Verification CA#1

Student Name:
Pouya Haji Mohammadi Gohari

SID:810102113

Date of deadline
Saturday 18th November, 2023

Dept. of Computer Engineering

University of Tehran

Contents

1 Introduction

In this computer assignment we are going to implement and coding in a modular way.
Here 2 sensors competing with each other to send their data through a sync channel (zero capacity) to CPU.CPU collects data recived by this channel and pack them into size of four and sent it to Radio. Radio in other hand will receive this packet.
So far was dicussed in the class, now we have more these divices and also a collector added.
We have multiple Radios sending their packed data and collector deciding which one to recieve. In Review section, we describe how to code modular and explain every part of code.
At last section we analysis our outputs and explain what our code does.

2 Review of my code

2.1 Define global values

In this section we are going to declare our global variables as you can see in figuer 1.

```
#define PACKET_SIZE 4;
#define RANDOM  (seed * 3 + 14) % 100

int NODES = 3;
chan bluetooth[NODES] = [0] of {int};
chan BUS[NODES] = [0] of {int, int, int, int};
chan Collector_BUS[NODES] = [0] of {int, int, int, int};
bool reciveFromBus[NODES] = false;
bool nodeCanSent[NODES] = false;
int seed = 0;
```

Figure 1: GLobal variabels

Declarations:

- We define *PACKET_SIZE* as 4 cause CPU is going to pack recieved-data by two sensors and send it to Radio.

- We have a random defined which multiply seed by 3 adding it to 14 and find remainder of 100.
- NODES stands for number of Radio we have(for modular coding).
- Blothooth channel are for sensors which have competiton to send data for CPU via this channel.
- BUS channel is designed for CPU that sends data for Radio.(Sending data will be packed as size of 4 so we difined BUS channel as $[0]\ of\ \{int, int, int, int\}$)
- Collector_BUS is basically a channel between number of Radios and a collector.
- reciveFromBus is array of boolean and are inititalized with false. Also if Radio received datas from CPU it will change to true and if it sends to Collector it will change back to false.
- nodeCantSent is also a boolean array set to false. It is designed for Collector processor if data was received from i_th Radio then $nodeCantSent[i] = true$;.
- At last we defined seed as 0.

2.2 Sensors

It is necessary to have two sensors for each CPU and Radio so we defined two Sensors with same behavior defined in figure 2.

```
proctype Sensor1(int i) {
    do
    :: seed == 100 -> seed = 0;
    :: else -> bluetooth[i]!RANDOM; seed++;
    od
}

proctype Sensor2(int i) {
    do
    :: seed == 100 -> seed = 0;
    :: else -> bluetooth[i]!RANDOM; seed++;
    od
}
```

Figure 2: Behavior of sensors

we changed Sensor structure a little bit to generate random number each time in order to see variate numbers which will be sent to radio over time.(If seed is equal to 100 we will set it to zero.)
Second reason of changing this structure and not sending 0 over time is for debuging would be easier to us.

2.3 CPU

This processor will pack received-data from sensors till it reach to *PACKET_SIZE*. After reaching to this state CPU send packed-data to radio and change receivedDate to zero.(figure 3).

```
proctype CPU(int i) {
    int sensorData[4] = 0;
    int receivedData = 0;
    do
    :: receivedData < PACKET_SIZE -> bluetooth[i]?sensorData[receivedData];
                receivedData++;
    :: else -> BUS[i]!sensorData[0],sensorData[1],sensorData[2],sensorData[3];
                receivedData = 0;
    od
}
```

Figure 3: Processor of CPU.

2.4 Radio

Radio gets the packed-data from CPU via BUS channel with gaurd of $reciveFromBus[i] == false$ and change it to true.(It means that it has data to send to collector).
Afterwards Radio start to send data to collector via channel Collector_BUS and again change $reciveFromBus[i]$ to true, for more details see figure 4.

```
proctype Radio(int i) {
    int comData[4];
    do
    :: reciveFromBus[i] == false -> BUS[i]?comData[0],comData[1],comData[2],comData[3];
                                    reciveFromBus[i] = true;
    :: reciveFromBus[i] == true  -> Collector_BUS[i]!comData[0],comData[1],comData[2],comData[3];
                                    reciveFromBus[i] = false;
    od
}
```

Figure 4: Processor for Radio.

