Static power consumption analysis of a simple router

Dennis Brentjes Raoul Estourgie

January 17, 2014

Supervisor
Marko van Eekelen

Abstract

There has not been a lot of study in the subject of power consumption reduction through code optimization. A new method has been introduced by Paolo Parisen Toldin et al [6] by defining a Hoare logic for energy consumption analysis which is parametric with respect to the hardware it controls. This means that you only need to know the specific components and its power consumption of a system to analyse the interaction of software and hardware. This paper describes the implementation process of this model on a TP-Link router, and uses the model to analyse and compare two pieces of code that are run on the machine. Our findings in this paper are that the power analysis method works. However to create a model one needs to have in depth knowledge of the hardware, which normally is only present at the producers of the hardware. We also express our concerns about the applicability of the proposed method in a much larger setting.

1 Introduction

The electronic industry is at a point where they can produce small and very powerful processors. This has stimulated the production of mobile devices world wide. Battery storage of mobile devices is limited because they need to be lightweight. Low power consumption of hardware and battery optimization are already researched thoroughly by the scientific community. The hardware is controlled by software, this means that the hardware still can consume more than intended because of wrong use of software. However, the research on low energy consuming software is still in its infancy. Currently the only way to get a fair measurement of the energy consumption of software is by using specialized expensive hardware. This will only give you a rough estimate on how the software matches with the current processor.

There has been some work related to static analysis of energy consumption. In a paper by S. Kerrison et al [3] a method to analyse the generated assembly for a fixed piece of hardware. They construct a energy consumption model for the particular assembly and transform an program from assembly to their so called CiaoPP intermediate representation. Then they can use their Assembly level models to get an energy consumption estimate. Real life measurements support their analytical findings.

Paolo Parisen Toldin et al [6] uses a higher level of abstraction to solve the same problems. They implemented a Hoare logic for energy consumption analysis which is parametric with respect to the hardware it controls. This method is state of the art as the method was just finished 6 months ago at the time of writing. Therefore it has yet to be tested in practice. In this paper, a model is created of an existing TP-Link router according to the definitions given by aforementioned paper. This model is used to compare and evaluate the efficiency of two different implementations of the same algorithm that this router could use in daily practice.

This paper will investigate if the analysis is comprehensible and intuitive. We should not have to know why it works and why it is correct to be able to use the technique. That is one of the major things to be tested by applying the technique in a real life situation.

The next big question is to construct a real model of a real piece of hardware. Is there enough information publicly available to at least make some rough estimates about power consumption and how easy is it to read and use this information with this technique.

Another issue to address is finding a real piece of code simple yet realistic enough to rewrite into a simple language on which the analysis can work.

In this paper we are first going to describe the steps needed to apply the semantic analysis. First we will describe the analysis method itself as described [6]. And give a small example as how to use it in section 2. We then create a model corresponding to a real life router in section 3. We show how we managed to create a realistic model from the information available to us and on which aspects the model might lack in precision. In section 4 we describe the software used to analyse the hardware with our analysis technique and the results that are derived from the analysis. These intermediate results will be used in section 5 to plot the energy consumption of the whole system. And finally we discuss the results and viability of this method in section 6.

2 A hoare logic for energy aware semantics

The method in the paper by Paolo Parisen Toldin et al [6], describes a static analysis method for analysing power consumption. It uses natural deduction to infer the power cost of a piece of software. For this it uses a set of constants that are specific for the system being analysed. These constants denote for each interesting component in the system how much power a certain action takes. For instance, in the Wireless Sensor Node example given by the aforementioned paper an interesting component would be the radio receiver/sender. From this component you

would need to know how much energy a send and receive consumes. One would also need the amount of energy the radio sender/receiver needs to be in operating condition, with other words the standby or passive energy usage. A component can even change between these active and passive states, therefore the energy analysis must be aware of these transitions.

This is why each of these components also have a local state. The state does not need to contain programming variables. These variables can actually represent physical states of a component as well as program variables used by the software. So the Radio can either be in the *on* or *off* state. While in the off state you cannot call certain operations and it will use less passive power. Note that components generally can not use no energy as they have to be able to listen for state changes to become active again, which in turn costs energy. Each component in the model of your hardware contains a state which in turn contains constants for power consumption of each operation you can do on that component.

