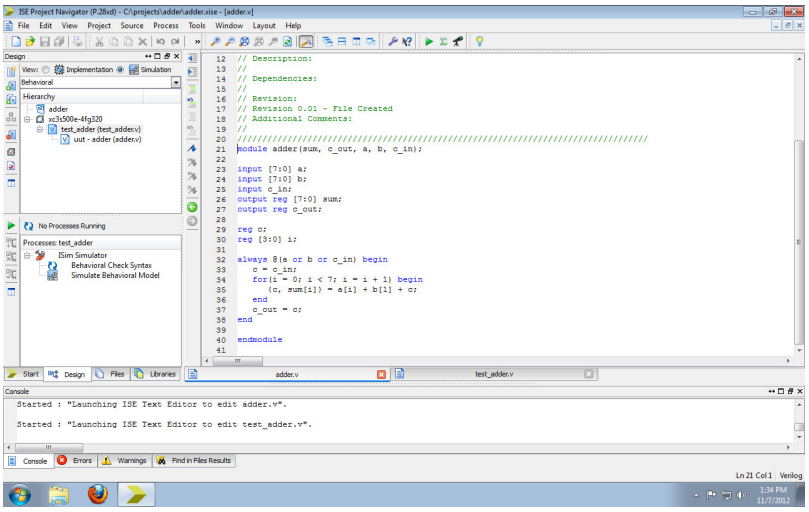
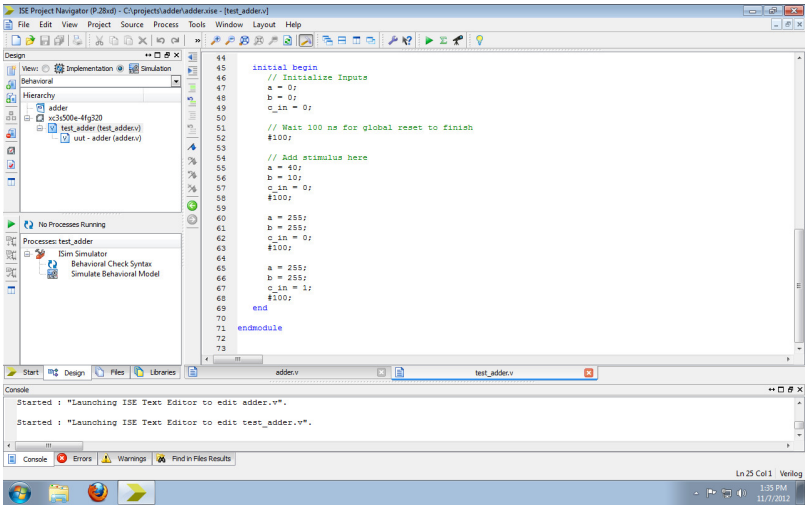
Debugging folosind Xilinx ISE

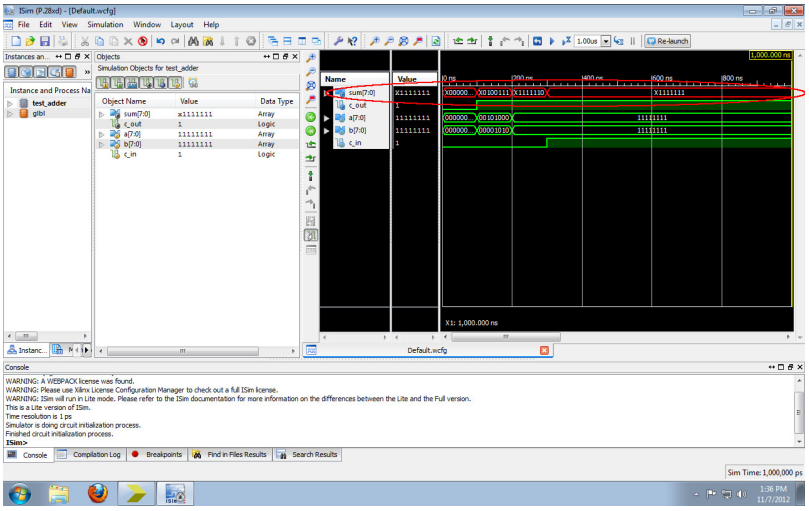
- Tutorialul își propune depanarea unui modul folosind uneltele de debugging puse la dispoziție de simulatorul ISim, integrat în Xilinx ISE. Ca exemplu, se va folosi un sumator cu propagare a transportului (ripple carry). O posibilă implementare în Verilog a acestuia, care conține câteva greșeli, este prezentată în figura următoare.



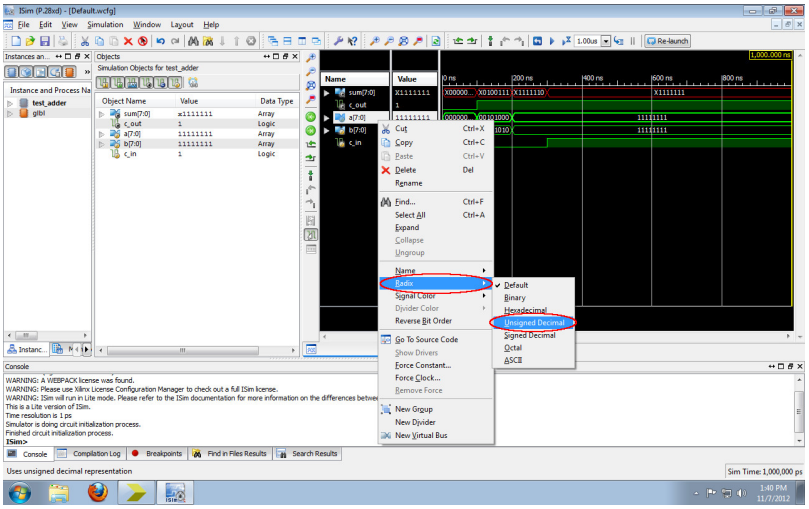
- Modulul de test folosit pentru simulare este următorul.



