Informe Pràctica 3 Sessió 4: Controlador de Byte

Màquina d'estats

La unitat de control a nivell de byte del mestre I2C es regeix principalment per una màquina d'estats. Aquesta consta de vuit estats, els quals estan recopilats a la Taula 1. Donades les especificacions d'operació del mestre I2C, és evident que la màquina d'estats característica del mestre és del tipus Mealy, ja que les sortides depenen dels senyals d'entrada i de l'estat en el que es troba la màquina (vegeu diagrama d'estats a la Figura 19).

Tanmateix, el sistema empra un comptador extern de 3 bits per controlar la sincronia amb la unitat de control a nivell de bit. Cada cop que es fa una operació d'escriptura o de lectura d'un byte, el comptador (de tipus *count-down*) compta fins a 7 vegades, una per cada bit transmès o llegit, mentre que el primer bit es gestiona amb els estats READ_A i WRITE_A. Fent ús d'aquest comptador s'optimitza de manera efectiva la quantitat d'estats necessaris per a la màquina, passant de 20 estats (un per a cada bit per a les dues operacions, lectura i escriptura, més els estats d'espera, inici, aturada i *acknowledge*) a 8 estats. La unitat de control realitza les següents operacions:

Taula 1: Llista dels diferents estats de la màquina d'estats implementada.

Estat	Descripció
IDLE	Estat d'espera. El mestre està a preparat per executar una escriptura, una lectura, aturar la transferència d'informació si prèviament ho estava fent, o mantenir-se en aquest estat si no hi ha cap comanda a executar.
START	Estat en el que es carrega el comptador extern, es carreguen les dades al registre de desplaçament i s'envia la comanda a executar.
READ_A	Estat en el que s'habilita els desplaçaments al registre de desplaçament, la recepció de bits d'Acknowledge i s'envia la comanda de lectura.
READ	Estat en el que es llegeixen els últims 7 bits d'entrada mitjançant desplaçaments al registre de desplaçament. Quan s'acaba la lectura, s'envia la comanda d'escriptura per a l'enviament del bit d'*Acknowledge*.
WRITE_A	Estat en el que carreguem al registre de desplaçament les dades a escriure, enviem la comanda d'escriptura per als següents 7 bits i habilitem els desplaçaments al registre de desplaçament per a escriure el primer bit.
WRITE	Estat en el que es van carregant en sèrie al registre de desplaçament els 7 restants.
ACK	Estat en el que, si s'està en mode de lectura, s'envia un ACK/NACK en funció de si s'han llegit correctament o no les dades, i si està en mode d'escriptura, es llegeix el bit d'ACK/NACK enviat per l'esclau en funció de

	si ha rebut correctament o no les dades. Es pot transitar o bé a l'estat de STOP, o bé a l'estat d'IDLE.
STOP	Estat en el que s'atura la comunicació via I2C.

Testbench

Taula 2: Llista dels diferents tasques/funcions del testbench i la seva funcionalitat.

Tasca	Descripció
checkErrors	Comprova el nombre d'errors que s'han donat en la simulació, concretament avalua si el valor de la variable *errors* és 0 o no i mostra per pantalla un missatge d'èxit o de fallida.
syncCheck	Comprova al següent flanc de pujada de rellotge que el valor de sortida obtingut sigui el mateix que el valor de sortida esperat (comprovació síncrona). En cas de que difereixin, incrementa la variable *errors* en 1 i mostra un missatge d'error per pantalla. En cas de que coincideixin, mostra un missatge de comprovació exitosa.
asyncCheck	Comprova després d'una quantitat de temps determinada que els valors de sortida esperat i obtingut siguin el mateix (comprovació asíncrona). En cas de que difereixin, incrementa la variable *errors* en 1 i mostra un missatge d'error per pantalla. En cas de que coincideixin, mostra un missatge d'èxit.
reset	Inicia posant el senyal de *rst_n* a nivell baix, espera un nombre determinat de cicles de rellotge amb la tasca *waitCycles(Ncycles)* i torna a posar-lo a nivell alt. El senyal és actiu per nivell baix.
waitCycles	Espera un nombre determinat de cicles de rellotge mitjançant l'execució repetida de la tasca *waitClk*.
waitClk	Afegeix un retard determinat per el define *DELAY* quan es dona un flanc de pujada de rellotge.

A més, al testbench es defineix un mòdul *delay*, el qual indica el retard entre la commutació de la sortida i la commutació de l'entrada, mitjançant un bloc *specify*.

Verificació Funcional

A la Figura 1 es mostra el resultat de la simulació pre-síntesi del testbench *tb_i2c_master_top.v*.

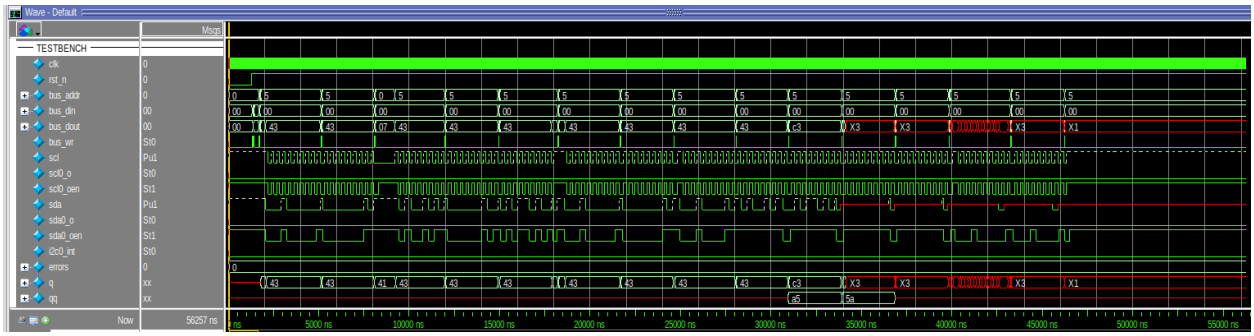

Figura 1: Diagrama d'ones del banc de proves *tb_i2c_master_top.v*.

A la Figura 3 es mostren els missatges del terminal de ModelSim al simular el testbench *tb_i2c_master_top.v*. Destacar que els missatges *# status: 36875.00 ns received xx from 3rd read address* i *# status: 39835.00 ns received xx from 4th read address* apareixen, ja que prèviament s'escriu a les adreces 01 i 02 de l'esclau, però no a les adreces 03 i 04. Per tant, les dades llegides en aquestes últimes són *xx*.