However, The CPU is a special component. The CPU must exists in each model and must define a certain set of energy consumption constants. The CPU is mandatory as this executes the code on the machine and the constants are needed to successfully analyse the power consumption of all the statements. For instance we need to know how long an integer operation takes and how much power it consumes in this time. This depends on the arithmetic operation constant that needs to be defined. The other constants that need to be defined are related to control structures. These represent the while and if overhead induced by the hardware.

Now that we have defined the constants needed to analyse our source code we need a strategy to come to a sound conclusion. As we do static analysis we don't have any runtime information. This means we have to overestimate our loop iterations and branches. So we basically analyse all the code and in case of a branch we choose the branch that costs the most energy. In case of the while we try to estimate the number of iterations and multiply the energy consumption by the loop body by the number of estimated loop iterations. To keep the analysis sound we always need to make an overestimation of the amount of loop iterations. Therefore we cannot make any assumptions on what is common in the runtime of the software and always make the overestimation.

The application of this method for a wireless sensor node can be found in the paper by Paolo Parisen Toldin et al [6]. Proof of the correctness of the system is given by the technical report [5]. Their example is smaller than ours and might be easier to understand. Their paper also gives a more complete run-down of the analysis method. We just repeat the basics, as our goal is to put this method in to practice, but for a better understanding of the analysis and for explanation of some of the subtleties we refer to these two papers.

2.1 The While language

The programming language that is used for our analysis is a so called While language. It is a fictional language with simple control structures, the hardest being a "while", and simple semantics. These types of language are often used in semantic analysis because they are so simple. Later when the method is proven sound with these languages you can slowly start to expand the language until you can prove equivalence with another language and you can apply your analysis on that real-life language.

This specific "While" language is set-up that there are no global variables, the only type is an "int" and parameters are passed by-value. This means functions in our language do not have side-effects on the program state. Recursion is also not allowed because this would require infinite proof trees while analysing it and this is impossible to do.

$$
\begin{array}{rcl}
c \in Const & = & n \in \mathbb{N} \mid i_0 \in \mathbb{N} \mid i_1 \in \mathbb{N} \mid \ \ldots \\
X \in Var & = & X_1 \mid X_2 \mid X_3 \mid \ \ldots \\
rf \in RankingFun & = & c \mid X \mid e_1 \odot e_2 \\
e \in Expr & = & c \mid X \mid X = e \mid e_1 \ \boxdot e_2 \mid C_i \ :: f(args) \mid f(args) \\
S \in Statement & = & \textbf{skip} \mid S_1; S_2 \mid e \mid \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if} \\
 & & \mid \textbf{while } e \textbf{ do } S \textbf{ end while} \\
fundef \in Func & = & \textbf{function } name(args)\ S \textbf{ return } X; \textbf{ end function} \\
p \in Prog & = & \epsilon \mid fundef\ p
\end{array}
$$

Where $\odot \in \{+, -, *\}$ *and* $\boxdot \in \{+, -, *, >, \geq, \equiv, \not\equiv, \leq, <, \wedge, \vee\}$

2.2 Energy-aware Semantics

The semantics are an important part of our research. It is a set of rules which updates states according to their current program statement. This means that they need to update timestamps, register component usage and store variables. This is done by constructing a semantic proof tree of both algorithm implementations. These proof trees can be found in the appendix for reference but the resulting tables, that can be found in section 4, are probably more interesting. The tables are the processed proof trees. they can be obtained by instantiating all the symbolic values in the tree with the real energy consumption constants of each component.

Below is an example of the semantic analysis. We use an assignment as example: $X_3 = 0$, which is one of the easiest expressions to process in the proof tree. But before we do that we need to explain the hoard logic involved. Hoare logic is just a natural deduction with pre- and post conditions. So preceding is a triple which contains the precondition and trailing after a node is a post condition triple. These triples contain the following things.