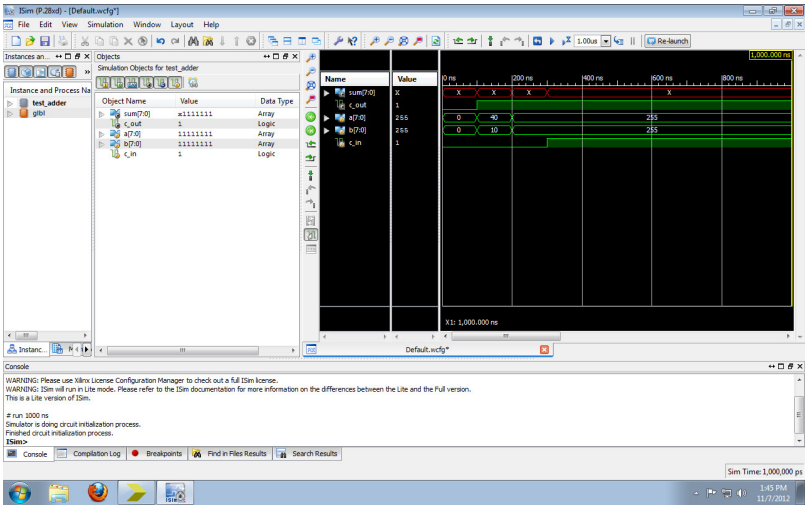
- Simularea inițială a modului poate fi observată în figura următoare. Se observă că semnalul `sum` conține valori nedefinite (x) pe toată durata de simulare a modulului.



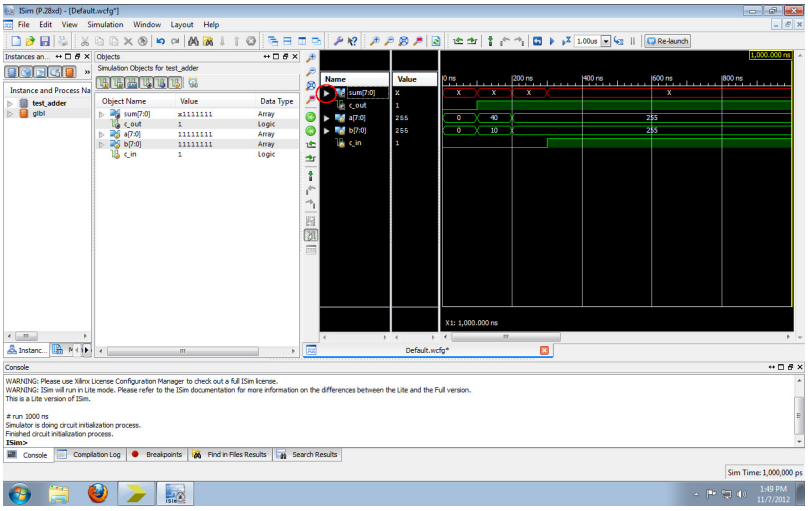
- Pentru a ne ușura analiza semnalelor vom schimba modul în care aceastea sunt afișate. Apăsând *click-dreapta* pe un semnal vom alege ca acesta să fie afișat ca un număr în baza 10, fără semn.



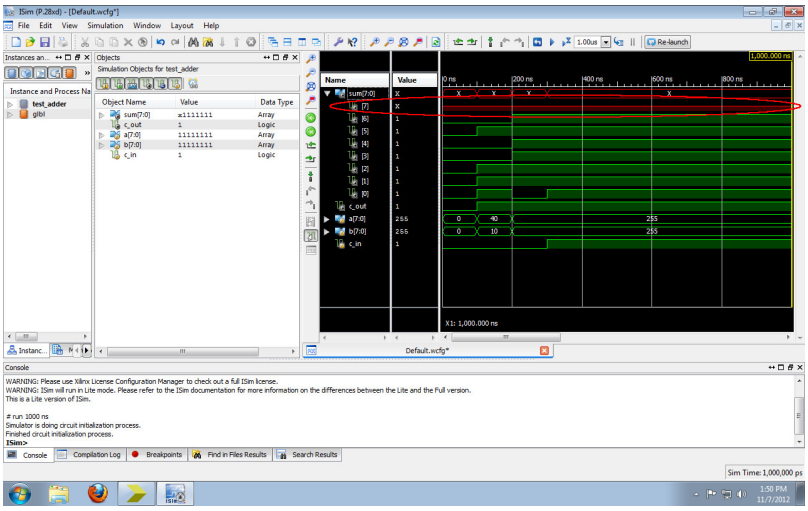
- După ce schimbăm toate semnalele, diagrama va arăta în felul următor.