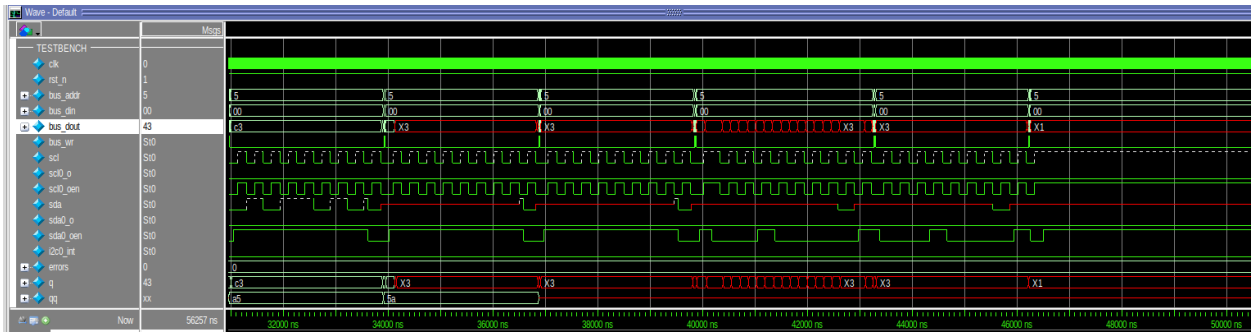

Figura 2: Ampliació de la zona d'interès. Les dades de la variable *bus_dout* en vermell són les dades obtingudes de llegir les adreces 03 i 04 (indefinides) de l'esclau I2C.

En modificar el testbench, per tal de que no es faci una lectura a aquestes adreces (línies 298 a 328 del fitxer *tb_i2c_master_top_4netlist.v*). Els missatges del terminal de ModelSim ara són els que es mostren a la Figura 4.

Figura 3: Captura terminal ModelSim amb els missatges de l'autoverificació. Fitxer *tb_i2c_master_top.v*.

```
Transcript
#
# status:    0.00 ns Testbench started
#
#
# [Info-    0.00 ns] Reset
# status: 1297.00 ns done reset
# status: 1405.00 ns programmed registers
# status: 1707.00 ns verified registers
# status: 1725.00 ns core enabled
# status: 1725.00 ns Write Access
# status: 1765.00 ns generate 'start', write cmd 20 (slave address+write)
# status: 5125.00 ns tip==0
# status: 5165.00 ns write slave memory address 01
# status: 8085.00 ns tip==0
# status: 8125.00 ns write data a5
# status: 8125.00 ns clock streching starts
# status: 9225.00 ns clock streching ends
# status: 11975.00 ns tip==0
# status: 12015.00 ns write slave memory address 02
# status: 14935.00 ns tip==0
# status: 14975.00 ns write next data 5a, generate 'stop'
# status: 18215.00 ns tip==0, busy==0
# status: 18215.00 ns Read Access
# status: 18255.00 ns generate 'start', write cmd 20 (slave address+write)
# status: 21655.00 ns tip==0
# status: 21695.00 ns write slave address 01
# status: 24615.00 ns tip==0
# status: 24655.00 ns generate 'repeated start', write cmd 21 (slave address+read)
# status: 28055.00 ns tip==0
# status: 28075.00 ns read + ack
# status: 30935.00 ns tip==0
# status: 30955.00 ns received a5
# status: 30975.00 ns read + ack
# status: 33895.00 ns tip==0
# status: 33915.00 ns received 5a
# status: 33955.00 ns generate 'start', write cmd 20 (slave address+write). Check invalid address
# status: 37335.00 ns tip==0
# status: 37375.00 ns write slave memory address 10
# status: 40295.00 ns tip==0
# status: 40295.00 ns Check for nack
# status: 40315.00 ns generate 'stop'
# status: 40335.00 ns tip==0
#
#
# status: 50337.00 ns Testbench done
```

Figura 4: Captura terminal ModelSim amb els missatges de l'autoverificació. Fitxer *tb_i2c_master_top_4netlist.v*.

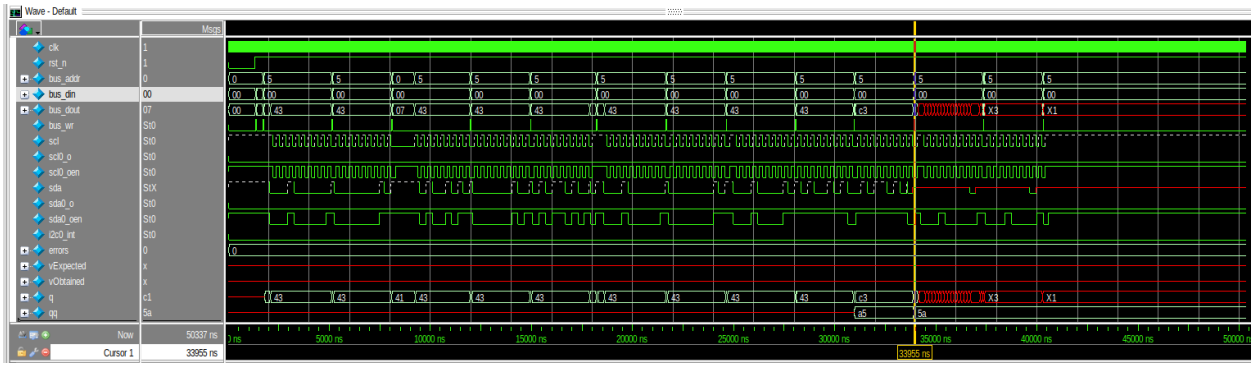

Figura 5: Diagrama d'ones del banc de proves *tb_i2c_master_top_4netlist.v*. El cursor està situat sobre el començament de la operació de comprovació d'adreça invàlida.

D'aquí en endavant, es seguirà la verificació funcional amb el banc de proves del segon cas. Primerament, es prepara el mestre I2C. Això es fa mitjançant un reset global i la programació dels registres de control, en particular, del registre de pre-escalat *prer*. A més, s'envia el senyal d'habilitació del mestre.

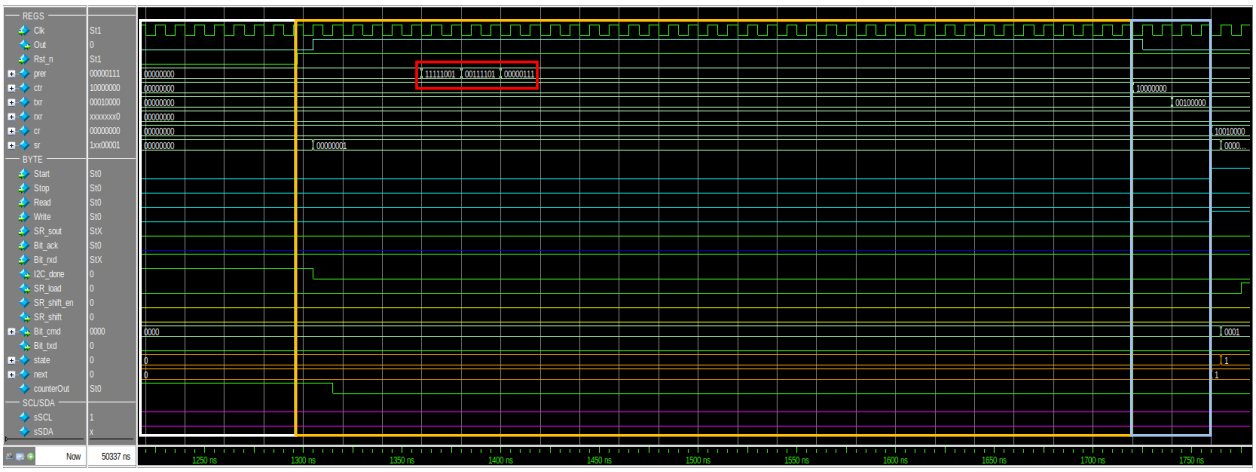

Figura 6: Diagrama d'ones del banc de proves centrat en la zona de reset inicial, programació de registres i habilitació del mestre.