- Γ which is the component state, this state will be updated every time a component will be used. The CPU is also a component, so with every expression or statement the CPU will also update Γ.
- t This is the current time stamp, which is necessary to calculate the idle time consumption of component models. Component time stamps are updated every expression or statement.
- ρ is updated every time a variable is assigned and is related to the ranking function.

The intuition behind these proof trees is as follows. From the precondition and the following statement we can derive the a postcondition. This postcondition becomes the precondition for the next statement. When we analyse the last statement we have a postcondition for the whole program and that triple contains the energy consumption of the whole program.

$$
\dfrac{\dfrac{}{\{\Gamma_1; t_1; \rho_1\}\ \ 0\ \ \{\Gamma_1; t_1; \rho_1\}}\ N \qquad \Gamma_2 = \Gamma_1[C_{cpu} :: \mathfrak{e} += C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_1; t_1; \rho_1\}\ \ X_3 = 0\ \{\Gamma_2; t_2 = t_1 + C_{cpu} :: \mathfrak{T}_a; \rho_2\}}\ A
$$

This example illustrates the assignment of a constant to a variable. This means that the Assignment (A) rule needs to be applied. Afterwards the N rule is applied, this is because of the assignment of a constant. It could also have been the assignment of a component which would have required to update the component state via the CF rule. The A rule makes sure the Γ state and t are updated. This is illustrated separately to avoid confusion in our tree.

2.3 A simple example

We will illustrate the steps with a simplified example from [5]. For our example we have modelled a small temperature sensor. We will be applying the following steps which will clarify what we will do on a bigger scale with the router we will model.

- Creating component model
- Take a small algorithm
- Create a semantic tree for the algorithm

2.3.1 Modelling

The temperature sensor consists of two components: the CPU and a temperature sensor. We will give any units for this example as it is not important to understand the method and we do not have concrete information about such a system to reliably construct a realistic model with the right values and units.

Processor The component C_{cpu} is part of every system as it needs to execute the algorithm you are analysing in conjunction with this these components. The model used in this example is given in the table. It assumes a stateless CPU which does not consume energy over time. Furthermore, since there is no state, the CPU model consists of a set of constants only, extended with the default values for the other model elements.

Constants	$C_{cpu} :: \mathfrak{T}_e = 10, C_{cpu} :: \mathfrak{T}_a = 5, C_{cpu} :: \mathfrak{T}_w = 25, C_{cpu} :: \mathfrak{T}_i te = 25,$ $C_{cpu} :: \mathfrak{E}_e = 10, C_{cpu} :: \mathfrak{E}_a = 5, C_{cpu} :: \mathfrak{E}_w = 25, C_{cpu} :: \mathfrak{E}_i te = 25$

Sensor Component model C_s represents the sensor. The sensor itself implements a simple function: m, which takes a measurement. Therefore, there is also a single time-usage constant and a single energy usage constant. The sensor cannot be turned off and has no state. Therefore, it has a constant power-draw, which is set to 3 in the model.

Constants	$C_s :: \mathfrak{T}_m = 10, C_s :: \mathfrak{E}_m = 40$
Power draw	$C_s :: \phi(-) = 3$

2.3.2 The algorithm

The algorithm is a very small example to just illustrate the concept of the technique. The temperature sensor will constantly check the temperature until it is above the temperature given in X_2.

```
1 X1 = 0;
2 X2 = 20;
3 while(X1 < X2) do
4         X1 = Cs.m();
5 end while
```

Listing 1: example implementation

2.3.3 Semantic analysis

In this subsection we will analyse this algorithm by constructing the proof tree. The construction of the proof three will be done according to the semantic rules shown in the previous section. The values of the hoare triplets can be calculated from this proof tree. These values are then used to calculate a total energy consumption degree. This can be estimated by taking the upper bound of the energy usage whenever there is a choice to make in the analysis path. The semantic tree can be found in Appendix A.1

2.4 Applying the model to the tree

In this next section we will use the tree to see how our energy consumption state is constructed at any given moment. Meanwhile we will substitute all the abstract values with the actual values in our energy consumption model. The results will be presented in a table because this is the most compact way to visualize all the information.