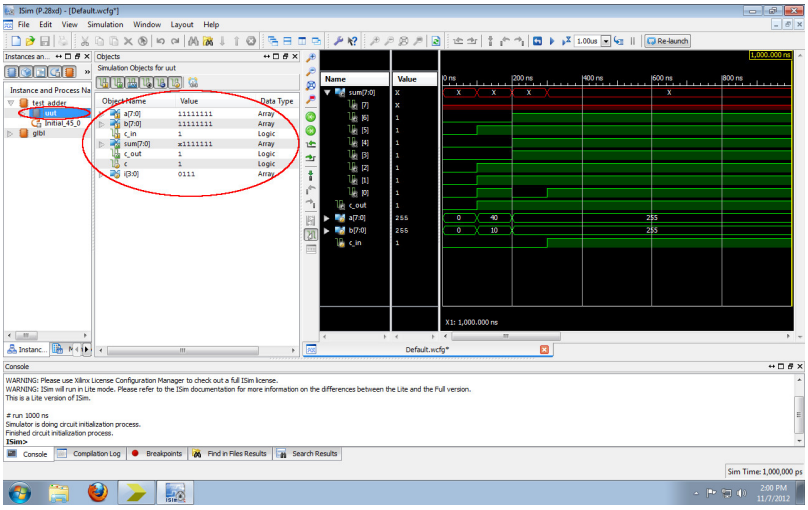
- Putem expanda un semnal pe mai multi biți pentru a inspecta toți biții acestuia.



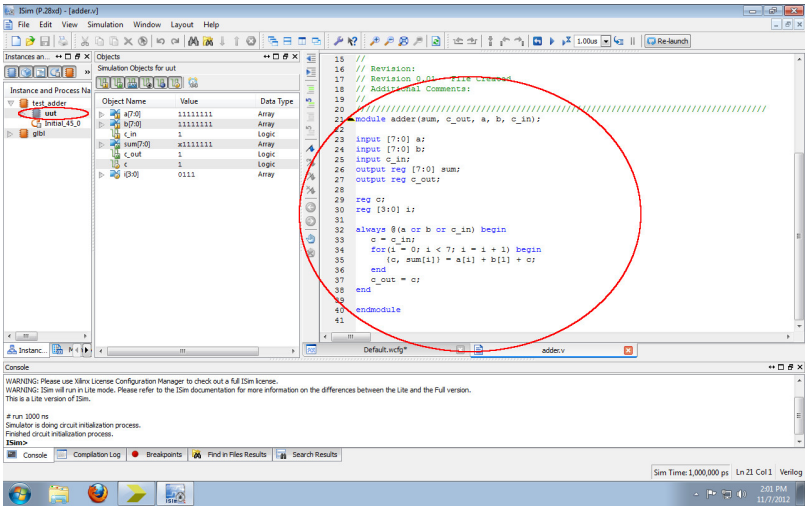
- Se poate observa că doar bitul 7 din `sum` este nedefinit.



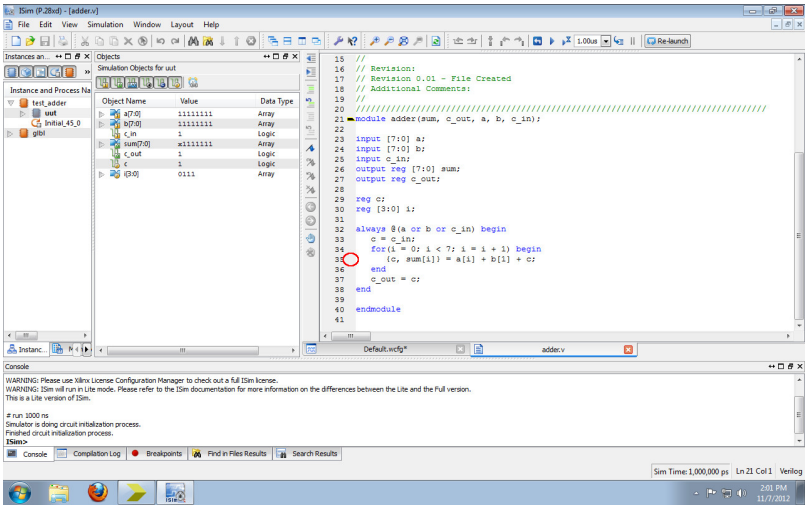
- Putem vedea variabilele interne ale unui modul dacă selectăm instanța acestuia din lista de instanțe. În cadrul ierarhiei, modulul nostru se află imediat sub modulul *test_adder* iar instanța acestuia se numește *uut* (*unit under test*).



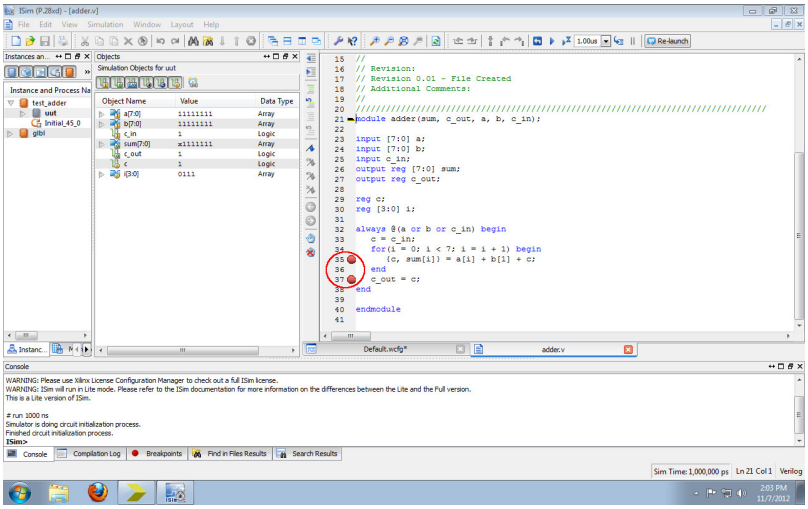
- Putem deschide codul sursă al unei instanțe în simulator făcând *dublu-click* pe aceasta.