2.5 Collector

Most impotant part of project lies within collector's processor.At first let's look at fundamental of particular code represented in figure 5.

```
active proctype Collector() {
    int comData[4];
    int count = 0;
    int defindeOnce = 0;
    do
    :: defindeOnce == 0 && count < NODES ->
                                    run Sensor1(count);
                                    run Sensor2(count);
                                    run CPU(count);
                                    run Radio(count);
                                    count ++;
    :: nodeCanSent[count] == false && defindeOnce == 1 && count < NODES ->
                                    Collector_BUS[count]?comData[0],comData[1],comData[2],comData[3];
                                    nodeCanSent[count] = true;
                                    count ++;

    :: count == NODES -> atomic{do
                        :: count > 0 -> nodeCanSent[count-1] = false;count --;
                        :: else -> count = 0;
                                   defindeOnce = 1;
                                   break;
                        od}
    od
}
```

Figure 5: Collector's processor.

As you can see we define comData for purpose of receiving data from radio via channel Collector_BUS. Also we defined counter and definedOnce. Counter stands for getting data respectively from number of Radios defined in global. For runing other processors within collector, we defined another value called definedOnce. More generally we use it in order just to create processors once for number of nodes we have. Next guard is when i-th radio didn't send any data yet, collector gets data and also change nodeCantSent of that node to true (means that it already sent it's data and can not send any more if other Radios didn't

send data).
Last guard is when counter is equal to number of nodes(NODES).This guard is designed for two purpose:

- First purpose is when we defined number of nodes(NODES) and we don't want to define more processors than already declare in NODES.(So we change definedOnce to 1)
- Second purpose is when collector finishes collecting data for number of Radios(NODES) phase.Cause of getting data by collector, we must start over this operation again.To do such a thing we should set nodeCantSent to false for all of Radios.

3 Analysis

In this part we run the code to see our traces. In figure 6 run processors for each node we have a radio,

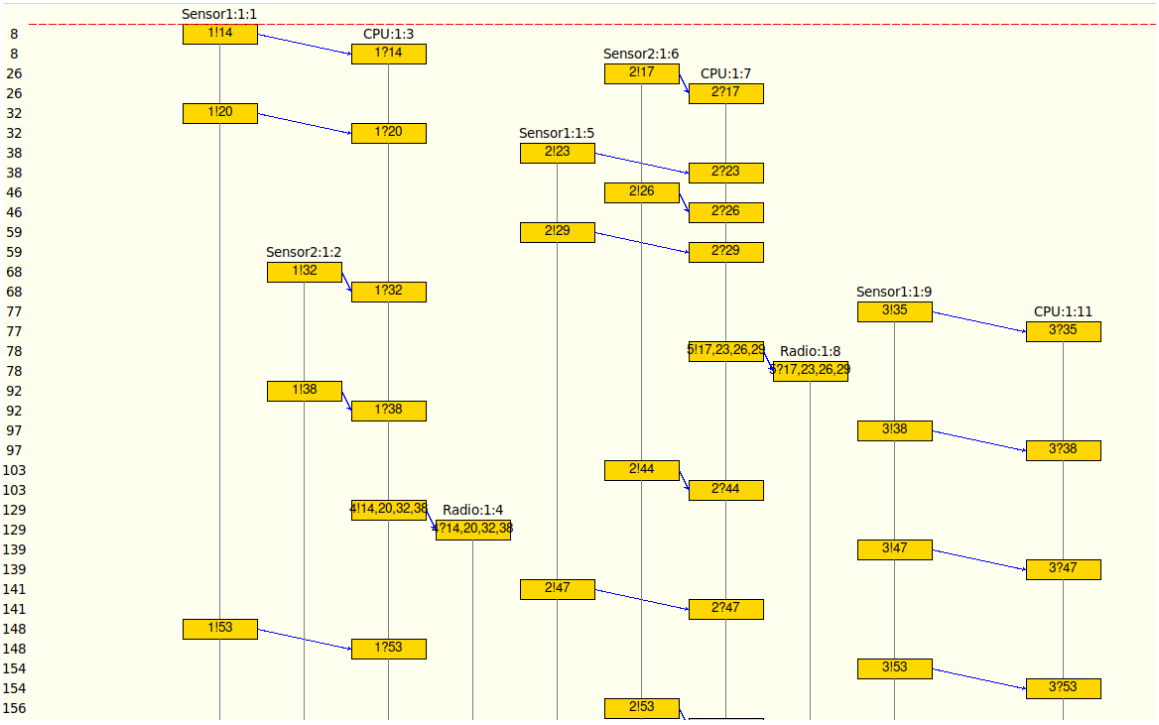

Figure 6: Run processors

CPU ,2 sensors.