n	t_n	$\mathfrak{e}_s in\ \Gamma_n$	$\mathfrak{e}_{cpu}\ in\ \Gamma_n$
1	0	0	0
2	$0 + 5 = 5$	-	$0 + 5 = 5$
3	$5 + 5 = 10$	-	$5 + 5 = 10$
4	$10 + 10 = 20$	-	$10 + 10 = 20$
5	$20 + 25 = 45$	-	$20 + 25 = 45$
6	$45 + 10 = 55$	$40 + 35 * 10 + \rho_3(X_1) * 10 * 45$	-
7	$45 + \rho_3(X_1) * 15 = 60$	$390 + \rho_3(X_1) * 10 * 45$	$45 + \rho_3(X_1) * 5$

2.5 The results

Because there is no analytical upper bound we cannot give a straight answer to how much power this snippet op code will consume however we can parametrize over the number of iterations. So the next graph shown in figure 1 will show how much power this piece of software uses on the described piece of hardware for the number of iterations specified.

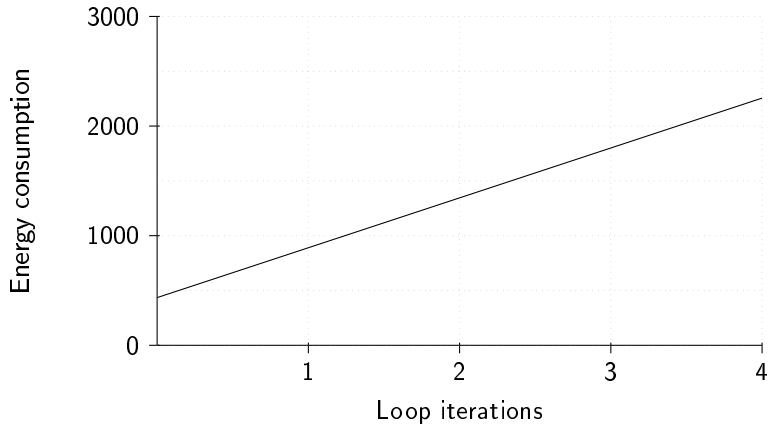

Figure 1: The amount of energy used in our example

Note that loop iterations are of course discrete values so the lines are not actually correct, but we will use the lines later in our actual results to show that the functions resulting from the analysis do not run parallel however they may seem so when only the dots are shown.

3 Modelling a router

As an example for our analysis we are modelling a small router. It is the TP-Link TL-WR703N. This router has lots of specification data available for each of its components. The analysis of this router would be much more complicated if not for Michael Stempin `http://squonk42.github.io/TL-WR703N/`. He reconstructed the schematic of this router. Kean electronics in conjunction with Sydney Hacker space `http://www.kean.com.au/oshw/WR703N/` compiled a set of technical specifications when tearing down one of the routers for each of the following components.

- CPU
- RAM
- Wifi transmitter/receiver

There are a few parts in the router which power consumption is constant (i.e. fuses) and cannot be influenced by software. These are not included in our model.

The model we use can be found in Tables 1 to 4. The rest of this section will be explaining the origins of these values in detail, if you want to see the application of this model with the technique described in the paper by Paolo Parisen Toldin et al [6] please refer to Section 4.

	$\mathfrak{T}$ $(10^{-6}s)$	$\mathfrak{E}$ $(10^{-9}J)$
expression	0.0275	1.1
assignment	0.0275	1.1
ite	0.055	2.2
while	0.055	2.2

Table 1: CPU power consumption and timing model

	$\mathfrak{T}$ $(10^{-6}s)$	$\mathfrak{E}$ $(10^{-9}J)$
get_port(X)	0.0325	7.1
check_ports(X)	X * 0.875 + 0.055	X * 9.3 + 2.2
store(X)	0.005	5.75

Table 2: RAM power consumption and timing model

	$\mathfrak{T}$ $(10^{-6}s)$	$\mathfrak{E}$ $(10^{-9}J)$
send(X)	1.55	1.0164
recv()	1.50	0.2838

Table 3: Wifi power consumption and timing model

	$\mathfrak{P}$ (W)
CPU	0.0
Wifi	0.82
RAM	0.7375

Table 4: Idle power consumption

3.1 CPU

The CPU is a MIPS32 24Kc processor that can operate up to 400MHz [1, page 21]. We assume the processor runs at full speed in the rest of our report. We also assume that a higher processor frequency will consume more power. If both assumptions hold we can overestimate CPU energy cost by assuming the CPU runs at full speed in this router. We use this assumption because the specification did not state at which frequency the CPU runs in this specific piece of hardware.

