

Formal Verification of Hardware and Software Systems
Final Project: Verification of Cache Protocols using Model
Checking

CSEE W6863 Fall 2022
Cam Coleman, Qiao Haung
cc4535, qh2234

1 Qiao's Side

1.1 Contributions

Our main contribution is model-checking 3 MSI cache protocol variants, which ranged from 6 to 14 states for the cache controllers and 4 states for the memory controller, under different bus assumptions. The difference between those variants is they work in different buses: The simplest one only works on the atomic-request-atomic-transaction bus, while the most complicated one works on the real-world non-atomic bus. What interests us is how the non-atomicity affects the design of the MSI protocol, so we modeled 3 different buses as well, and - Verified that these variants are correct under corresponding bus atomicity assumptions - Run the simpler protocols under non-atomic buses, verified that they went wrong, and used the error-tracing feature to see why they didn't work and how the more advanced variants fix the problem. In the following section, we will talk about which MSI variants we model checked, how we model them along with different buses with TLA+ and our results.

1.1.1 MSI cache coherence model

Consider one cache line (or other units for atomic data storage/transition) of data under the MSI protocol. For each core in the system, as its name suggested, the data has 3 stable states: "M/Modified" means the data is locally modified and not necessarily synced to the memory, "S/Shared" means that the data is in the read mode and potentially several cores have a copy of this data simultaneously, "I/Invalid" means that data is either not in the local cache or outdated. It's trivial to model the transitions if we only consider the stable states: - Before any operation, sync the local cache of other cores to the memory and make sure other cores are not in the M state - When core A writing data, we set A to the M state and other cores to the I state - When core A reading data, we set A to the S state

While the idea is straightforward, it's not as easy as it sounds to implement under concurrent circumstances. The way that MSI protocol sync between the cores is bus snooping: every core observe the request sent by other cores on the bus and process them in the same order. An example of a core read timeline would look like this: 1. Core A sends a request on the bus 2. Every core process the request (A included) 3. The memory sends back the requested data 4. Core A receives the data and updates its state

There are 3 kinds of requests that each core can send to the bus: - GetS: The core wants to read from the memory and become the "S" state - GetM: The core wants to write to the memory and become the "M" state. It's called "Get" because the core is not in the "M" state now, so it has to fetch the most recent value and make sure other cores have written their changes. - PutM: The core wants to write to the memory while it's in the "M" state. There is an Upgrade request we didn't consider because it's optional and doesn't introduce more insight. In our TLA+ program, these are all 3 requests that a core can send to the bus. Moreover, a core can also snoop and process a request on the bus, or write to the local data if permitted, in other words, in the state "M". We model the behavior of the cores in TLA+ as follows:

```
coreAction(core) ==  
  \/\ coreBusEvent(core) \* snoop and process the request on the bus
```

```

\ / getS(core)
\ / getM(core)
\ / putM(core)
\ / /\ write(core)
  /\ UNCHANGED << state, bus, queue >>

```

The `coreBusEvent(core)` function implements the state transition of the MSI protocol, transient states included.

The memory also has its state machine, but it can't send requests subjectively. All that the memory can do is process the requests on the bus and send/store data accordingly. Combined, we can model the MSI protocol in TLA+ as:

```

Next_MSI = \ / \E core \in Cores: coreAction(core)
           \ / memoryAction

```

So each step of the TLA+ model corresponds to a time slot on the bus where only one core is granted bus access. We choose this abstraction level to model the MSI protocol because we can both show the impact of bus characteristics on the variants of MSI protocols and limit the model complexity.

For a detailed description of the state machine for cores and memory, please refer to [1] and our code.

1.1.2 Bus model

We are interested in 2 characteristics of the bus: atomic-request, and atomic-transaction. Still, we only consider the requests about the same unit of data. The atomic request means that every request sent by the cores appears on the bus and gets processed immediately, in other words, there won't be any request between a core issuing a request and itself receiving it from the bus. The atomic transaction means that when core A issues a request, other requests will not appear on the bus until A receives the response for the request; However, other cores could issue requests during the transaction. An example of an atomic-transaction but not atomic-request bus could be: 1. Core A issues GetS 2. Core B issues GetS 3. A:GetS appears on the bus and every one process it 4. Memory put data on the bus and A receives the data from the bus 5. B:GetS appears on the bus and every one process it 6. Memory put data on the bus and B receives the data from the bus Since core B issues a GetS request when the first request from A has not finished, this bus is not atomic-request. However, in step 4 the first transaction for A:GetS finishes, then the request from B appears on the bus in step 5, so it has atomic transaction property.

For simplicity, we call a bus that is atomic-request and atomic-transaction as ARAT bus. Similarly, NRAT means not atomic-request but atomic-transaction, and NRNT means neither atomic-request nor atomic-transaction.

We used a queue to model ARAT and NRAT buses. Whenever a core or the memory sends requests or sends data, a request/data message is added to the global queue. The bus will repeatedly select a message from the queue and broadcast it to everyone. Without other constraints, this model is NRNT because the data returned by the memory could pend in the queue for a long time after the bus chooses other requests to broadcast, so we model the properties in TLA+ as:

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size
00:00:18	70	5,475,949	2,042,329	0
00:00:04	40	698,312	272,845	39,692
00:00:01	0	1	1	1

```
atomicRequest ==
  /\ Cardinality(queue') <= 1
  /\ isRequest(bus) => (\A msg \in queue': isData(msg))

atomicTransaction ==
  /\ Cardinality({msg \in queue': isData(msg)}) <= 1
  /\ (Cardinality(queue') < Cardinality(queue) /\ isRequest(bus)) =>
    \A msg \in queue': isRequest(msg)
```

where **bus** means the broadcasting message, and **queue** means the pending ones. What the code means is that if - there is never more than 1 pending message - no request can be sent to the queue when there is another request broadcasting then the bus is ARAT; If - there is never more than 1 pending data message - data messages get broadcasted before requests then the bus is NRAT. A bus that has ARAT/NRAT properties doesn't necessarily satisfy those TLA+ statements, but it's general enough and it creates 3 kinds of buses to model check our MSI variants. Moreover, one can hardly write a single statement equal to the ARAT/NRAT property without getting into trouble, and we'll discuss this issue later.