A la Figura 6 es pot observar com al fer un reset (*Rst_n* actiu per nivell baix) s'esborra el contingut dels registres, com es veu al requadre blanc. Seguidament es fa la programació de registres, segons el banc de proves, s'escriu al registre *prer* els valors 8'hF9, 8'h3D i 8'h07 (a la Figura 7 mostrats dins del requadre vermell). Finalment, s'habilita el mestre I2C escrivint un 1 al bit 7 del registre *ctr* (a la figura il·lustrat amb el requadre blau cel).

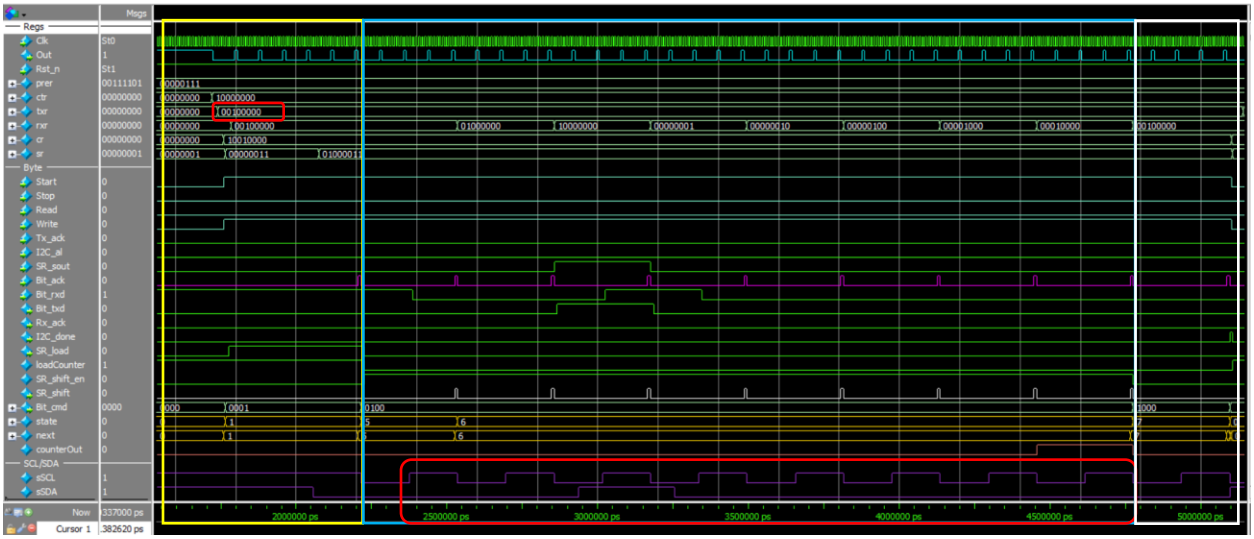

Figura 7: Diagrama d'ones del banc de proves centrat en la primera escriptura del test.

En aquesta figura podem localitzar diverses seccions:

- **Groga:** Aquí veiem com, quan es reben els senyals de Start i Write, la màquina d'estats transiciona a l'estat 1 (estat *START*). En aquest estat, podem veure com sSDA canvia a nivell baix i, uns clocks després, sSCL també baixa. El sistema marca que ha acabat la seqüència de START amb un Bit_ack.
- **Blava:** Aquí succeeix la seqüència d'escriptura. Aquí passa per un estat preliminar *READ_A* i després passa a l'estat *READ*, transicionant quan el comptador d'estats s'aixeca. Veiem com la línia SCL marca els polsos corresponents i la línia SDA transmet una seqüència de dos 0, un 1 i cinc 0 més, que es correspon amb el valor que s'havia d'escriure.
- **Blanca:** Després de l'escriptura es passa a l'estat *ACK*, el qual (en aquest cas, ja que és escriptura) espera un "acknowledge" per part del receptor i, quan el rep, marca un "i2c_done" i transiciona a l'estat *IDLE*.

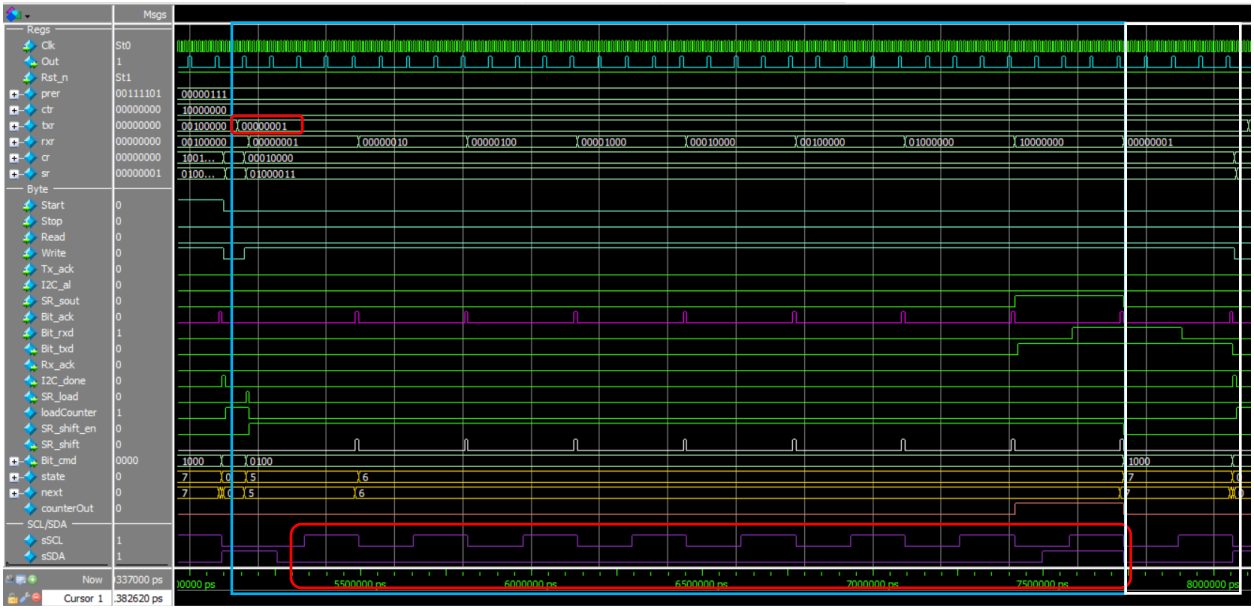

Figura 8: Diagrama d'ones del banc de proves centrat en la segona escriptura del test.

En aquesta figura es pot veure la segona escriptura del test, la qual comença sense fer un Start al principi. Es pot veure com només el senyal de Write està habilitat i com es transmet el valor esperat per la línia SDA (amb SCL marcant els polsos apropiadament). Després, transiciona a l'estat d'*ACK* fins que rep el Bit_ack, on transiciona a l'estat d'*IDLE*.