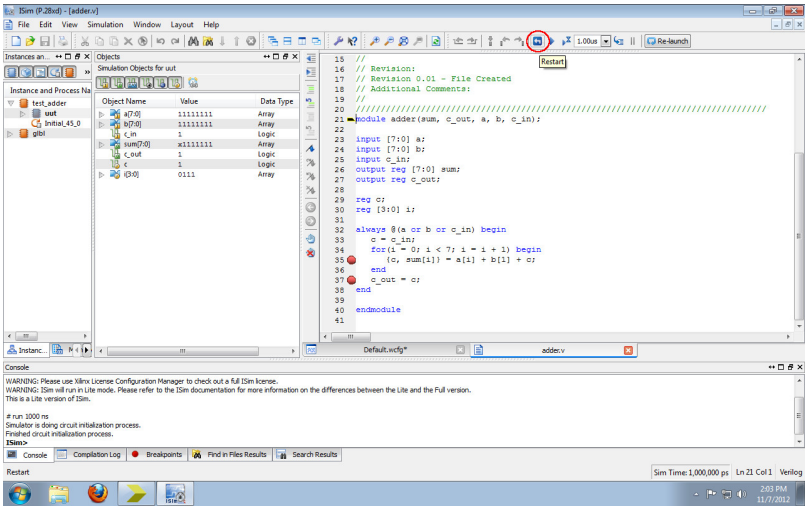
- În cadrul editorului de cod putem activa un breakpoint la linia curentă folosind tasta *F9* sau făcând click pe porțiunea gri din dreapta numărului liniei.



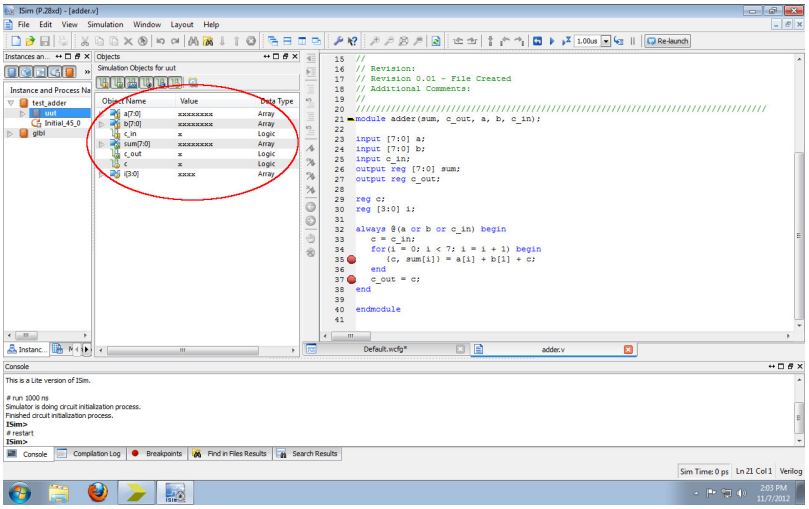
- În continuare vom activa 2 breakpoint-uri, la liniile 35 și 37, pentru a putea inspecta starea variabilelor în fiecare iterație și la sfârșitul ciclului.



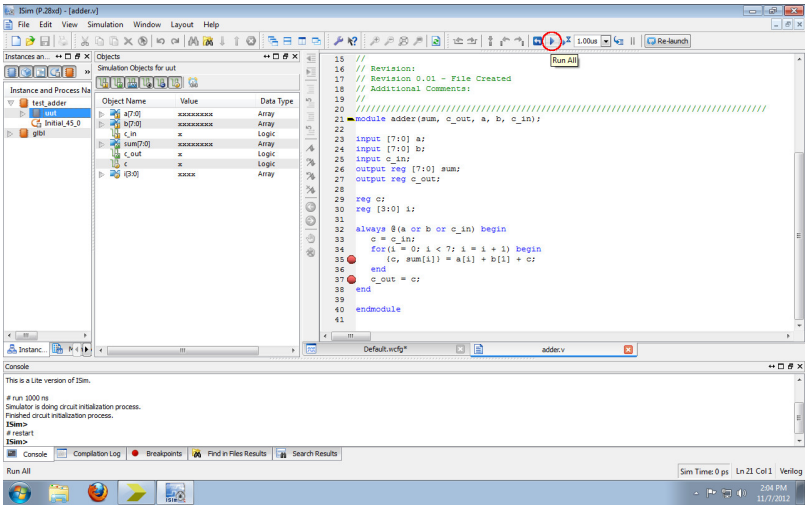
- Repornim simularea folosind butonul *Restart*.



- Se poate observa că la începutul simulării toate variabilele instanței *uut* sunt nedefinite, lucru normal deoarece simularea nu a început încă .

