador de la possibilitat de comunicar-se amb dispositius externs, ja sigui de forma virtual (per consola en el mateix host) o real (a través d'un port sèrie cap a un altre dispositiu).

En aquest sentit, l'objectiu principal d'aquest treball consisteix en implementar i integrar un model de perifèric basat en la UART 16450 que permeti realitzar les operacions fonamentals de lectura i escriptura, en el simulador d'un processador RISC-V.

- *E-mail de contacte: oriol.casasb@e-campus.uab.cat*
- *Menció en Enginyeria de Computadors*
- *Treball tutoritzat per: Màrius Montón Macián (Departament de Microelectrònica i Sistemes Electrònics)*
- *Curs 2020/21*

2 → CONCEPTES PREVIS

2.1 RISC-V

RISC-V és un repertori d'instruccions ISA de codi obert que va néixer com un projecte de la Universitat de California-Berkeley al 2010. El fet que sigui de lliure accés ha estat clau en el seu èxit i la seva expansió. Molts desenvolupadors, sobretot petites startups, institucions acadèmiques i membres de la comunitat, prefereixen aquesta opció gratuïta, en comptes d'assumir la costosa llicència que es requereix per a implementar un processador amb un altre ISA (com per exemple, ARM o x86)[2].

Per a poder executar algun programa en un processador RISC-V, es necessita tenir un processador RISC-V real o un entorn de treball preparat per a la seva execució. S'han publicat varis processadors des de que RISC-V es va crear[3], dels quals en destaca Qemu[4].

2.2 UART 16450

La UART 16450 correspon a un tipus de UART senzilla, ja obsoleta, que representa la base fonamental del funcionament de la resta de UARTs modernes (com la 16550A).

Quan el processador rep o envia un byte cap al dispositiu sèrie, es genera una interrupció on el processador gestiona aquest byte de dades, cap a memòria (si rep la informació) o cap al port sèrie (si transmet la informació). En el cas de la UART 16450, el buffer de dades és d'un sol byte. Això representa una interrupció cada vegada que s'envia 1 byte d'informació. Per a velocitats de transferència relativament baixes, aquest fenòmen no representa cap problema. Però per a velocitats de transferència més altes, es podrien perdre dades que el processador no té temps de gestionar (*overrun*)[5].

La versió posterior UART 16550A, que és la més popularment utilitzada, inclou un FIFO (*first-in first-out*) buffer de 16 bytes. Això significa que el processador pot enviar fins a 16 bytes al mateix temps abans de generar una interrupció. Aquest avantatge representa una millora considerable sobre la versió obsoleta, però aquest aspecte no s'ha considerat rellevant per a la implementació d'aquest treball.

3. → IMPLEMENTACIÓ SOFTWARE

3.1. MEMORY-MAPPED I/O

Un mètode per a implementar entrada/sortida entre un perifèric i la CPU d'un ordinador és a través d'E/S mapejada en memòria (*Memory-Mapped I/O*). Això significa que es pot utilitzar l'espai d'adreces tant per a accedir a memòria principal com a dispositius d'entrada/sortida. La memòria i els registres de cadascun dels perifèrics que es comuniquin amb el host estan associats a unes adreces de memòria específiques. Així, quan la CPU realitza operacions d'escriptura o lectura, es pot estar referint a una part de la memòria física (RAM, per exemple) o a una part de la memòria del perifèric.
Per tant, per a simular els registres de la UART, cal reservar un espai d'adreces per a aquest perifèric. En aquest simulador, això es realitza des del controlador del bus (*Bus Controller*). Aquest mòdul és el responsable de la comunicació entre tots els mòduls. Per això, s'ha definit una nova adreça específica per als registres de la UART i s'ha implementat que quan s'enviïn o es rebin dades pel bus d'aquesta adreça, el bus es comuniqui amb el mòdul UART.

3.2. TLM SOCKETS

La interfície entre els diferents mòduls en aquest simulador es modela mitjançant l'ús de sockets, seguint *Transaction Level Modelling* (TLM). Aquest model utilitza transaccions per a simplificar la comunicació entre els mòduls de forma més abstracta. La implementació d'aquesta transacció es troba a la funció *b_transport*, en la qual es realitza l'accés des d'un Initatior socket (per exemple, el socket de la UART) cap a un Target Socket (per exemple, el bus de dades), amb les dades de lectura o escriptura, i un temps de resposta (que idealment es pot especificar a 0).
Així doncs, s'ha definit un nou socket per al mòdul de la UART, ja que SystemC ja inclou els estàndards per a TLM2, i s'ha afegit a la implementació de *b_transport*.