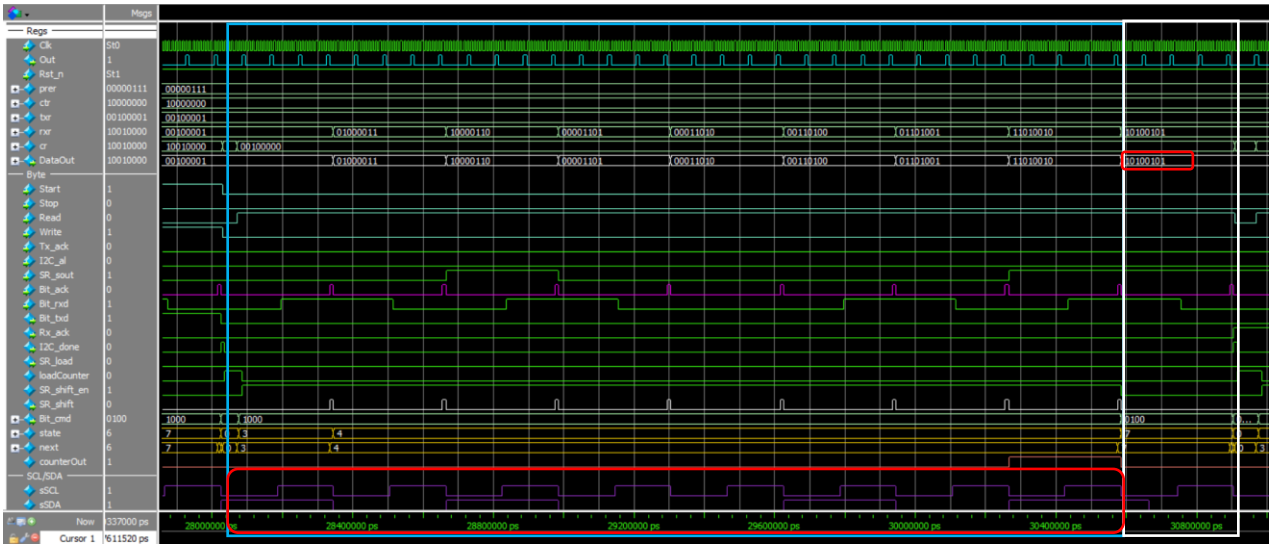

Figura 9: Diagrama d'ones del banc de proves centrat en la primera lectura del test.

Aquí tenim una situació similar a l'escriptura: primer arriba el senyal de lectura i, al llarg de 8 polsos, arriben els polsos a través del bus SDA, que van emmagatzemant-se al registre de desplaçament fins tenir el valor que toca (10100101). Després d'això, el sistema transiciona a l'estat d'*ACK*, on envia un pols per a mostrar que ha rebut el missatge.

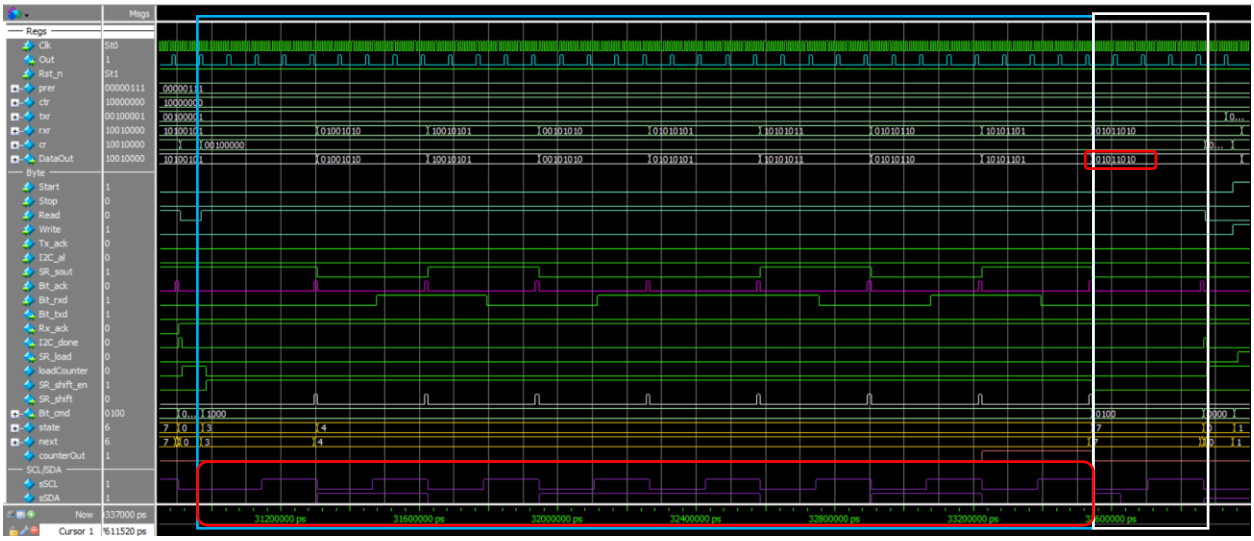

Figura 10: Diagrama d'ones del banc de proves centrat en la segona lectura del test.

En aquest cas tenim una situació gairebé idèntica a l'anterior. Veiem com el sistema rep les dades a través de SDA i les va emmagatzemant al registre de desplaçament un a un. Després d'això, transiciona a l'estat d'*ACK* i envia un acknowledge conforme ha rebut les dades.

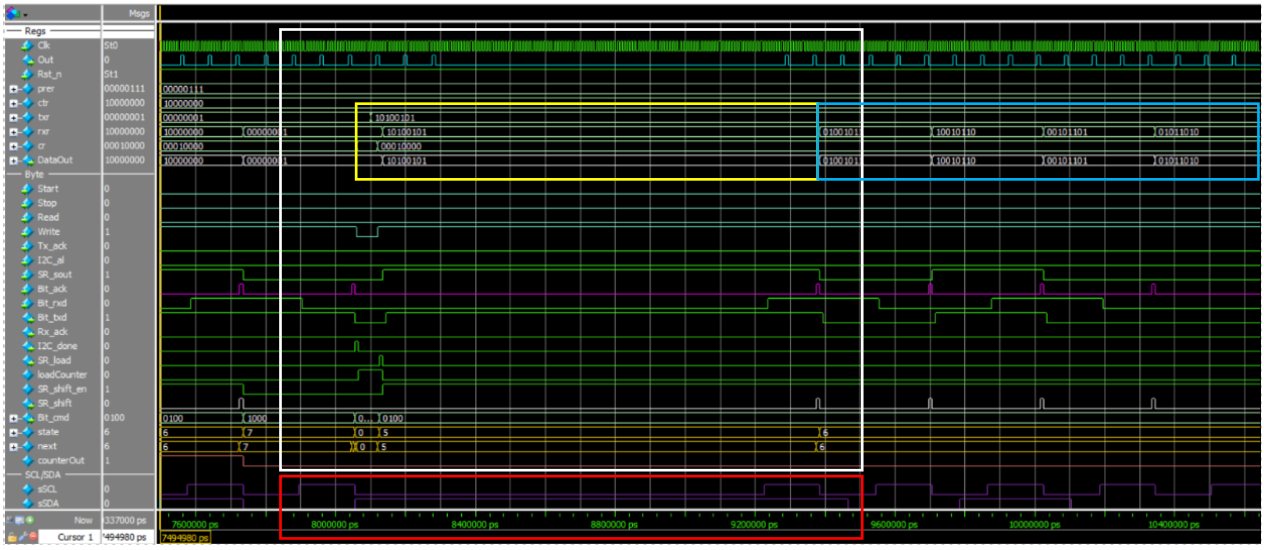

Figura 11: Diagrama d'ones del banc de proves centrat en la zona de "clock stretching".