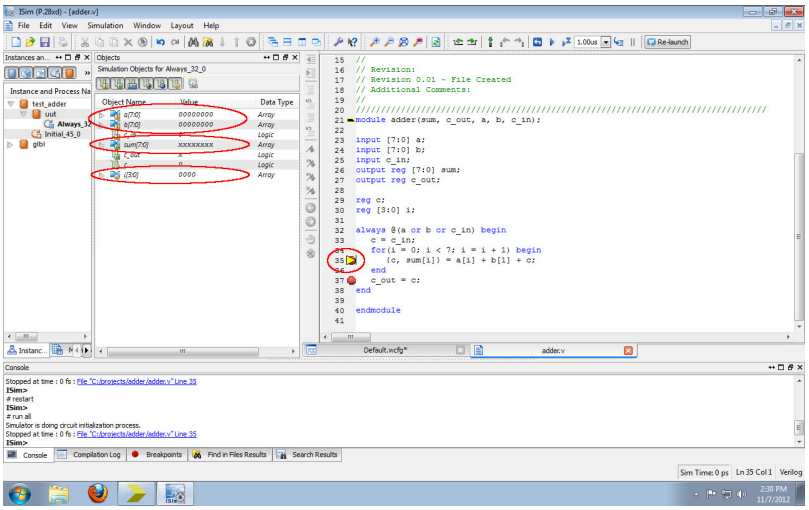


- Folosim butonul *Run All* pentru a porni simularea continuă.

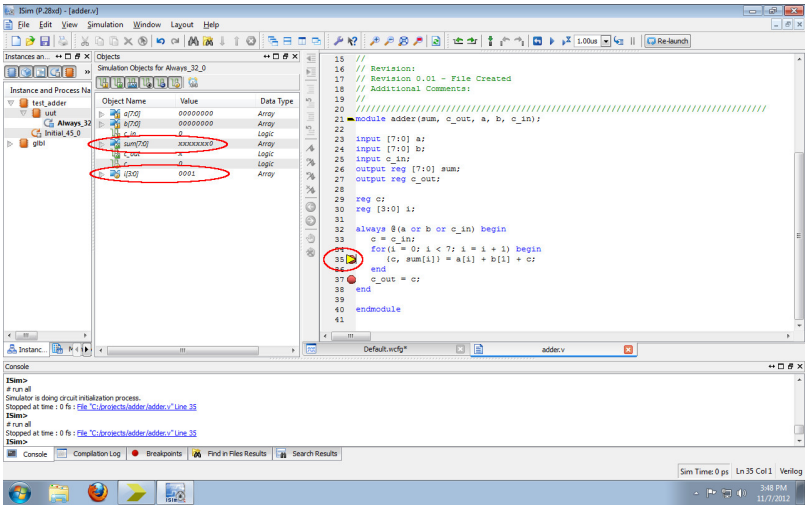


- Simularea se va opri la orice breakpoint întâlnit. În cazul nostrul, primul breakpoint întâlnit este cel de la linia 35, iar simularea se va opri **înainte** de a executa instrucțiunea de la linia 35. În acest moment al simulării putem vedea că ambele intrări ale modulului au valoarea 0, contorul ciclului are valoarea 0 (ne aflăm în prima iterație), iar ieșirile sunt încă nedefinite, deoarece încă nu le-am

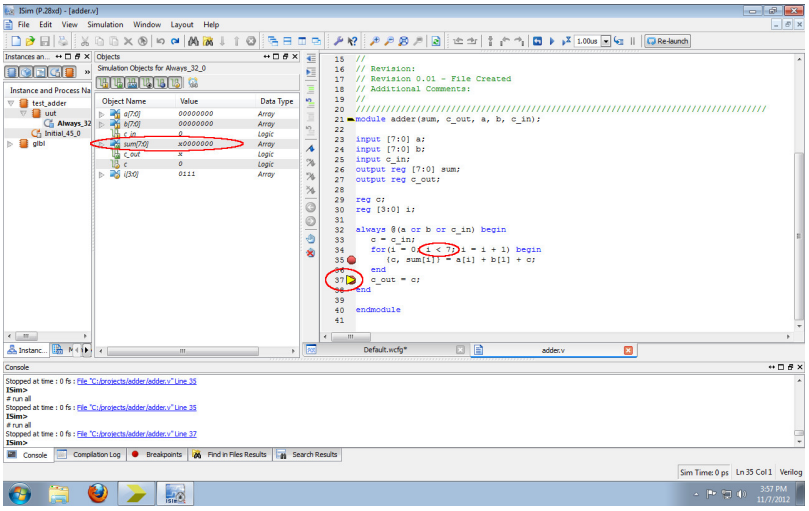
atribuit nici o valoare.