Our NRNT bus model is different from the model above. In short, we used 2 FIFO queues, one for request and one for data, to model the NRNT bus, because the protocol may deadlock if requests and data share one bus.

1.1.3 Property and results

The property we want to check is simple: At any time the cache on all cores and the memory are coherent. The results show that all 3 variants of the MSI protocol passed the model checker under corresponding buses, but simpler protocol combined with non-atomic buses will yield counterexamples. The successful results are as follows:

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size
00:02:25	72	108,609,373	26,487,397	0
00:02:04	64	92,791,241	22,711,598	190,838
00:01:04	50	47,574,535	12,007,574	1,610,756
00:00:04	30	1,610,687	428,763	108,882
00:00:01	0	1	1	1

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size	
00:03:05	84	111,964,333	34,420,600	0	
00:02:04	62	75,350,654	24,262,277	1,829,977	
00:01:04	53	38,816,195	13,027,947	1,774,855	
00:00:04	32	1,161,769	404,952	75,204	
00:00:01	0	1	1	1	

1.2 Observations

1.2.1 PlusCal

Our first try was to model the protocol with PlusCal, which is a pseudocode-like language that can be translated into TLC+ code. The reason we try PlusCal first is that its syntax is close to programming languages, not math expressions, thus easy to learn and write. However, the expression power of PlusCal is lower than TLC+ and doesn't meet our requirements.

Initially, we modeled each core as a while loop process in PlusCal and used the **with** clause to select a core to run, but the compiler says that the **while** clause can't nest inside a **with** clause. As we must use **with** to select a core and **while** to model the repeating behavior of cores, we have to switch to plain TLA+. With a bit of investigation, we find that the issue is caused by the translation method from PlusCal to TLA+: The translator simulates the PlusCal program nearly on the machine code level with a pc counter. Within the **with** block, the pc counter can't change, but the while loop needs the pc counter to track. Anyway, it turns out that writing state machines with TLC+ is much easier than PlusCal because each transition can be nicely expressed in one statement.

1.2.2 Property formatting

It's natural to formulate the properties we want to check as something like $\Box \text{AtomicRequest} \Rightarrow \Box \text{Coherence}$; However, we found it ending in deadlock in our setups. For example, some state machine transition is unreachable under ARAT buses, when we use the NRNT bus and the model check the temporal formula above with the ARAT MSI variant, the model checker will run into paths that don't satisfy ARAT, invoking unreachable transitions in the protocol and cause deadlock.

The problem occurs with $\Box A \Rightarrow \Box B$ kinds of statements, where $\Box A$ is the assumption of the model, in our case ARAT/NRAT. Generally, the model checker can't know if $\Box A$ holds or not until running the state machine to the end, which means it can't quit early and stop exploring the paths that $\Box A$ is false; If the behavior of the model is only defined for paths that $\Box A$ holds, the model checker will induce undefined behaviors. On the other side, the model has no reason to consider transitions outside its designed assumptions, thus forcing us to incorporate the model assumptions into the overall state machine, rather than sharing a bigger state space and checking for $\Box A \Rightarrow \Box B$ kinds of formulas.

Recall our discussion about writing a single statement equal to the ARAT/NRAT property. Even if we come up with such a statement $\Box \text{ARAT}$ that holds if and only if the bus has the property, checking for $\Box \text{ARAT} \Rightarrow \Box \text{Coherence}$ will end up in a deadlock. To avoid deadlock, we must incorporate the property into explicit state machine transition rules for the bus, which means the bus we describe is highly concrete and couldn't represent any ARAT buses. In conclusion, it's hard, if even possible, to check "this MSI protocol works under **any** ARAT buses" without specifying the undefined behaviors.

One possible approach is to suppose that if the model deadlock then ARAT doesn't hold, so we can still use a general bus model, check for $\Box \text{ARAT} \Rightarrow \Box \text{Coherence}$, and discard those deadlock paths. But it has 2 problems too. First, the assumption itself is not obvious, thus requiring to prove or a derived model checking. Second, reaching deadlock is too late to realize ARAT doesn't hold, we much prevent deadlock beforehand. But preventing deadlock explicitly in the state machine in

turn means that the bus model must be ARAT thus not general enough.

In the end, we used `Spec == Init /\ [] [Next /\ atomicRequest]_vars` and model check for `[]Coherence`.

1.2.3 TLA+ as a language

Since all 3 variants share the basic MSI idea, we want to reuse code as much as possible and separate the protocol state machines with bus models if possible. However, we ended up copy paste a lot more than we intended to. The main reason is that TLA+ lacks polymorphism, which means we can't use an interface as a placeholder to model the high-level structure of the MSI protocol and fill the detailed implementation in variant-specific files.

The **UNCHANGED** clause of the TLA+ also impeded our code reuse effort. As we introduce more variables in a new variant, we must include it in every old function that doesn't change the new variable. If we write the **UNCHANGED** clause outside the function, it becomes the boilerplate code itself. We searched for the reason that the designer of TLA+ enforce the **UNCHANGED** clause, he said that by requiring explicitly listing unchanged variables, he could avoid many bugs about unintentionally changing a variable. However, some users suggest making this clause optional and identifying unchanged variables automatically.

1.3 References

- [1] Sorin D J, Hill M D, Wood D A. A primer on memory consistency and cache coherence[J]. Synthesis lectures on computer architecture, 2011, 6(3): 1-212.

Cam's Side

- **Timeline.** When referring to the first half and second half, the first half is before the Project Update, and the second half is after the Project Update.
During the first half, I was not able to make much progress. I stopped attempting to implement Rumur and instead attempted to help out Qiao in TLA. During the first half I spent a little while learning more about TLA as well as its abilities to simulate formal verification of cache protocols, and while doing so I was attempting to find other means of formal verification of cache protocols using C/C++, which led me to attempt to use SPIN.
I was using SPIN for the last week of the first half. I was attempting to figure out how to utilize it and had a lot of trouble finding the software online, which is when Professor Ivancic suggested I use software similar to Ubuntu. After the conversation, I decided I should attempt to use and install Ubuntu as a last-minute resort toward using Rumur. I was able to successfully get Ubuntu installed and started using Rumur a couple of weeks into the second half, which was around the beginning of December.
I spent the remaining 2 weeks learning about Murphi's language, how to use Murphi's language and Objective C with Rumur, and how to properly.