A la Figura 11 s'il·lustra l'efecte de *clock stretching*. S'observa que quan el senyal counterOut commuta de nivell alt a nivell baix, les dades a escriure es mantenen. Es deixen de rebre Bit_ack fins que l'esclau allibera la línia SCL; aleshores es tornen a escriure dades.

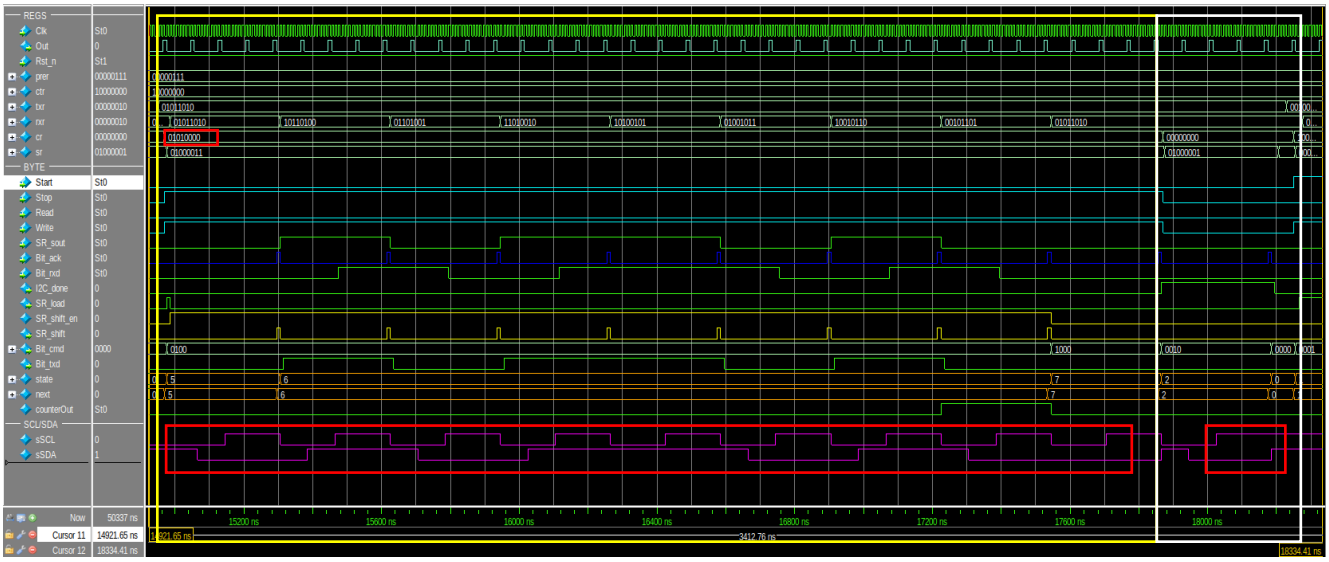

Figura 12: Diagrama d'ones del banc de proves centrat en la generació de stop després de l'escriptura.

A la figura anterior es pot veure com al commutar els senyals de Write i Stop, s'envia una ordre d'escriptura i stop (cr = 01010000). Seguidament s'escriuen primer les 8 dades i després es genera l'stop. Observar que durant l'escriptura les línies SDA i SCL commuten per a l'escriptura i quan es genera l'stop, la línia SCL commuta a nivell alt i seguidament la línia SDA commuta a nivell alt.

Síntesi en FPGA

Les Figures i mostren l'esquema RTL de la netlist generada amb el Quartus i a la Figura es mostra en detall el diagrama d'estats de la unitat de control. La Taula 3 mostra els recursos utilitzats de la FPGA.

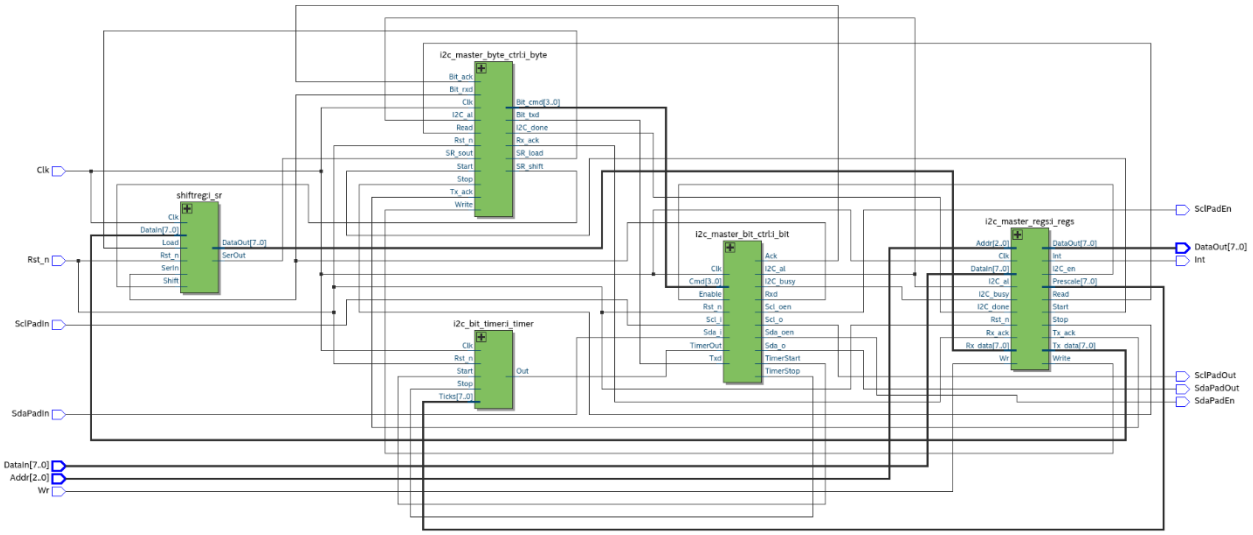

Figura 13: Esquema RTL de la netlist generada amb el Quartus.