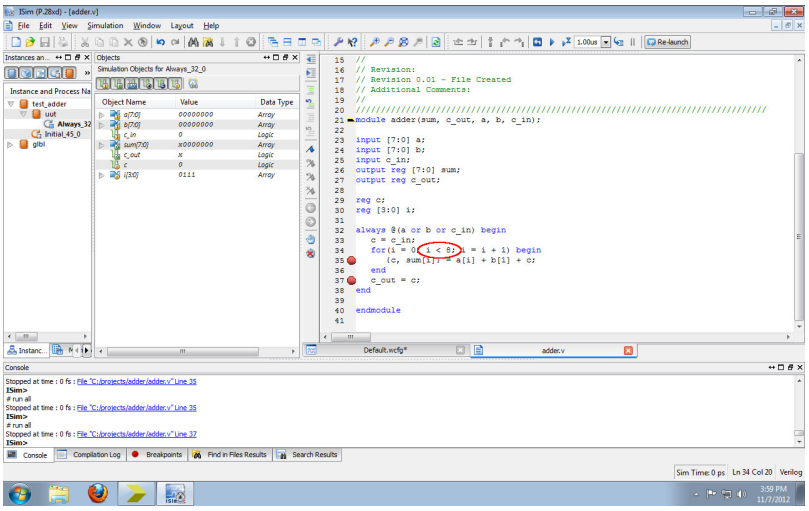
- Vom rula simularea în continuare folosind butonul *Run All*. Simularea se va opri din nou la breakpoint-ul de la linia 35, însă în momentul acesta ne aflăm în a doua iterație a ciclului. Putem vedea lucrul acesta inspectând valoarea contorului, care este 1 (prima valoare a contorului a fost 0). În această iterație vedem că primul bit al lui `sum` a fost setat la valoarea 0.



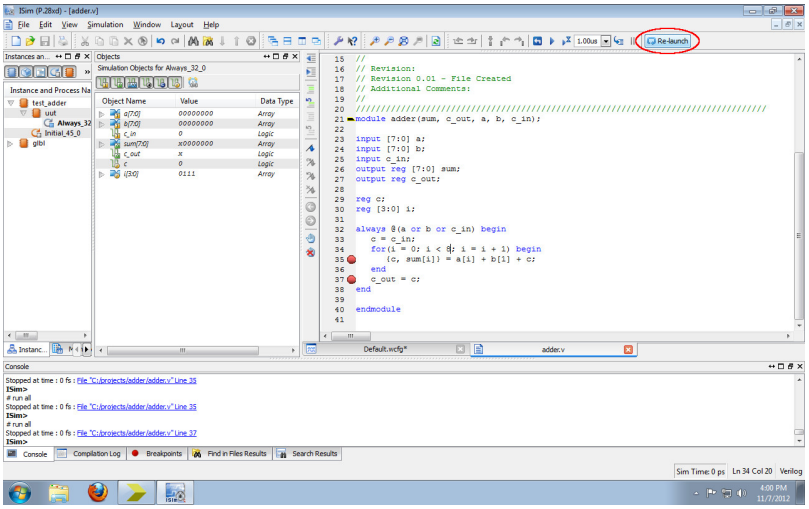
- Rulăm în continuare simularea, folosind butonul *Run All*, până când execuția ciclului se termină și ajungem la breakpoint-ul de la linia 37. În acest moment, pentru o implementare corectă, suma celor două intrări ar trebui să fie calculată complet. Observăm însă că cel mai semnificativ bit al lui `sum` a rămas nedefinit. Înseamnă că ciclul nostru s-a terminat prea repede. Inspectând condiția de oprire, vedem că ieșirea din ciclu se face când `i` devine 7. Iterația corespunzătoare lui `i == 7` nu se execută, ceea ce explică de ce bitul 7 din `sum` este nedefinit.



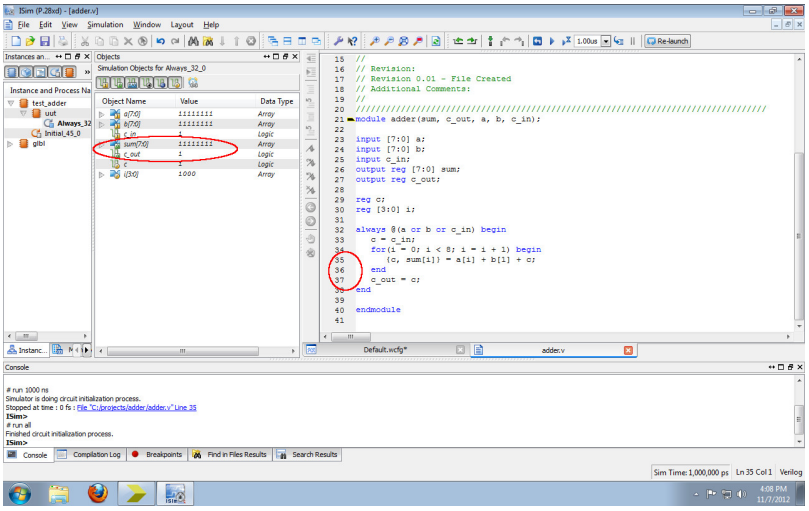
- Modificăm codul corectând cu condiția corectă: `i < 8`.



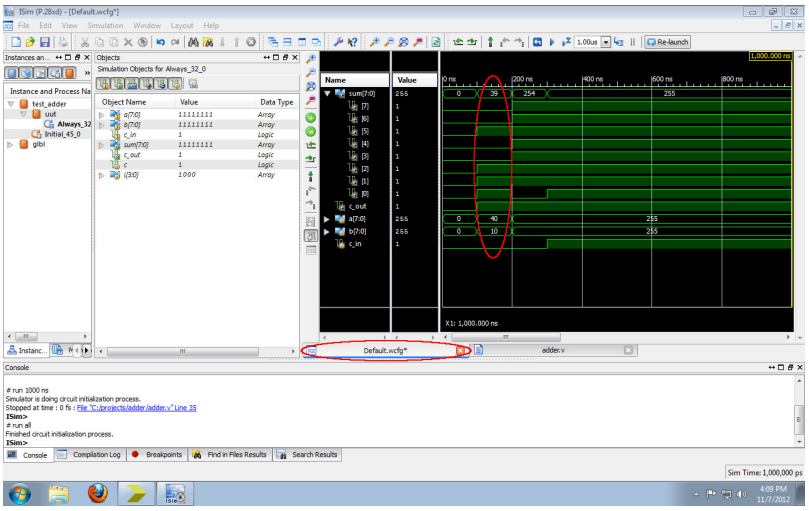
- După ce am modificat codul modulului este necesar să recompilăm simularea pentru a vedea efectele schimbării. Folosim butonul *Re-launch* pentru a recompila și rerula simularea.