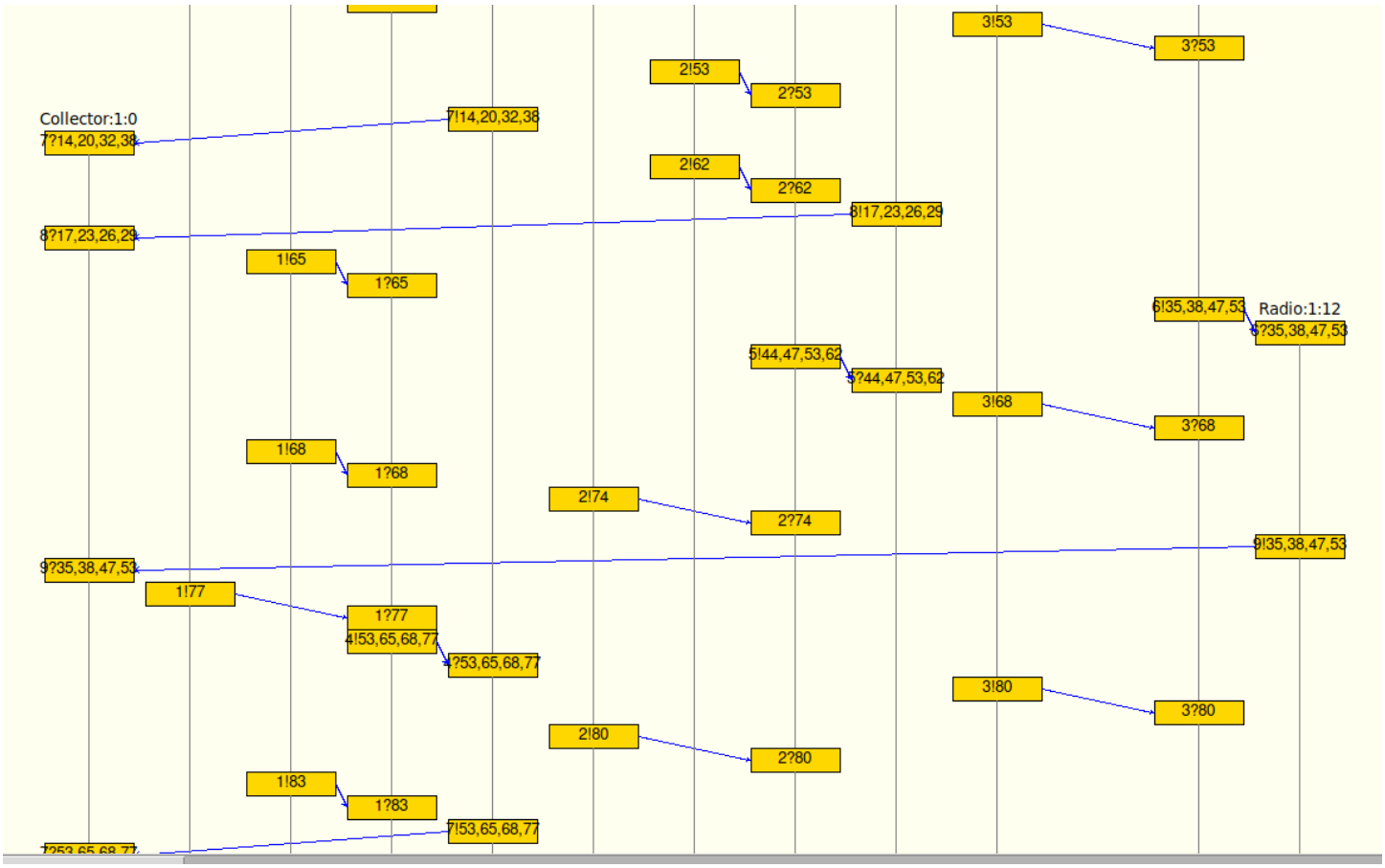

Figure 7: Sending data to collector.