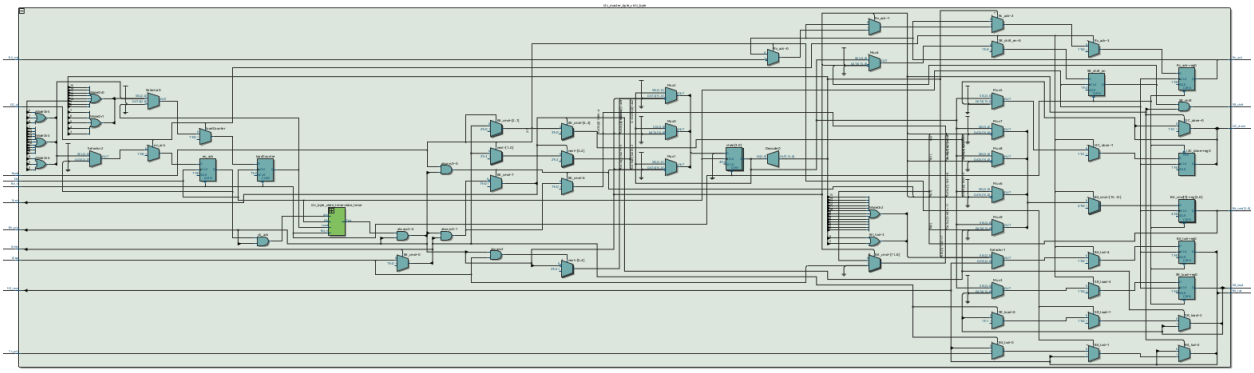

Figura 14: Esquema RTL de la instància del mòdul *i2c_master_byte_ctrl.v* (controlador de byte de la UC del mestre I2C).

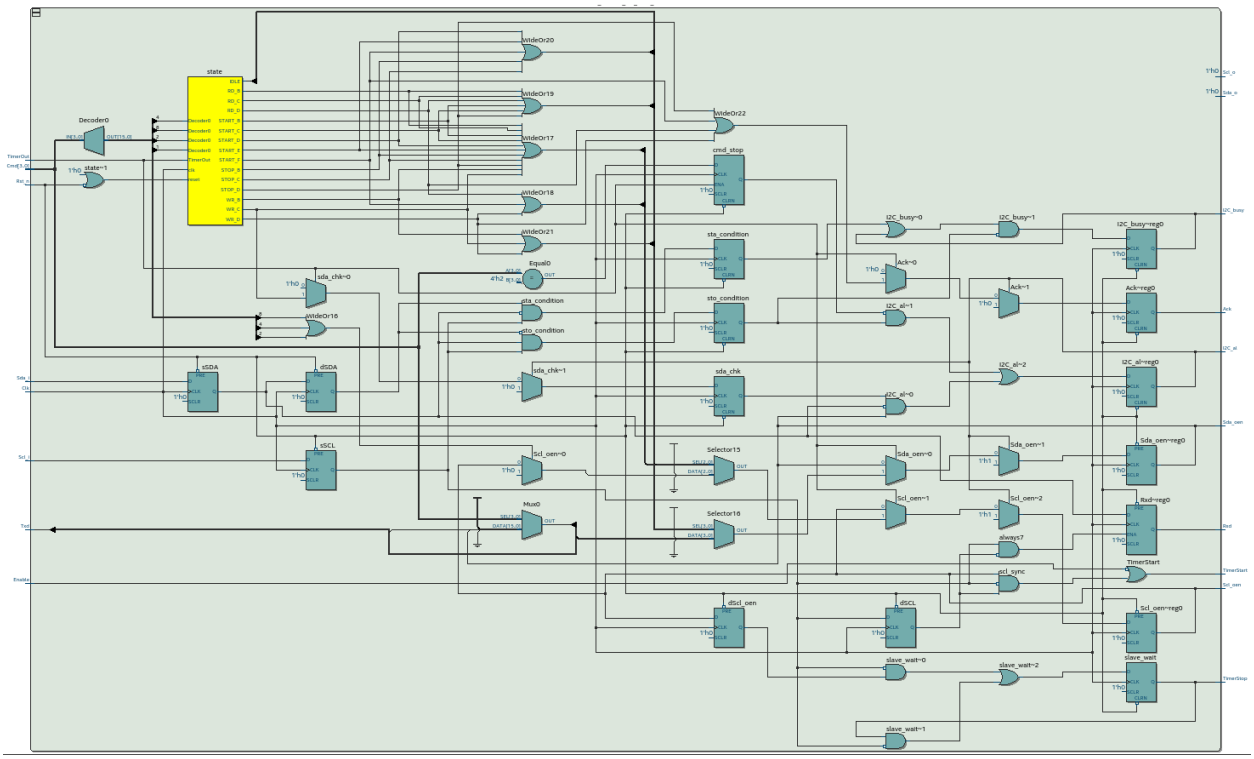

Figura 15: Esquema RTL de de la instància del mòdul *i2c_master_bit_ctrl.v* (controlador de bit de la UC del mestre I2C).

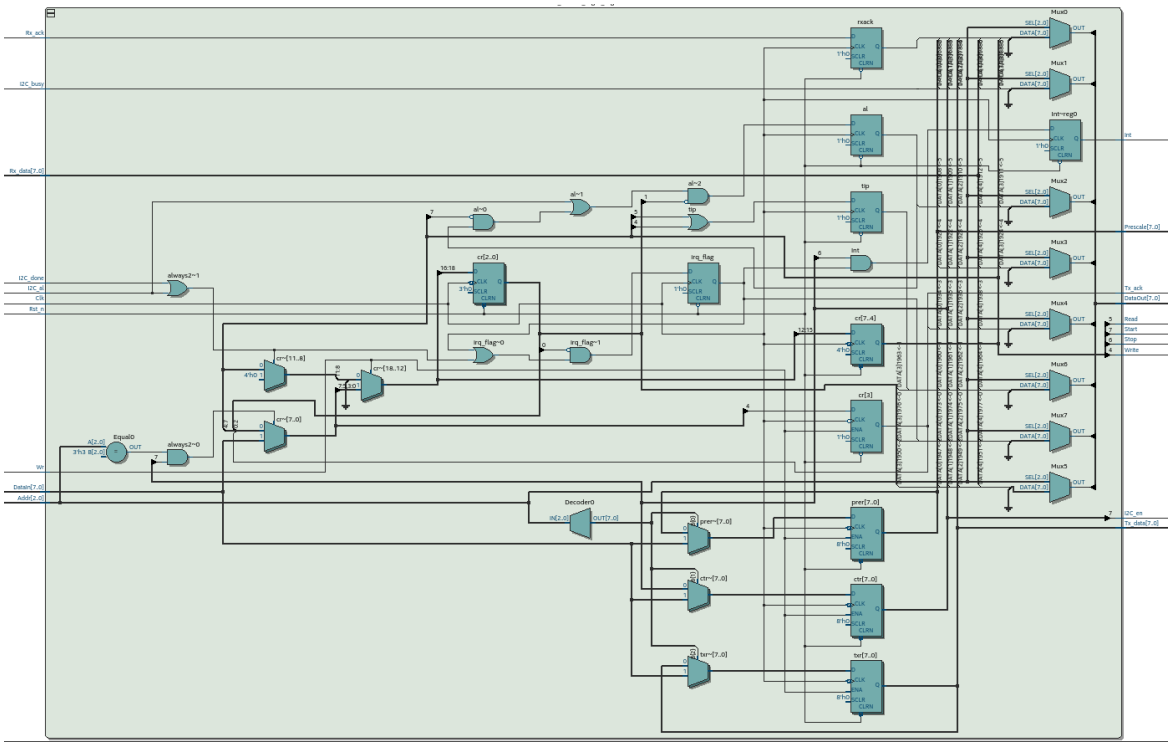

Figura 16: Esquema RTL de la instància del mòdul *i2c_master_regs.v* (registres de comanda i control del mestre I2C).