- **What was learned.**

```

7
8 type
9   state_status: enum { MODIFIED, SHARED, INVALID };
10  sttate: enum { q };
11
12 var
13   condition: array[sttate] of state_status
14
15 startstate begin
16   condition[q] := SHARED;
17 end
18
19 ruleset c: sttate do
20   rule "local read hit" condition[c] = SHARED | condition[c] = INVALID | condition[c] = MODIFIED ==> begin
21     condition[c] := SHARED;
22   end
23
24   rule "local write hit" condition[c] = SHARED | condition[c] = MODIFIED | condition[c] = INVALID ==> begin
25     condition[c] := MODIFIED;
26   end
27
28   rule "remote write/replacement" condition[c] = MODIFIED | condition[c] = SHARED ==> begin
29     condition[c] := INVALID;
30   end
31
32
33
34
35
36 end

```

Figure 1: MSI protocol code, no error.

```

electro49@The-Station: /mnt... x Windows PowerShell
End of the error trace.

=====
Status:
  1 error(s) found.

State Space Explored:
  2 states, 2 rules fired in 0s.
electro49@The-Station: /mnt/c/users/light/desktop/HMSW_cacheprotocol/Rumur/Can's Trials and Tests$ rumur MSI.m --output MSI.c.c
electro49@The-Station: /mnt/c/users/light/desktop/HMSW_cacheprotocol/Rumur/Can's Trials and Tests$ gcc -O3 MSI.c.c -mcs16 -o MSI_final
electro49@The-Station: /mnt/c/users/light/desktop/HMSW_cacheprotocol/Rumur/Can's Trials and Tests$ ./MSI_final
Memory usage:
  * The size of each state is 2 bits (rounded up to 1 bytes).
  * The size of the hash table is 65536 slots.

Progress Report:

=====
Status:
  No error found.

State Space Explored:
  3 states, 8 rules fired in 0s.
electro49@The-Station: /mnt/c/users/light/desktop/HMSW_cacheprotocol/Rumur/Can's Trials and Tests$

```

Figure 2: What a no-error message looks like

- ***Rumur/Murphi.*** Murphi and Rumur are tools for formally verifying concurrent systems. They are designed to automatically check a model of a system for errors and other issues, such as deadlocks, livelocks, and unbounded behavior. Murphi and Rumur use a modeling language that is similar to a programming language, allowing designers to specify a system at a high level of abstraction. The tool then automatically synthesizes the high-level specification into a low-level hardware description, which can be used to implement the system on a specific hardware platform.

Murphi and Rumur are particularly useful for verifying concurrent and hardware systems, such as distributed systems, communication protocols, and multi-threaded software. They are able to handle large and complex models efficiently, using techniques such as symbolic execution and counterexample-guided refinement to quickly and accurately check the model for errors and other issues. Murphi and Rumur are open source, which means that they are freely available to use and modify, and it has a wide range of applications in the field of formal verification.

The reason why I compare Murphi and Rumur is due to timing. Murphi was released in 1992 but discontinued. A lot of software packages implemented and/or continued Murphi code that assisted in their formal verification of concurrent and hardware systems. They are nearly not as popular as Murphi was (except for SPIN), so there is rarely any documentation for both Murphi and Rumur.

- ***Using Rumur.*** When given an MSI protocol, you can use Rumur to simulate its correctness. Based on how the data is created, you can see step by step of the state machine process as well.

Rumur is a program that is mainly used to provide feedback on the created state machine. It gives the option to provide extra printed information within the program to track the progress of the state machine, as well as provide run speed and status. We see Rumur as a very basic program that is quick and efficient at determining whether the state machine creates errors or not, the size of each states and its data usage, as well as states explored in an amount of time. Even with the heaviest of generated state machines run in under a second, but can often get to the double digits of bits in terms of the size of each state. Using the program, however, would require a lot of prior practice with Objective C as well as Murphi.

- **Encountered issues.** One of the biggest and main issues that happened with Rumur was the implementation process. Both Github and Ubuntu contained very old versions of Rumur, so it was extremely difficult to properly implement the software onto the computer. The reason why was because of the lack of resources that were found, especially in the first half.

1. **Anaconda.** Anaconda properly installed on the Windows computer. The CMake, GCC, and GMP packages did not have the proper implemented functions in order to properly install Rumur. Due to channel complications and many other issues, Anaconda no longer works on Windows. There may be another attempt to reinstall Anaconda on Windows for a final attempt. Anaconda did not properly install on the Mac
2. **Terminal/Powershell.** Windows was not able to properly install any of the packages needed to start the installation of Rumur in Powershell. Mac properly installed all of

the packages using Homebrew, but there were still multiple issues found with the package installation when cmake was ran in the "build" directory. Cam may go back to that to see if the issue can be fixed.

3. **MobaXterm.** MobaXterm was not able to install the proper packages on both Mac and Windows due to some language conflictions as well as some packages bashing with each other. Cam will see if that problem can be fixed in their free time.
4. **Visual Studio Code.** VSCode was not able to be properly utilized because of a few issues. VSCode may not have been properly installed on Windows, which caused some issues with installing and using packages in VSCode's terminal. Cam may go back to try VSCode on the Mac.
5. **PuTTY.** Putty was not able to be utilized on the Windows because Cam was not able to find a proper environment to start doing work. May go back to see if Cam can do some research or ask questions to find a proper environment for PuTTY. Cam will do that before attempting to use it on the Mac.

During the second half, the only issue that was being encountered was the lack of resources. Like mentioned before, Rumur does not have a lot of supporting resources due to its lack of publishing and lack of usage. The only issue that was being encountered was attempting to work off of example files that were provided in order to create a new MSI protocol state machine.

- **Unfinished work.** i would not say that there was no unfinished work. Just points in the project where I wish I had more time. I was successfully able to simulate an MSI program, and I was headed towards implementing an MSI program that used more states to see how much data it would use and how much time it would take. I unfortunately was not able to fit it into my schedule due to the amount of time that I spent trying to implement the software in the first place.