In figure 7 we can see that collector is getting data from recivers from radios in order.
Now let's see the traces.

```
  0:        proc  - (:root:) creates proc  0 (Collector)
  1:        proc  0 (Collector:1) ca_1.pml:53 (state 1) [((defindeOnce==0)&&(count<NODES)))]
Starting Sensor1 with pid 1
  2:        proc  0 (Collector:1) creates proc  1 (Sensor1)
  2:        proc  0 (Collector:1) ca_1.pml:54 (state 2) [(run Sensor1(count))]
  3:        proc  1 (Sensor1:1) ca_1.pml:16 (state 3) [else]
Starting Sensor2 with pid 2
  4:        proc  0 (Collector:1) creates proc  2 (Sensor2)
  4:        proc  0 (Collector:1) ca_1.pml:55 (state 3) [(run Sensor2(count))]
  5:        proc  2 (Sensor2:1) ca_1.pml:23 (state 3) [else]
Starting CPU with pid 3
  6:        proc  0 (Collector:1) creates proc  3 (CPU)
  6:        proc  0 (Collector:1) ca_1.pml:56 (state 4) [(run CPU(count))]
  7:        proc  3 (CPU:1) ca_1.pml:31 (state 1) [((receivedData<4))]
  8:        proc  1 (Sensor1:1) ca_1.pml:16 (state 4) [bluetooth[i]!(((seed*3)+14)%100)]
  8:        proc  3 (CPU:1) ca_1.pml:31 (state 2) [bluetooth[i]?sensorData[receivedData]]
Starting Radio with pid 4
  9:        proc  0 (Collector:1) creates proc  4 (Radio)
  9:        proc  0 (Collector:1) ca_1.pml:57 (state 5) [(run Radio(count))]
 10:        proc  4 (Radio:1) ca_1.pml:41 (state 1) [((reciveFromBus[i]==0))]
 11:        proc  0 (Collector:1) ca_1.pml:58 (state 6) [count = (count+1)]
 13:        proc  3 (CPU:1) ca_1.pml:32 (state 3) [receivedData = (receivedData+1)]
 14:        proc  1 (Sensor1:1) ca_1.pml:16 (state 5) [seed = (seed+1)]
 15:        proc  0 (Collector:1) ca_1.pml:53 (state 1) [((defindeOnce==0)&&(count<NODES)))]
Starting Sensor1 with pid 5
 16:        proc  0 (Collector:1) creates proc  5 (Sensor1)
 16:        proc  0 (Collector:1) ca_1.pml:54 (state 2) [(run Sensor1(count))]
 17:        proc  5 (Sensor1:1) ca_1.pml:16 (state 3) [else]
Starting Sensor2 with pid 6
 19:        proc  0 (Collector:1) creates proc  6 (Sensor2)
 19:        proc  0 (Collector:1) ca_1.pml:55 (state 3) [(run Sensor2(count))]
Starting CPU with pid 7
 21:        proc  0 (Collector:1) creates proc  7 (CPU)
 21:        proc  0 (Collector:1) ca_1.pml:56 (state 4) [(run CPU(count))]
 22:        proc  1 (Sensor1:1) ca_1.pml:16 (state 3) [else]
 23:        proc  6 (Sensor2:1) ca_1.pml:23 (state 3) [else]
 24:        proc  7 (CPU:1) ca_1.pml:31 (state 1) [((receivedData<4))]
 25:        proc  3 (CPU:1) ca_1.pml:31 (state 1) [((receivedData<4))]
 26:        proc  6 (Sensor2:1) ca_1.pml:23 (state 4) [bluetooth[i]!(((seed*3)+14)%100)]
 26:        proc  7 (CPU:1) ca_1.pml:31 (state 2) [bluetooth[i]?sensorData[receivedData]]
 27:        proc  6 (Sensor2:1) ca_1.pml:23 (state 5) [seed = (seed+1)]
 29:        proc  7 (CPU:1) ca_1.pml:32 (state 3) [receivedData = (receivedData+1)]
Starting Radio with pid 8
 30:        proc  0 (Collector:1) creates proc  8 (Radio)
 30:        proc  0 (Collector:1) ca_1.pml:57 (state 5) [(run Radio(count))]
 31:        proc  8 (Radio:1) ca_1.pml:41 (state 1) [((reciveFromBus[i]==0))]
 32:        proc  1 (Sensor1:1) ca_1.pml:16 (state 4) [bluetooth[i]!(((seed*3)+14)%100)]
 32:        proc  3 (CPU:1) ca_1.pml:31 (state 2) [bluetooth[i]?sensorData[receivedData]]
```

Figure 8: Starting processors.