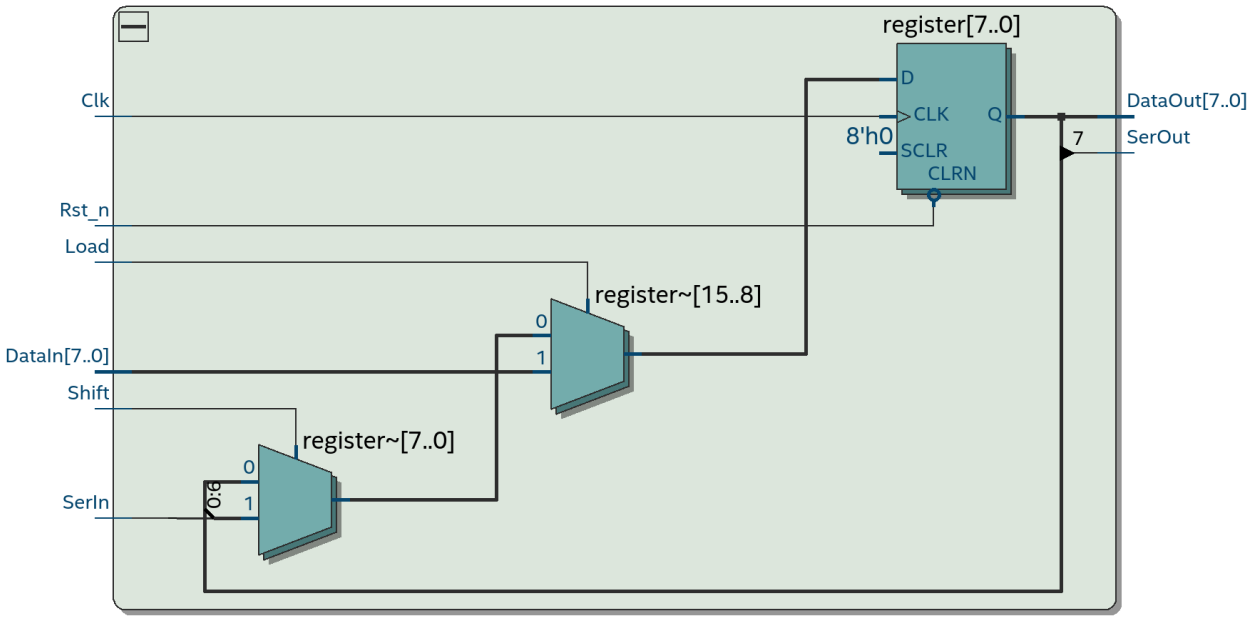

Figura 17: Esquema RTL de la instància del mòdul *shiftreg.v* (registre de desplaçament).

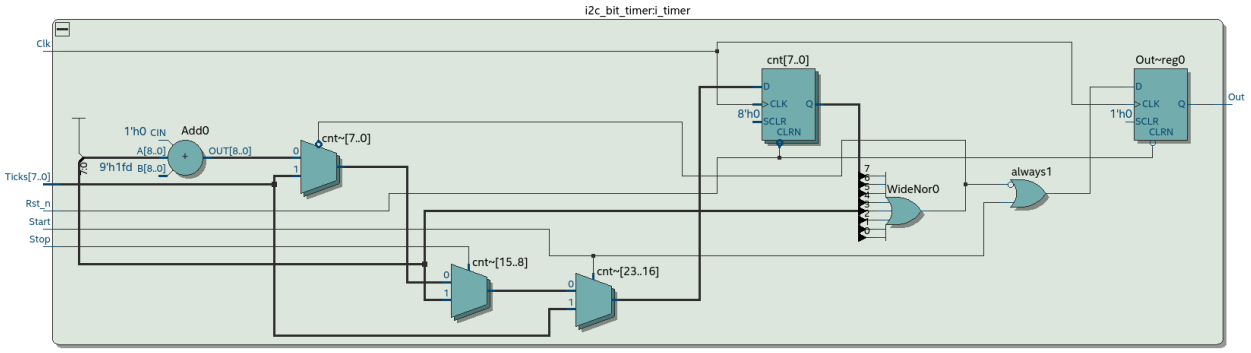

Figura 18: Esquema RTL de la instància del mòdul *i2c_master_timer.v* (timer del mestre I2C).