- **Limits and Advantages of using Rumur.**

- Rumur is limited mainly because it was built very close to a software that was discontinued. A lot of similarities cause one to have to go back to find Murphi resources, which are already very hard to find.
- Rumur is not an extremely visual tool. It is a simple tool which allows one to test for correctness as well as resources used for their state machine. Although it is a fast tool, it is not useful on a visual aspect, especially when using a language that is generally harder to understand.
- Although Rumur is not the best to use visually, it is a great tool to use in order to understand how to create a state machine and how to use model checking. It also is decently descriptive in terms of errors that you may have, such as deadlocks and impossible rules set in place, as seen below. Deadlocks are often caused due to mutual exclusion or recourse ordering, and it is important not to have them. Deadlocks can be prevented thorough careful design.

```

1 // ... test put statements get parsed correctly, the expected output is just chosen
2 // ... to match one of the array entries that is correlated with its implementing
3 // ... this feature, i.e. a first attempt at implementing iteration for array
4 // ... printing made the mistake of inaccurate indexing resulting in "[0]read".
5
6 type
7 state_status: enum { MODIFIED, SHARED, INVALID };
8
9 struct state {
10     state_status: state_status;
11     holder: enum { loc_read, loc_write, replacement };
12 };
13
14 var
15 condition: array[1024] of state_status;
16 x: state_status;
17 y: holder;
18
19 rule "local read hit" (y == loc_read & x == SHARED) | (y == loc_read & x == INVALID) | (y == loc_read & x == MODIFIED) --> begin
20     condition[i] := SHARED;
21 end
22
23 rule "local write hit" (y == loc_write & x == SHARED) | (y == loc_write & x == MODIFIED) | (y == loc_write & x == INVALID) --> begin
24     condition[i] := MODIFIED;
25 end
26
27 rule "remote write/replacement" (y == replacement & x == MODIFIED) | (y == replacement & x == SHARED) --> begin
28     condition[c] := INVALID;
29 end
30
31 startstate begin
32     x := SHARED;
33     y := loc_read;
34     put condition;
35
36 // ... testing the initial condition ...
37 put condition;
38
39 y := loc_read;
40 put condition;
41
42 y := loc_write;
43 put condition;
44
45 y := loc_read;
46 put condition;
47
48 y := loc_write;
49 put condition;
50
51 y := loc_read;
52 put condition;
53
54 y := loc_write;
55 put condition;
56
57 y := loc_read;
58 put condition;
59
60 y := loc_write;
61 put condition;
62
63 y := loc_read;
64 put condition;
65
66 y := loc_write;
67 put condition;
68
69 y := loc_read;
70 put condition;
71
72 y := loc_write;
73 put condition;
74
75 y := loc_read;
76 put condition;
77
78 y := loc_write;
79 put condition;
80
81 y := loc_read;
82 put condition;
83
84 y := loc_write;
85 put condition;
86
87 y := loc_read;
88 put condition;
89
90 y := loc_write;
91 put condition;
92
93 y := loc_read;
94 put condition;
95
96 y := loc_write;
97 put condition;
98
99 y := loc_read;
100 put condition;
101
102 y := loc_write;
103 put condition;
104
105 y := loc_read;
106 put condition;
107
108 y := loc_write;
109 put condition;
110
111 y := loc_read;
112 put condition;
113
114 y := loc_write;
115 put condition;
116
117 y := loc_read;
118 put condition;
119
120 y := loc_write;
121 put condition;
122
123 y := loc_read;
124 put condition;
125
126 y := loc_write;
127 put condition;
128
129 y := loc_read;
130 put condition;
131
132 y := loc_write;
133 put condition;
134
135 y := loc_read;
136 put condition;
137
138 y := loc_write;
139 put condition;
140
141 y := loc_read;
142 put condition;
143
144 y := loc_write;
145 put condition;
146
147 y := loc_read;
148 put condition;
149
150 y := loc_write;
151 put condition;
152
153 y := loc_read;
154 put condition;
155
156 y := loc_write;
157 put condition;
158
159 y := loc_read;
160 put condition;
161
162 y := loc_write;
163 put condition;
164
165 y := loc_read;
166 put condition;
167
168 y := loc_write;
169 put condition;
170
171 y := loc_read;
172 put condition;
173
174 y := loc_write;
175 put condition;
176
177 y := loc_read;
178 put condition;
179
180 y := loc_write;
181 put condition;
182
183 y := loc_read;
184 put condition;
185
186 y := loc_write;
187 put condition;
188
189 y := loc_read;
190 put condition;
191
192 y := loc_write;
193 put condition;
194
195 y := loc_read;
196 put condition;
197
198 y := loc_write;
199 put condition;
200
201 y := loc_read;
202 put condition;
203
204 y := loc_write;
205 put condition;
206
207 y := loc_read;
208 put condition;
209
210 y := loc_write;
211 put condition;
212
213 y := loc_read;
214 put condition;
215
216 y := loc_write;
217 put condition;
218
219 y := loc_read;
220 put condition;
221
222 y := loc_write;
223 put condition;
224
225 y := loc_read;
226 put condition;
227
228 y := loc_write;
229 put condition;
230
231 y := loc_read;
232 put condition;
233
234 y := loc_write;
235 put condition;
236
237 y := loc_read;
238 put condition;
239
240 y := loc_write;
241 put condition;
242
243 y := loc_read;
244 put condition;
245
246 y := loc_write;
247 put condition;
248
249 y := loc_read;
250 put condition;
251
252 y := loc_write;
253 put condition;
254
255 y := loc_read;
256 put condition;
257
258 y := loc_write;
259 put condition;
260
261 y := loc_read;
262 put condition;
263
264 y := loc_write;
265 put condition;
266
267 y := loc_read;
268 put condition;
269
270 y := loc_write;
271 put condition;
272
273 y := loc_read;
274 put condition;