In trace 8 at first collector is start to work cause it is defined active. Next part program will create 2 sensors, 1 CPU and 1 Radio for each node. This procedure will be continued till all of processors created.(we have 3 nodes here)

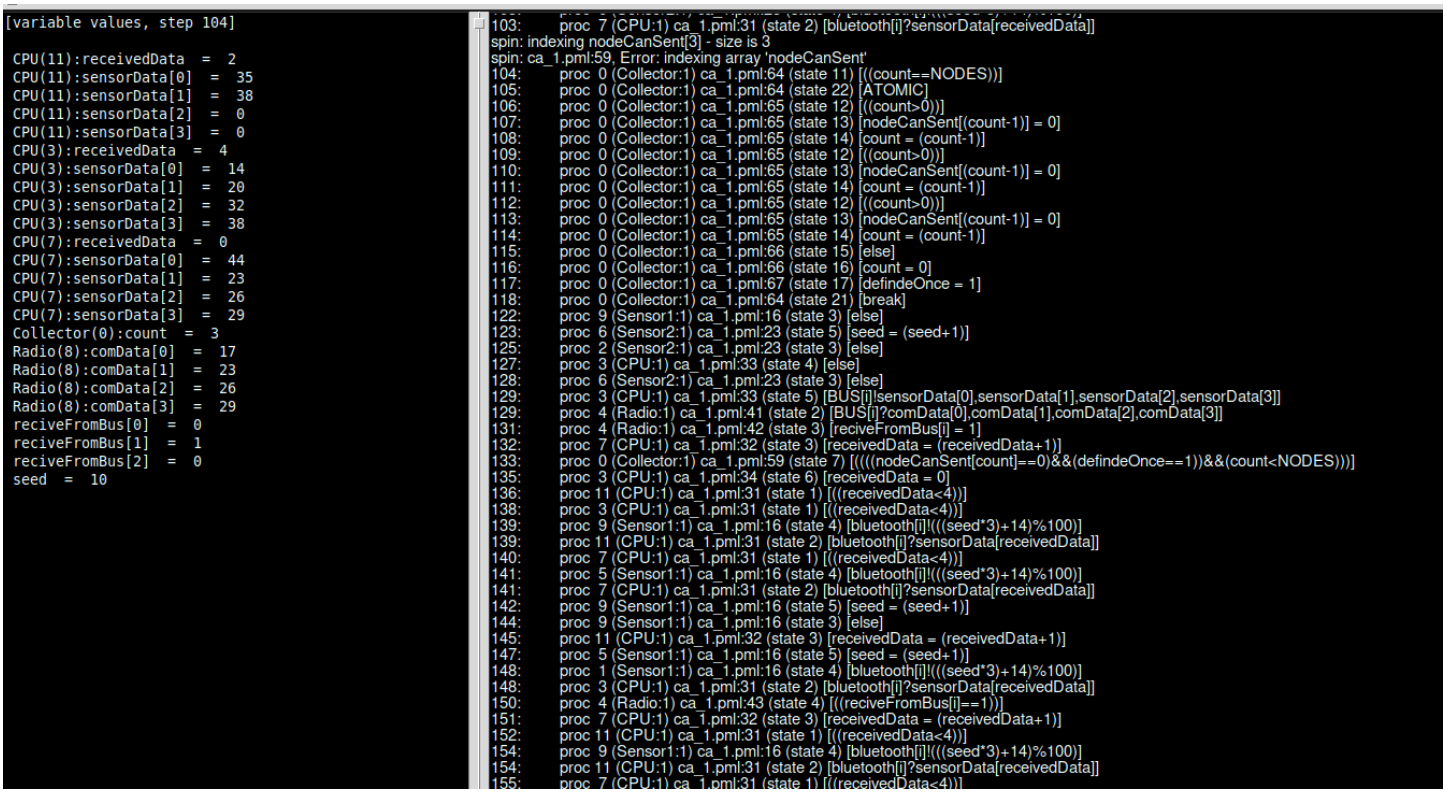

Figure 9: set counter to zero and definedOnce to 1

In trace 9 we can see collector starts to recieve data from Radios, and also whole Sensors, CPUs, Radios are working even if data was not collected from collector.