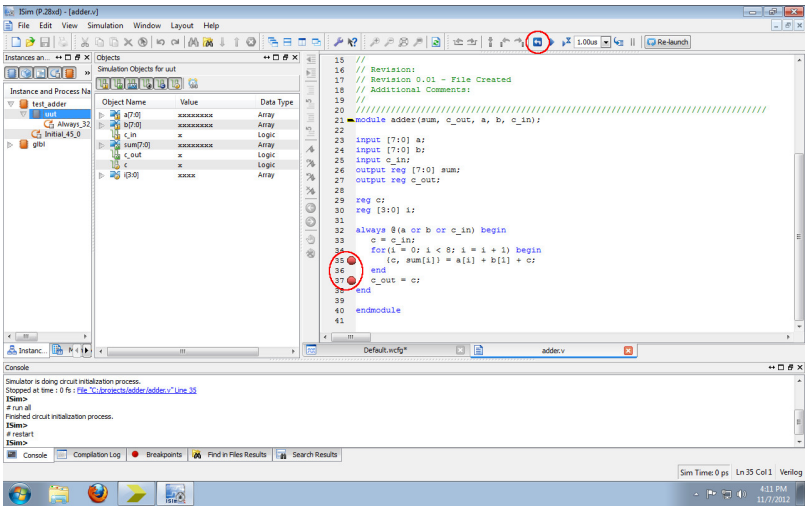
- Eliminăm breakpoint-urile făcând click pe ele, sau folosind tasta *F9*, pentru a putea rula ușor simularea până la capăt. La o primă vedere, din fereastra de variabile, rezultatul pare a fi corect, semnalele nedefinite au dispărut, iar suma pare a fi calculată corect.



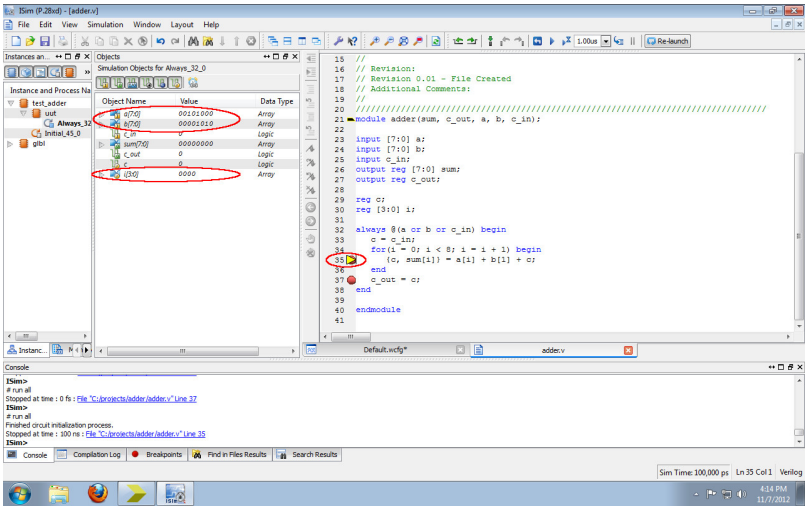
- Inspectarea digramei de semnale, însă, arată că suma dintre 10 și 40 este 39.



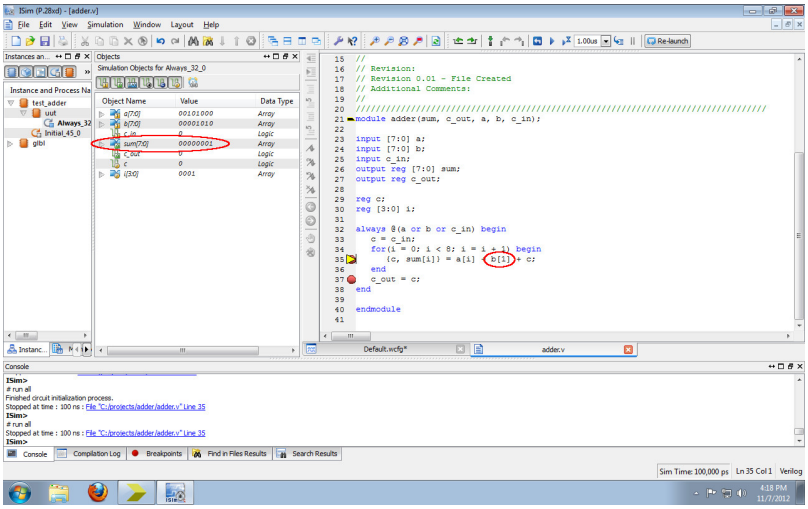
- Adăugăm din nou breakpoint-urile anterioare și reluăm simularea de la început cu butonul *Restart*.