275
276 y := loc_write;
277 put condition;
278
279 y := loc_read;
280 put condition;
281
282 y := loc_write;
283 put condition;
284
285 y := loc_read;
286 put condition;
287
288 y := loc_write;
289 put condition;
290
291 y := loc_read;
292 put condition;
293
294 y := loc_write;
295 put condition;
296
297 y := loc_read;
298 put condition;
299
300 y := loc_write;
301 put condition;
302
303 y := loc_read;
304 put condition;
305
306 y := loc_write;
307 put condition;
308
309 y := loc_read;
310 put condition;
311
312 y := loc_write;
313 put condition;
314
315 y := loc_read;
316 put condition;
317
318 y := loc_write;
319 put condition;
320
321 y := loc_read;
322 put condition;
323
324 y := loc_write;
325 put condition;
326
327 y := loc_read;
328 put condition;
329
330 y := loc_write;
331 put condition;
332
333 y := loc_read;
334 put condition;
335
336 y := loc_write;
337 put condition;
338
339 y := loc_read;
340 put condition;
341
342 y := loc_write;
343 put condition;
344
345 y := loc_read;
346 put condition;
347
348 y := loc_write;
349 put condition;
350
351 y := loc_read;
352 put condition;
353
354 y := loc_write;
355 put condition;
356
357 y := loc_read;
358 put condition;
359
360 y := loc_write;
361 put condition;
362
363 y := loc_read;
364 put condition;
365
366 y := loc_write;
367 put condition;
368
369 y := loc_read;
370 put condition;
371
372 y := loc_write;
373 put condition;
374
375 y := loc_read;
376 put condition;
377
378 y := loc_write;
379 put condition;
380
381 y := loc_read;
382 put condition;
383
384 y := loc_write;
385 put condition;
386
387 y := loc_read;
388 put condition;
389
390 y := loc_write;
391 put condition;
392
393 y := loc_read;
394 put condition;
395
396 y := loc_write;
397 put condition;
398
399 y := loc_read;
400 put condition;
401
402 y := loc_write;
403 put condition;
404
405 y := loc_read;
406 put condition;
407
408 y := loc_write;
409 put condition;
410
411 y := loc_read;
412 put condition;
413
414 y := loc_write;
415 put condition;
416
417 y := loc_read;
418 put condition;
419
420 y := loc_write;
421 put condition;
422
423 y := loc_read;
424 put condition;
425
426 y := loc_write;
427 put condition;
428
429 y := loc_read;
430 put condition;
431
432 y := loc_write;
433 put condition;
434
435 y := loc_read;
436 put condition;
437
438 y := loc_write;
439 put condition;
440
441 y := loc_read;
442 put condition;
443
444 y := loc_write;
445 put condition;
446
447 y := loc_read;
448 put condition;
449
450 y := loc_write;
451 put condition;
452
453 y := loc_read;
454 put condition;
455
456 y := loc_write;
457 put condition;
458
459 y := loc_read;
460 put condition;
461
462 y := loc_write;
463 put condition;
464
465 y := loc_read;
466 put condition;
467
468 y := loc_write;
469 put condition;
470
471 y := loc_read;
472 put condition;
473
474 y := loc_write;
475 put condition;
476
477 y := loc_read;
478 put condition;
479
480 y := loc_write;
481 put condition;
482
483 y := loc_read;
484 put condition;
485
486 y := loc_write;
487 put condition;
488
489 y := loc_read;
490 put condition;
491
492 y := loc_write;
493 put condition;
494
495 y := loc_read;
496 put condition;
497
498 y := loc_write;
499 put condition;
500
501 y := loc_read;
502 put condition;
503
504 y := loc_write;
505 put condition;
506
507 y := loc_read;
508 put condition;
509
510 y := loc_write;
511 put condition;
512
513 y := loc_read;
514 put condition;
515
516 y := loc_write;
517 put condition;
518
519 y := loc_read;
520 put condition;
521
522 y := loc_write;
523 put condition;
524
525 y := loc_read;
526 put condition;
527
528 y := loc_write;
529 put condition;
530
531 y := loc_read;
532 put condition;
533
534 y := loc_write;
535 put condition;
536
537 y := loc_read;
538 put condition;
539
540 y := loc_write;
541 put condition;
542
543 y := loc_read;
544 put condition;
545
546 y := loc_write;
547 put condition;
548
549 y := loc_read;
550 put condition;
551
552 y := loc_write;
553 put condition;
554
555 y := loc_read;
556 put condition;
557
558 y := loc_write;
559 put condition;
560
561 y := loc_read;
562 put condition;
563
564 y := loc_write;
565 put condition;
566
567 y := loc_read;
568 put condition;
569
570 y := loc_write;
571 put condition;
572
573 y := loc_read;
574 put condition;
575
576 y := loc_write;
577 put condition;
578
579 y := loc_read;
580 put condition;
581
582 y := loc_write;
583 put condition;
584
585 y := loc_read;
586 put condition;
587
588 y := loc_write;
589 put condition;
590
591 y := loc_read;
592 put condition;
593
594 y := loc_write;
595 put condition;
596
597 y := loc_read;
598 put condition;
599
600 y := loc_write;
601 put condition;
602
603 y := loc_read;
604 put condition;
605
606 y := loc_write;
607 put condition;
608
609 y := loc_read;
610 put condition;
611
612 y := loc_write;
613 put condition;
614
615 y := loc_read;
616 put condition;
617
618 y := loc_write;
619 put condition;
620
621 y := loc_read;
622 put condition;
623
624 y := loc_write;
625 put condition;
626
627 y := loc_read;
628 put condition;
629
630 y := loc_write;
631 put condition;
632
633 y := loc_read;
634 put condition;
635
636 y := loc_write;
637 put condition;
638
639 y := loc_read;
640 put condition;
641
642 y := loc_write;
643 put condition;
644
645 y := loc_read;
646 put condition;
647
648 y := loc_write;
649 put condition;
650
651 y := loc_read;
652 put condition;
653
654 y := loc_write;
655 put condition;
656
657 y := loc_read;
658 put condition;
659
660 y := loc_write;
661 put condition;
662
663 y := loc_read;
664 put condition;
665
666 y := loc_write;