Please note that even if the assumption that running on a higher frequency consumes more power is correct, it is not certain that lowering the CPU frequency will save energy for the whole system. Other components might have a higher running or idle costs and they would supersede the extra cost of running the CPU at 400MHz when waiting for the CPU to catch up. This is exactly the reason that we need to use this type of analysis, to know for certain if this is the case. You might even want to parametrize it over the scalable frequencies of your processor to find the most optimal setup

For estimating the power consumption of the CPU we had to go by the manufactures claims. They claim that the processor uses *0.10 mW/MHz* on the product website `http://www.imgtec.com/mips/mips-32-24k.asp`. This would translate to $0.1 * 10^{-9}$ Joules per Tick as $0.1mW/Mhz = 0.1mW/Mhz * 400Mhz = 40mW$. *Energy per tick* $= 40mW * \frac{1}{400*10^6hz} = 0.1^{-9}J$. We need the energy per tick in our next calculation.

The only further information we have comes from the CPU specification. The specification states the CPU pipeline has a maximum of *11* stages [4][page 4]. Whether or not all of these 11 stages are actually executed is non trivial so we just overestimate. This means we use all 11 stages for all operations in our model.

This leaves us with simple mathematics to get the energy consumption of some of the simple tasks. One operation will cost: $1.1 * 10^{-9}$ Joules, and take $0.0275 * 10^{-6}s$. Both an expression and assignment fall under this category and therefore use the same energy consumption constant. This does mean however that we assume we have infinitely many registers.

We do this because we could either consider using a fixed amount of registers and keep track of how many registers were being used and store back values that could not be held by registers any more, or assume infinitely many registers. Keeping track of register allocation would become too difficult to manage both within the hoare logic but also when deciding what values to allocate to registers at any given time. We would have to come up with additional semantic objects and rules to handle this new information. This is a research topic on itself and not part of the problem we are trying to solve. So we assume infinitely many registers but try to do some realistic reads and writes to the ram to simulate a realistic scenario. These manual stores and reads to the RAM will be discussed in section 3.2.

This concludes all necessary information for our CPU model. But this processor has a special unit for integer multiplication and division. Its an extra pipeline that comes on top of the standard pipeline we just discussed.

This second pipeline takes the *size of the integer* $+1$ extra tick except for the 32 bit integer which takes the full 32 cycles.

	$\mathfrak{T}$ $(10^{-6}s)$	$\mathfrak{E}$ $(10^{-9}J)$
int_8	0.05	(1.1 + 0.9) = 2.0
int_{16}	0.07	(1.1 + 1.7) = 2.8
int_{24}	0.09	(1.1 + 2.5) = 3.6
int_{32}	0.1075	(1.1 + 3.2) = 4.3

Table 5: Integer multiplication and division model

3.2 RAM

We decided to model the RAM of this router. This was challenging to do because most system include some sort of caches and we decided to ignore them for the coming analysis. The RAM module is the Zentel A3S56D40FTP. The router uses its RAM to store message buffers from the wireless chip and therefore we need to take into account the RAM when processing wireless messages. For this we use the methods *store get_port* and *check_port* domain specific RAM interaction functions.

The RAM is also a volatile type of memory, so it uses energy over time. RAM therefore always consumes a constant rate of power in idle mode and consumes a bit more when data needs to be read or stored.