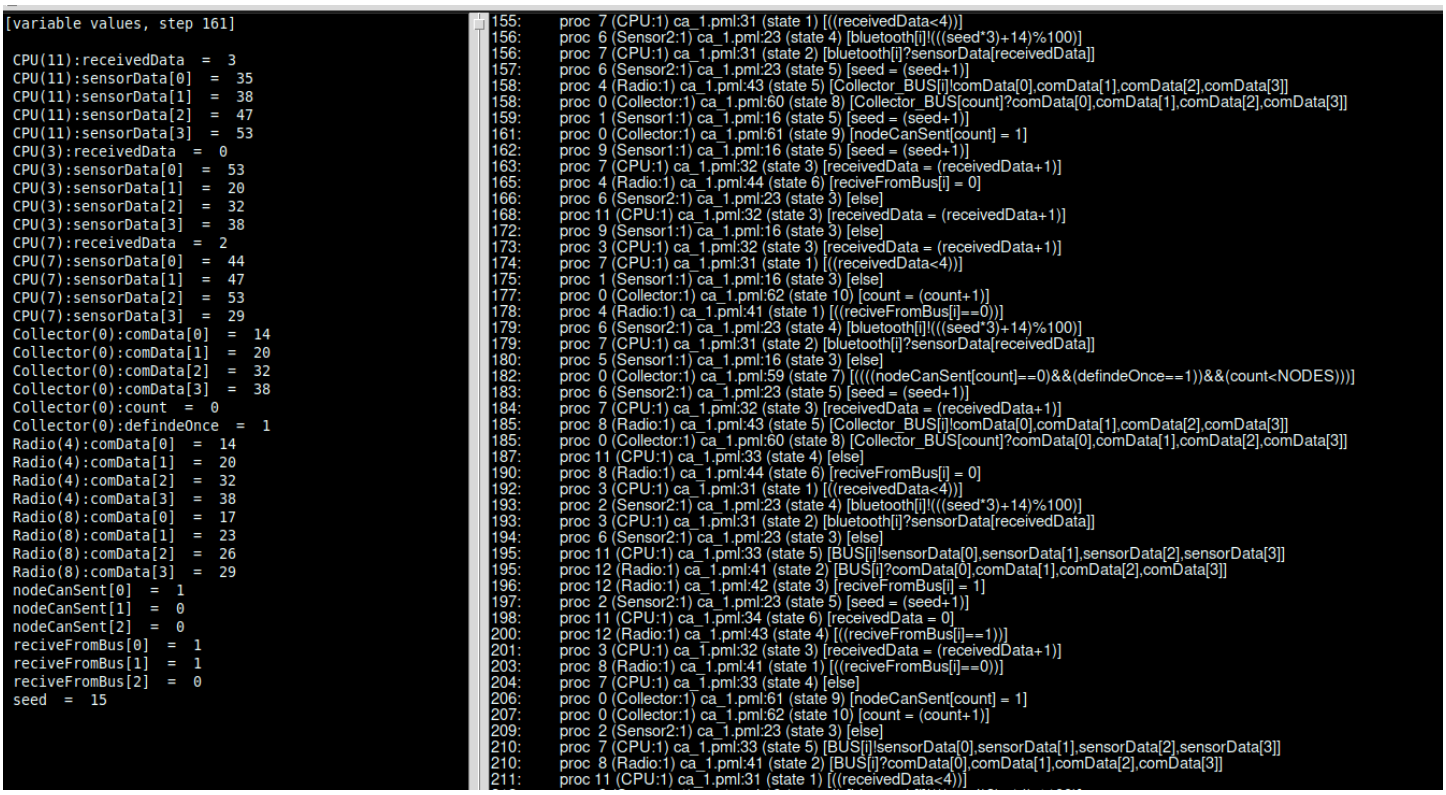

Figure 10: line 158 collector received it first data.