667 put condition;
668
669 y := loc_read;
670 put condition;
671
672 y := loc_write;
673 put condition;
674
675 y := loc_read;
676 put condition;
677
678 y := loc_write;
679 put condition;
680
681 y := loc_read;
682 put condition;
683
684 y := loc_write;
685 put condition;
686
687 y := loc_read;
688 put condition;
689
690 y := loc_write;
691 put condition;
692
693 y := loc_read;
694 put condition;
695
696 y := loc_write;
697 put condition;
698
699 y := loc_read;
700 put condition;
701
702 y := loc_write;
703 put condition;
704
705 y := loc_read;
706 put condition;
707
708 y := loc_write;
709 put condition;
710
711 y := loc_read;
712 put condition;
713
714 y := loc_write;
715 put condition;
716
717 y := loc_read;
718 put condition;
719
720 y := loc_write;
721 put condition;
722
723 y := loc_read;
724 put condition;
725
726 y := loc_write;
727 put condition;
728
729 y := loc_read;
730 put condition;
731
732 y := loc_write;
733 put condition;
734
735 y := loc_read;
736 put condition;
737
738 y := loc_write;
739 put condition;
740
741 y := loc_read;
742 put condition;
743
744 y := loc_write;
745 put condition;
746
747 y := loc_read;
748 put condition;
749
750 y := loc_write;
751 put condition;
752
753 y := loc_read;
754 put condition;
755
756 y := loc_write;
757 put condition;
758
759 y := loc_read;
760 put condition;
761
762 y := loc_write;
763 put condition;
764
765 y := loc_read;
766 put condition;
767
768 y := loc_write;
769 put condition;
770
771 y := loc_read;
772 put condition;
773
774 y := loc_write;
775 put condition;
776
777 y := loc_read;
778 put condition;
779
780 y := loc_write;
781 put condition;
782
783 y := loc_read;
784 put condition;
785
786 y := loc_write;
787 put condition;
788
789 y := loc_read;
790 put condition;
791
792 y := loc_write;
793 put condition;
794
795 y := loc_read;
796 put condition;
797
798 y := loc_write;
799 put condition;
800
801 y := loc_read;
802 put condition;
803
804 y := loc_write;
805 put condition;
806
807 y := loc_read;
808 put condition;
809
810 y := loc_write;
811 put condition;
812
813 y := loc_read;
814 put condition;
815
816 y := loc_write;
817 put condition;
818
819 y := loc_read;
820 put condition;
821
822 y := loc_write;
823 put condition;
824
825 y := loc_read;
826 put condition;
827
828 y := loc_write;
829 put condition;
830
831 y := loc_read;
832 put condition;
833
834 y := loc_write;
835 put condition;
836
837 y := loc_read;
838 put condition;
839
840 y := loc_write;
841 put condition;
842
843 y := loc_read;
844 put condition;
845
846 y := loc_write;
847 put condition;
848
849 y := loc_read;
850 put condition;
851
852 y := loc_write;
853 put condition;
854
855 y := loc_read;
856 put condition;
857
858 y := loc_write;
859 put condition;
860
861 y := loc_read;
862 put condition;
863
864 y := loc_write;
865 put condition;
866
867 y := loc_read;
868 put condition;
869
870 y := loc_write;
871 put condition;
872
873 y := loc_read;
874 put condition;
875
876 y := loc_write;
877 put condition;
878
879 y := loc_read;
880 put condition;
881
882 y := loc_write;
883 put condition;
884
885 y := loc_read;
886 put condition;
887
888 y := loc_write;
889 put condition;
890
891 y := loc_read;
892 put condition;
893
894 y := loc_write;
895 put condition;
896
897 y := loc_read;
898 put condition;
899
900 y := loc_write;
901 put condition;
902
903 y := loc_read;
904 put condition;
905
906 y := loc_write;
907 put condition;
908
909 y := loc_read;
910 put condition;
911
912 y := loc_write;
913 put condition;
914
915 y := loc_read;
916 put condition;
917
918 y := loc_write;
919 put condition;
920
921 y := loc_read;
922 put condition;
923
924 y := loc_write;
925 put condition;
926
927 y := loc_read;
928 put condition;
929
930 y := loc_write;
931 put condition;
932
933 y := loc_read;
934 put condition;
935
936 y := loc_write;
937 put condition;
938
939 y := loc_read;
940 put condition;
941
942 y := loc_write;
943 put condition;
944
945 y := loc_read;
946 put condition;
947
948 y := loc_write;
949 put condition;
950
951 y := loc_read;
952 put condition;
953
954 y := loc_write;
955 put condition;
956
957 y := loc_read;
958 put condition;
959
960 y := loc_write;
961 put condition;
962
963 y := loc_read;
964 put condition;
965
966 y := loc_write;
967 put condition;
968
969 y := loc_read;
970 put condition;
971
972 y := loc_write;
973 put condition;
974
975 y := loc_read;
976 put condition;
977
978 y := loc_write;
979 put condition;
980
981 y := loc_read;
982 put condition;
983
984 y := loc_write;
985 put condition;
986
987 y := loc_read;
988 put condition;
989
990 y := loc_write;
991 put condition;
992
993 y := loc_read;
994 put condition;
995
996 y := loc_write;
997 put condition;
998
999 y := loc_read;
1000 put condition;
1001
1002 y := loc_write;
1003 put condition;
1004
1005 y := loc_read;
1006 put condition;
1007
1008 y := loc_write;
1009 put condition;
1010
1011 y := loc_read;
1012 put condition;
1013
1014 y := loc_write;
1015 put condition;
1016
1017 y := loc_read;
1018 put condition;
1019
1020 y := loc_write;
1021 put condition;
1022
1023 y := loc_read;
1024 put condition;
1025
1026 y := loc_write;
1027 put condition;
1028
1029 y := loc_read;
1030 put condition;
1031
1032 y := loc_write;
1033 put condition;
1034
1035 y := loc_read;
1036 put condition;
1037
1038 y := loc_write;
1039 put condition;
1040
1041 y := loc_read;
1042 put condition;
1043
1044 y := loc_write;
1045 put condition;
1046
1047 y := loc_read;
1048 put condition;
1049
1050 y := loc_write;
1051 put condition;
1052
1053 y := loc_read;
1054 put condition;
1055