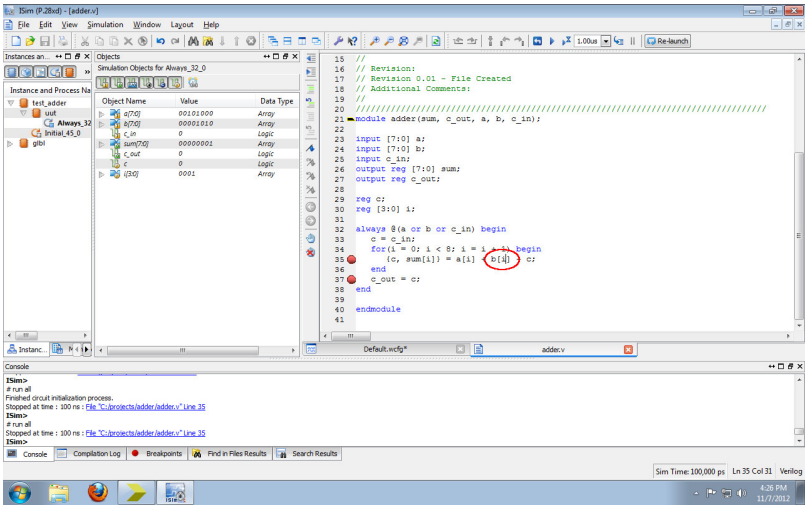
- Rulăm simularea până ajungem în prima iterație a celei de-a doua perechi de numere adunate. Recunoaștem această stare după valorile `00101000` și `00001010` prezente la intrările a și b și valoarea `0` a controului `i`. Valoarea sumei, `00000000`, este rămasă de la adunarea anterioară.



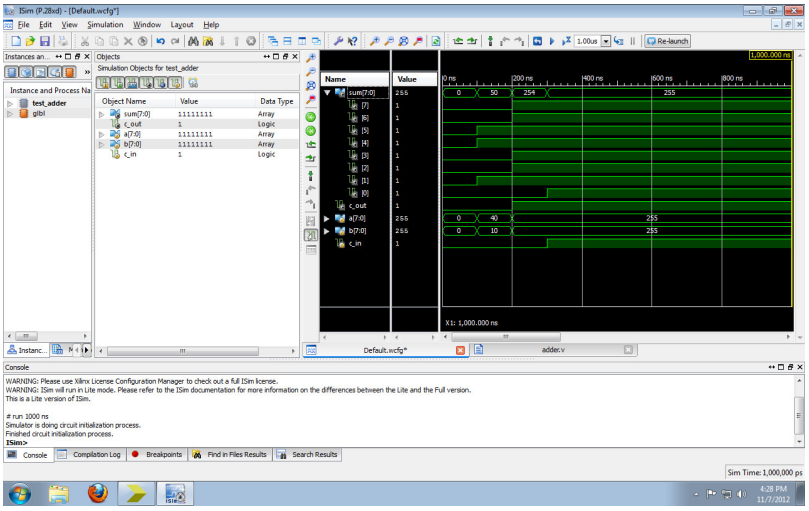
- În a doua iterație vedem că suma calculată în prima iterație pentru cel mai puțin semnificativ bit al lui `sum` este **1**. Din datele de intrare: `a[0] == 0`, `b[0] == 0` și `c_in == 0` am deduce că această sumă ar trebui să fie **0**. Inspectând expresia pentru calcul sumei, observăm că rezultatul pentru toți biții lui `sum` se calculează cu `b[1]`.



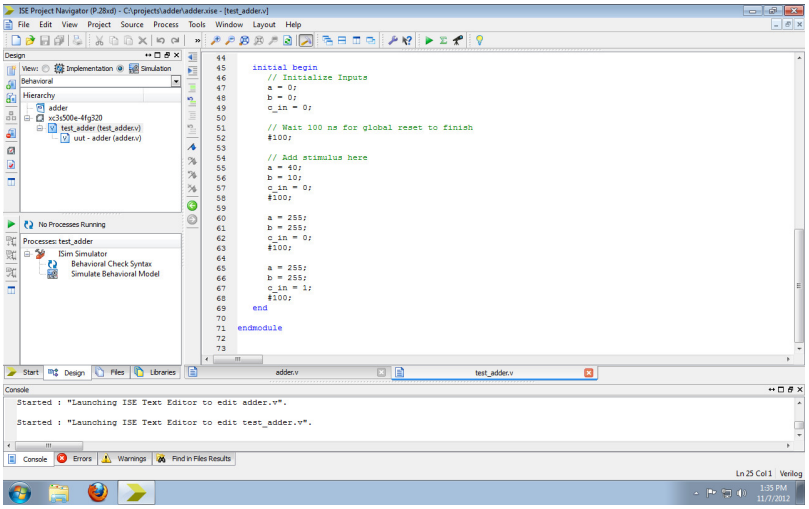
- Modificăm codul, corectând cu expresia corectă a sumei, care depinde de `i`, `b[i]`.



- Dezactivăm breakpoint-urile, recompilăm simularea și inspectăm din nou diagrama de semnale. Cu modificările făcute, circuitul pare să funcționeze corect (cel puțin pentru cele 4 cazuri testate).



- În continuare, pentru a ne asigura de funcționarea corectă a modului, acesta trebuie testat folosind mai multe combinații pentru variabilele de intrare (în special cazuri limită). La descoperirea unei erori în output, vom relua procesul de debugging, pentru a găsi și corecta bug-ul.



PDF tutorial