As you can see in trace 10 collector received it's first data from radio and it is from node 0 cause to count is 0.
In line 161 collector wants to receive data but it can not becuase nodeCantSent[0] is true means 0-th Radio has sent data already and collector won't get that data again. Afer counter increase by 1 in line 177, in line 182 collector checks if data is given by 1-th radio and 1-th radio does not sent any message then guard will be satisfied. Eventually in line 185 will get that data from 1-th radio.
This procedure will be continued till all radio in order sent their data.

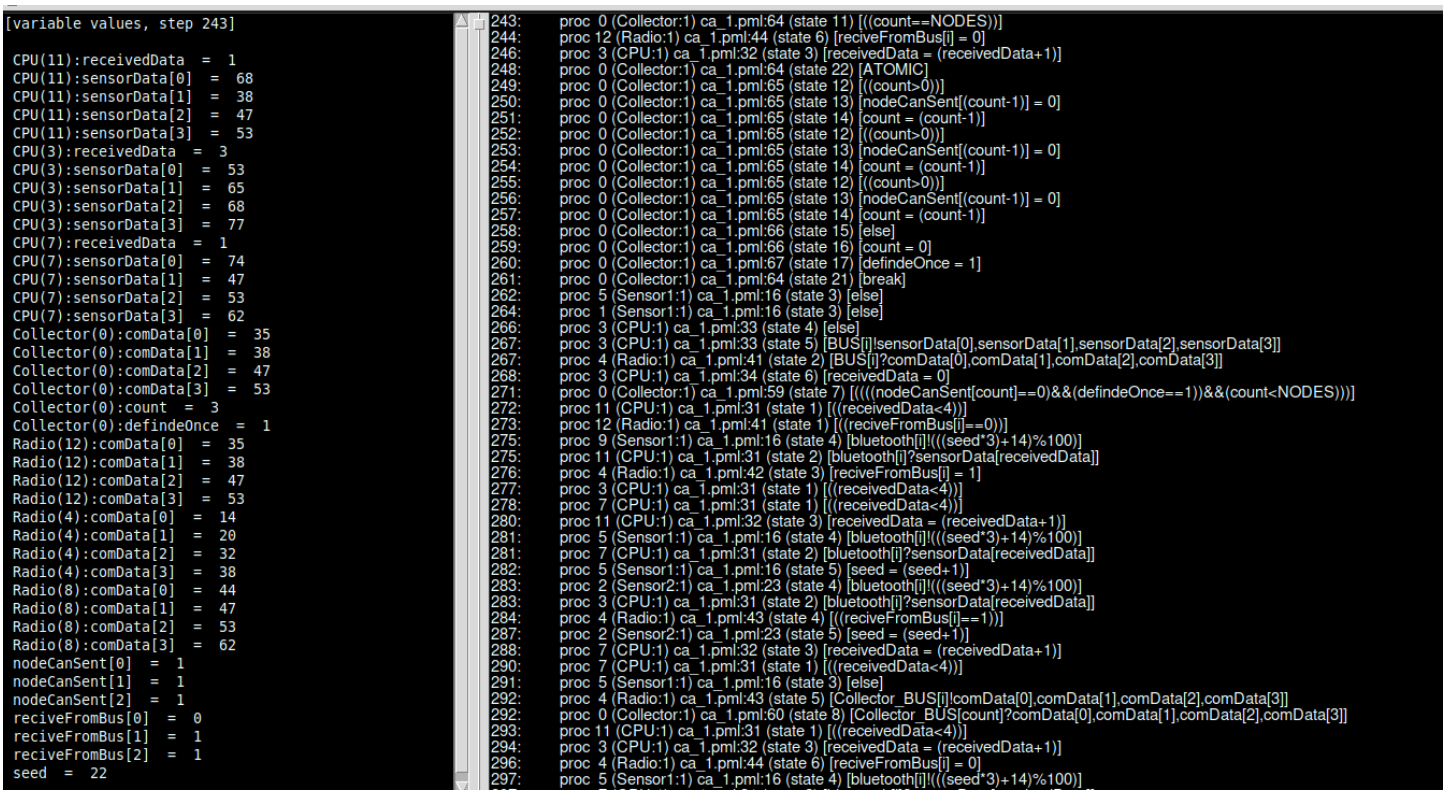

Figure 11: Counter is equal to NODES.

In trace 11 we can see in line 243 counter is equal to NODES, then in line 248 will do an atomic job which sets all nodeCantSent to false and start over the routine procedure.