1056 y := loc_write;
1057 put condition;
1058
1059 y := loc_read;
1060 put condition;
1061
1062 y := loc_write;
1063 put condition;
1064
1065 y := loc_read;
1066 put condition;
1067
1068 y := loc_write;
1069 put condition;
1070
1071 y := loc_read;
1072 put condition;
1073
1074 y := loc_write;
1075 put condition;
1076
1077 y := loc_read;
1078 put condition;
1079
1080 y := loc_write;
1081 put condition;
1082
1083 y := loc_read;
1084 put condition;
1085
1086 y := loc_write;
1087 put condition;
1088
1089 y := loc_read;
1090 put condition;
1091
1092 y := loc_write;
1093 put condition;
1094
1095 y := loc_read;
1096 put condition;
1097
1098 y := loc_write;
1099 put condition;
1100
1101 y := loc_read;
1102 put condition;
1103
1104 y := loc_write;
1105 put condition;
1106
1107 y := loc_read;
1108 put condition;
1109
1110 y := loc_write;
1111 put condition;
1112
1113 y := loc_read;
1114 put condition;
1115
1116 y := loc_write;
1117 put condition;
1118
1119 y := loc_read;
1120 put condition;
1121
1122 y := loc_write;
1123 put condition;
1124
1125 y := loc_read;
1126 put condition;
1127
1128 y := loc_write;
1129 put condition;
1130
1131 y := loc_read;
1132 put condition;
1133
1134 y := loc_write;
1135 put condition;
1136
1137 y := loc_read;
1138 put condition;
1139
1140 y := loc_write;
1141 put condition;
1142
1143 y := loc_read;
1144 put condition;
1145
1146 y := loc_write;
1147 put condition;
1148
1149 y := loc_read;
1150 put condition;
1151
1152 y := loc_write;
1153 put condition;
1154
1155 y := loc_read;
1156 put condition;
1157
1158 y := loc_write;
1159 put condition;
1160
1161 y := loc_read;
1162 put condition;
1163
1164 y := loc_write;
1165 put condition;
1166
1167 y := loc_read;
1168 put condition;
1169
1170 y := loc_write;
1171 put condition;
1172
1173 y := loc_read;
1174 put condition;
1175
1176 y := loc_write;
1177 put condition;
1178
1179 y := loc_read;
1180 put condition;
1181
1182 y := loc_write;
1183 put condition;
1184
1185 y := loc_read;
1186 put condition;
1187
1188 y := loc_write;
1189 put condition;
1190
1191 y := loc_read;
1192 put condition;
1193
1194 y := loc_write;
1195 put condition;
1196
1197 y := loc_read;
1198 put condition;
1199
1200 y := loc_write;
1201 put condition;
1202
1203 y := loc_read;
1204 put condition;
1205
1206 y := loc_write;
1207 put condition;
1208
1209 y := loc_read;
1210 put condition;
1211
1212 y := loc_write;
1213 put condition;
1214
1215 y := loc_read;
1216 put condition;
1217
1218 y := loc_write;
1219 put condition;
1220
1221 y := loc_read;
1222 put condition;
1223
1224 y := loc_write;
1225 put condition;
1226
1227 y := loc_read;
1228 put condition;
1229
1230 y := loc_write;
1231 put condition;
1232
1233 y := loc_read;
1234 put condition;
1235
1236 y := loc_write;
1237 put condition;
1238
1239 y := loc_read;
1240 put condition;
1241
1242 y := loc_write;
1243 put condition;
1244
1245 y := loc_read;
1246 put condition;
1247
1248 y := loc_write;
1249 put condition;
1250
1251 y := loc_read;
1252 put condition;
1253
1254 y := loc_write;
1255 put condition;
1256
1257 y := loc_read;
1258 put condition;
1259
1260 y := loc_write;
1261 put condition;
1262
1263 y := loc_read;
1264 put condition;
1265
1266 y := loc_write;
1267 put condition;
1268
1269 y := loc_read;
1270 put condition;
1271
1272 y := loc_write;
1273 put condition;
1274
1275 y := loc_read;
1276 put condition;
1277
1278 y := loc_write;
1279 put condition;
1280
1281 y := loc_read;
1282 put condition;
1283
1284 y := loc_write;
1285 put condition;
1286
1287 y := loc_read;
1288 put condition;
1289
1290 y := loc_write;
1291 put condition;
1292
1293 y := loc_read;
1294 put condition;
1295
1296 y := loc_write;
1297 put condition;
1298
1299 y := loc_read;
1300 put condition;
1301
1302 y := loc_write;
1303 put condition;
1304
1305 y := loc_read;
1306 put condition;
1307
1308 y := loc_write;
1309 put condition;
1310
1311 y := loc_read;
1312 put condition;
1313
1314 y := loc_write;
1315 put condition;
1316
1317 y := loc_read;
1318 put condition;
1319
1320 y := loc_write;
1321 put condition;
1322
1323 y := loc_read;
1324 put condition;
1325
1326 y := loc_write;
1327 put condition;
1328
1329 y := loc_read;
1330 put condition;
1331
1332 y := loc_write;
1333 put condition;
1334
1335 y := loc_read;
1336 put condition;
1337
1338 y := loc_write;
1339 put condition;
1340
1341 y := loc_read;
1342 put condition;
1343
1344 y := loc_write;
1345 put condition;
1346
1347 y := loc_read;
1348 put condition;
1349
1350 y := loc_write;
1351 put condition;
1352
1353 y := loc_read;
1354 put condition;
1355
1356 y := loc_write;
1357 put condition;
1358
1359 y := loc_read;
1360 put condition;
1361
1362 y := loc_write;
1363 put condition;
1364
1365 y := loc_read;
1366 put condition;
1367
1368 y := loc_write;
1369 put condition;
1370
1371 y := loc_read;
1372 put condition;
1373
1374 y := loc_write;
1375 put condition;
1376
1377 y := loc_read;
1378 put condition;
1379
1380 y := loc_write;
1381 put condition;
1382
1383 y := loc_read;
1384 put condition;
1385
1386 y := loc_write;
1387 put condition;
1388
1389 y := loc_read;
1390 put condition;
1391
1392 y := loc_write;
1393 put condition;
1394
1395 y := loc_read;
1396 put condition;
1397
1398 y := loc_write;
1399 put condition;
1400
1401 y := loc_read;
1402 put condition;
1403
1404 y := loc_write;
1405 put condition;
1406
1407 y := loc_read;
1408 put condition;
1409
1410 y := loc_write;
1411 put condition;
1412
1413 y := loc_read;
1414 put condition;
1415
1416 y := loc_write;
1417 put condition;
1418
1419 y := loc_read;
1420 put condition;
1421
1422 y := loc_write;