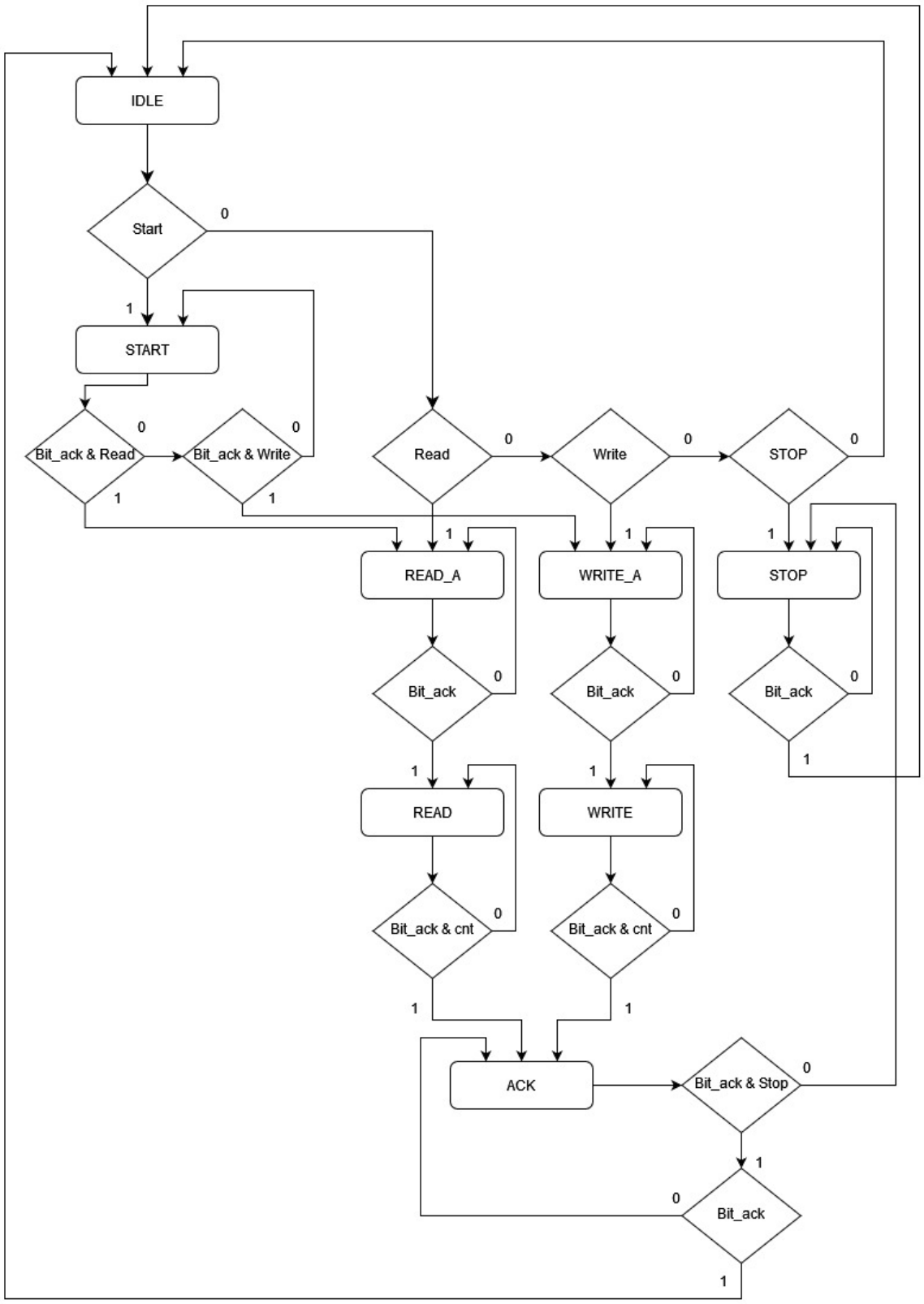

Figura 19: Diagrama d'estats de la unitat de control implementada.

Taula 3: Llista dels recursos utilitzats en la implementació del controlador de byte.

Recurs	**Utilitzats**	**%**
Total logic elements	174 / 22320	< 1
-- Combinational with no register	72	-
-- Register only	21	-
-- Combinational with a register	81	-
Total registers	102 / 23018	< 1
Virtual pins	0	-
I/O pins	29 /154	19
-- Clock pins	2 / 7	29
-- Dedicated Input pins	0 / 9	0
Total block memory bits	0 / 608256	0
Total RAM block bits	0	0
Embedded Multiplier 9-bit elements	0 / 132	0
Total PLLs	0 / 4	0
Fmax summary		
-- Slow 1200mV 85ºC	53.9 MHz	-
-- Slow 1200mV 0ºC	58.75 MHz	-

Verificació Post-síntesi

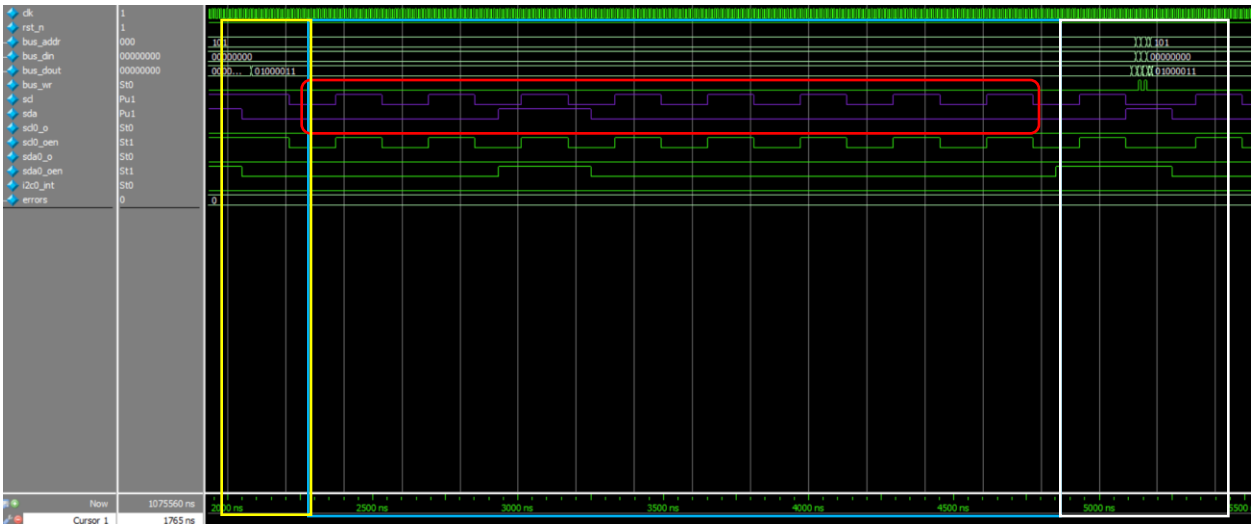

Figura 16: Diagrama d'ones del primer test d'escriptura de la verificació post-síntesi.

En aquesta figura es pot veure com es fa la primera escriptura correctament. Tenim un altre cop els tres estats: un *START* inicial, seguit d'una escriptura *WRITE* i tot seguit un *ACK*, on es fa la lectura de l'acknowledge. No s'aprecien retards.

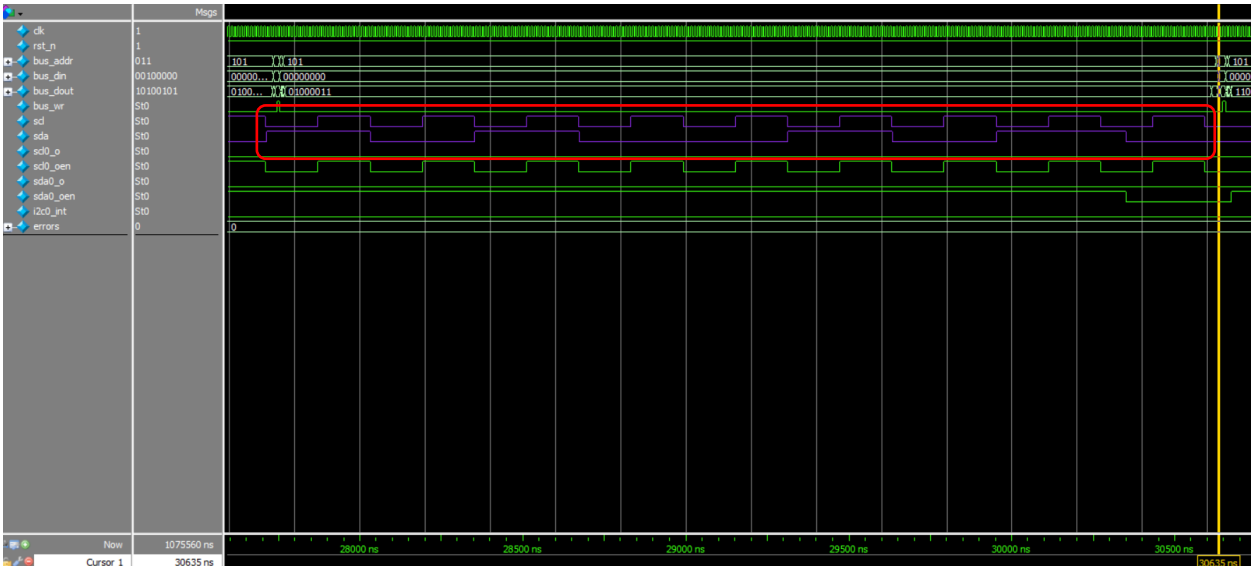

Figura 17: Diagrama d'ones del primer test de lectura de la verificació post-síntesi. En aquest cas tampoc s'aprecien retards.

Com es pot veure, a la verificació post síntesi el sistema funciona correctament i no presenta retards apreciables.