3.3. MODE DE FUNCIONAMENT PER POLLING

A diferència del funcionament per interrupció més comú en la versió de la UART 16450, el mode de funcionament seleccionat per a aquest cas és per *polling*. Això significa que cal comprovar l'estat del buffer de dades de transmissió mitjançant l'ús d'un registre de control.
Per aquest motiu, s'ha implementat un procediment en el qual un thread anirà preguntant en un bucle

infinit si hi ha dades en el buffer per a transmetre. Si és així, i el registre de control està activat, s'enviarà les dades cap a l'exterior, simulant el funcionament d'un perifèric (en el cas d'aquest treball, s'imprimeix per la consola).

4. → TESTS I RESULTATS

Un cop compilada la implementació software del nou mòdul, s'ha procedit amb el disseny de diversos tests per comprovar l'execució i la fiabilitat de com es reben i es transmeten les dades pel perifèric.

També cal tenir en compte el tipus i format de les dades per a la integritat de la comunicació. S'utilitzen registres de 32 bits, ja que aquesta és la longitud de les paraules del simulador (arquitectura 32bits). Però pel funcionament de la UART els registres utilitzats són de 8 bits, pel que les dades efectives només es guardaràn en el primer byte de cada registre, i la resta de bytes (els 3 restants) s'utilitzaran com a padding per a la següent posició de memòria corresponent.

La figura següent (veure Figura 1) mostra els resultats de l'execució del nou thread, seguint el mode de funcionament per polling. En el test d'exemple, s'escriuen 4 caràcters (un per un) a la posició de memòria del buffer de la UART (0x40008000 en aquest cas). El thread escriu aquests caràcters per pantalla quan rep la senyal de que s'han rebut les dades a transmetre en el buffer (mitjançant un event de SystemC).

```
oriol@oriol-pc:~/RISC-V-TLM$ ./RISCV_TLM tests/C/uart2/uart2.hex
LogLevel set to 0
02 extended address 0x10000
03 PC set to 0x10084
Get DMI_PTR
START TRANSMISSION
Data in address: 40008000 to be written: 54
END TRANSMISSION
1. Data written:T
START TRANSMISSION
Data in address: 40008000 to be written: 65
END TRANSMISSION
2. Data written:e
START TRANSMISSION
Data in address: 40008000 to be written: 73
END TRANSMISSION
3. Data written:s
START TRANSMISSION
Data in address: 40008000 to be written: 74
END TRANSMISSION
4. Data written:t

ECALL Instruction called, stopping simulation
```

Figura 1: Execució de l'exemple per a transmetre el missatge: "T E S T".

5. → CONCLUSIONS

Aquest treball implementa un model de UART fonamental per a realitzar operacions bàsiques de lectura i escriptura, en un simulador RISC-V.
Aquest treball també investiga sobre la potencialitat de l'arquitectura RISC-V per al futur disseny de processadors personalitzats a l'abast de tothom. En el procediment, s'argumenten els criteris de disseny del perifèric, així com la metodologia emprada per al desenvolupament del projecte.
I finalment, aquest treball també pretén servir com a base per a futures solucions de comunicació en sèrie per al simulador del processador en qüestió.

REFERÈNCIES

[1] Màrius Montón. 2020. *A RISC-V SystemC-TLM simulator.* CARRV 2020. http://mariusmonton.com/wp-uploads/2020/05/CARRV2020_paper_7_Monton.pdf

[2] Anton Shilov. Octubre 2020. *Major Arm licensee adopts royalty-free RISC-V core for next-gen processors.* Tech Radar. https://www.techradar.com/news/major-arm-licensee-adopts-royalty-free-risc-v-core-for-next-gen-processors

[3] RISC-V International 2020. *RISC-V Exchange: Available Software.* https://riscv.org/exchange/software/

[4] QEMU, The Fast Processor Emulator. 2020. *Official QEMU mirror* https://github.com/qemu/qemu

[5] The Linux Documentation Project. Febrer 2011. *Serial HOWTO.* https://tldp.org/HOWTO/Serial-HOWTO-18.html

APÈNDIX

El codi font del projecte és de codi obert i publicat en el següent enllaç:

https://github.com/oriolcasas/RISC-V-TLM