1423 put condition;
1424
1425 y := loc_read;
1426 put condition;
1427
1428 y := loc_write;
1429 put condition;
1430
1431 y := loc_read;
1432 put condition;
1433
1434 y := loc_write;
1435 put condition;
1436
1437 y := loc_read;
1438 put condition;
1439
1440 y := loc_write;
1441 put condition;
1442
1443 y := loc_read;
1444 put condition;
1445
1446 y := loc_write;
1447 put condition;
1448
1449 y := loc_read;
1450 put condition;
1451
1452 y := loc_write;
1453 put condition;
1454
1455 y := loc_read;
1456 put condition;
1457
1458 y := loc_write;
1459 put condition;
1460
1461 y := loc_read;
1462 put condition;
1463
1464 y := loc_write;
1465 put condition;
1466
1467 y := loc_read;
1468 put condition;
1469
1470 y := loc_write;
1471 put condition;
1472
1473 y := loc_read;
1474 put condition;
1475
1476 y := loc_write;
1477 put condition;
1478
1479 y := loc_read;
1480 put condition;
1481
1482 y := loc_write;
1483 put condition;
1484
1485 y := loc_read;
1486 put condition;
1487
1488 y := loc_write;
1489 put condition;
1490
1491 y := loc_read;
1492 put condition;
1493
1494 y := loc_write;
1495 put condition;
1496
1497 y := loc_read;
1498 put condition;
1499
1500 y := loc_write;
1501 put condition;
1502
1503 y := loc_read;
1504 put condition;
1505
1506 y := loc_write;
1507 put condition;
1508
1509 y := loc_read;
1510 put condition;
1511
1512 y := loc_write;
1513 put condition;
1514
1515 y := loc_read;
1516 put condition;
1517
1518 y := loc_write;
1519 put condition;
1520
1521 y := loc_read;
1522 put condition;
1523
1524 y := loc_write;
1525 put condition;
1526
1527 y := loc_read;
1528 put condition;
1529
1530 y := loc_write;
1531 put condition;
1532
1533 y := loc_read;
1534 put condition;
1535
1536 y := loc_write;
1537 put condition;
1538
1539 y := loc_read;
1540 put condition;
1541
1542 y := loc_write;
1543 put condition;
1544
1545 y := loc_read;
1546 put condition;
1547
1548 y := loc_write;
1549 put condition;
1550
1551 y := loc_read;
1552 put condition;
1553
1554 y := loc_write;
1555 put condition;
1556
1557 y := loc_read;
1558 put condition;
1559
1560 y := loc_write;
1561 put condition;
1562
1563 y := loc_read;
1564 put condition;
1565
1566 y := loc_write;
1567 put condition;
1568
1569 y := loc_read;
1570 put condition;
1571
1572 y := loc_write;
1573 put condition;
1574
1575 y := loc_read;
1576 put condition;
1577
1578 y := loc_write;
1579 put condition;
1580
1581 y := loc_read;
1582 put condition;
1583
1584 y := loc_write;
1585 put condition;
1586
1587 y := loc_read;
1588 put condition;
1589
1590 y := loc_write;
1591 put condition;
1592
1593 y := loc_read;
1594 put condition;
1595
1596 y := loc_write;
1597 put condition;
1598
1599 y := loc_read;
1600 put condition;
1601
1602 y := loc_write;
1603 put condition;
1604
1605 y := loc_read;
1606 put condition;
1607
1608 y := loc_write;
1609 put condition;
1610
1611 y := loc_read;
1612 put condition;
1613
1614 y := loc_write;
1615 put condition;
1616
1617 y := loc_read;
1618 put condition;
1619
1620 y := loc_write;
1621 put condition;
1622
1623 y := loc_read;
1624 put condition;
1625
1626 y := loc_write;
1627 put condition;
1628
1629 y := loc_read;
1630 put condition;
1631
1632 y := loc_write;
1633 put condition;
1634
1635 y := loc_read;
1636 put condition;
1637
1638 y := loc_write;
1639 put condition;
1640
1641 y := loc_read;
1642 put condition;
1643
1644 y := loc_write;
1645 put condition;
1646
1647 y := loc_read;
1648 put condition;
1649
1650 y := loc_write;
1651 put condition;
1652
1653 y := loc_read;
1654 put condition;
1655
1656 y := loc_write;
1657 put condition;
1658
1659 y := loc_read;
1660 put condition;
1661
1662 y := loc_write;
1663 put condition;
1664
1665 y := loc_read;
1666 put condition;
1667
1668 y := loc_write;
1669 put condition;
1670
1671 y := loc_read;
1672 put condition;
1673
1674 y := loc_write;
1675 put condition;
1676
1677 y := loc_read;
1678 put condition;
1679
1680 y := loc_write;
1681 put condition;
1682
1683 y := loc_read;
1684 put condition;
1685
1686 y := loc_write;
1687 put condition;
1688
1689 y := loc_read;
1690 put condition;
1691
1692 y := loc_write;
1693 put condition;
1694
1695 y := loc_read;
1696 put condition;
1697
1698 y := loc_write;
1699 put condition;
1700
1701 y := loc_read;
1702 put condition;
1703
1704 y := loc_write;
1705 put condition;
1706
1707 y := loc_read;
1708 put condition;
1709
1710 y := loc_write;
1711 put condition;
1712
1713 y := loc_read;
1714 put condition;
1715
1716 y := loc_write;
1717 put condition;
1718
```

- It is able to cleanly implement MSI into its software. Using Modified, Invalid, and shared, it can be implemented in multiple different ways. The best thing about Rumur is that it is rather flexible and you can put in very original and unique code ideas and still be able to properly implement what you are looking for.
- As listed, a major advantage of Rumur is its ability to use limited resources in both time and bits. It is as flexible as any other coding language as well. Although it is not visual, it is very easy to implement hardware designs, including cache protocols, into the software for simulation. It is a good tool to give a description of errors that may be within the code. Other than finding an error, it is not very descriptive besides that. It seems to be software that is simply used to determine the efficiency and correctness of a state machine.

- **Sources.**

- <https://github.com/Smattr/rumur>
- <https://manpages.ubuntu.com/manpages/focal/man1/rumur.1.html>
- <https://www.eecs.umich.edu/courses/eecs570/discussions/w22/murphi.html>