The RAM specifics we need to estimate power-consumption can be found in the specifications and are listed below. Clock speed of the RAM is a choice made of possible modes described in the technical specification [8, Page 5]. The rest of the values are estimated from table found in that same technical specification [8, Page 19]

- 200 MHZ
- 2.5 Volt
- Operating Idle mode = 295mA
- Read + 180mA
- Write + 165mA

Idle mode will be a constant power-consumption. Read or Write requires additional amperage and consequently more energy. The next calculations will give us a rough estimate of the consumptions. But unfortunately RAM is not a component that can be described on its own. There is a bus that takes care of communication with the CPU. There also needs to be some kind of synchronization between the CPU and ram cycles. All this information is impossible to fit inside the current model, but like we stated earlier we are not going to add these capabilities to the model as it is out of scope for this research. So we need to make a couple of assumptions.

First assumption the RAM reads and writes take exactly one CPU cycle. This is very unrealistic as this implies that the ram is always in sync with the CPU and that the internal circuitry of the ram is small enough to allow for all of the information to be in the right place in one CPU cycle.

The second assumption is that no time is lost between sending the read or write request and the Ram actually handling the request. also this is very unrealistic as the ram and CPU are relatively far apart.

We need these two assumptions to keep track or our timestamps during our analysis. As we can add the time of one CPU cycle to our time stamp whenever we use the ram.

$$Idlemode = 2.5 * 0,295 = 0,7375\ WATT$$
$$Readmode = \frac{2.5 * 0,475}{(200 * 10^6)} = 5,9375 * 10^{-9}\ Joules\ per\ tick$$
$$Writemode = \frac{2.5 * 0,460}{(200 * 10^6)} = 5,75 * 10^{-9}\ Joules\ per\ tick$$

The convenience functions store, *check_ports* and *get_port* are derived from these values. The *check_ports* and *get_port* also have some incidental energy usage that should be counted towards the CPU. For instance when using *get_port* you need some pointer arithmetic to access an address. This is done by the CPU but is added to the RAMs energy usage because the RAMs read operation is much more expensive than the pointer arithmetic. So the *store* function maps one on one with the write mode of the RAM. The *check_ports* is a read with two CPU expressions times the argument and while overhead. While the *get_port* is a read with one CPU expression. For the end results of these calculations please refer to Table 2

3.3 Wifi transmitter/receiver

The specification of the AR9331 Wifi module [1, page 314 - table 7-11] only specifies the power usage all the components involved in sending and receiving Wifi signals. These include extra components are listed below. They are necessary for normal operation of the Wifi module so are included in our analysis.

PA Power Amplifier: This component converts a low energy signal to a high energy signal that can sent by an antenna. The normal signal would be to weak.

LNA Low Noise Amplifier: Amplifies a signal received from the antenna. It is usually placed close to the detection device to minimize impact of signal loss by cable resistance.

LDO Low Drop Out Regulator: A LDO transforms a higher possibly unstable voltage in to a lower but much more stable output voltage. Needed by the antenna to minimize noise.

Another choice to be made was the operating mode. Wifi can operate in multiple modes and each of these modes consume different amounts of energy.

802.11b The IEEE 802.11b Wifi standard.

802.11g The IEEE 802.11g Wifi standard.

HT20 Part of the IEEE 802.11n standard. It uses a 20MHz wide band just like 802.11b/g. This interferes less with other networks than his HT40 counterpart as it ranges over less of the other networks bandwidth.

HT40 Part of 802.11n standard. It uses a 40MHz wide band, therefore is much faster than it's HT20 counterpart.

We chose for the HT20 mode as seems to be the most popular these days. This gives us the following power consumption estimates from the technical specification.

Transmission	$3.3\ Volt * 308mA = 1.0164\ W$
Reception	$3.3\ Volt * 86mA = 0.2838\ W$

There is one more thing we need to know about the Wifi module. We need the timing data of receiving and sending messages. This is not trivial and very chipset/antenna dependent. We found some similar research on this subjects and used their results for a rough estimate for our own model.

They concluded that the time it takes to send or receive a message is dependent on the size of the packet. So we had to make an estimate of the average packet. With help of [2] we determined that a 750 byte packet would be average. And our corresponding timings in Table 3 was based on their timings.

We are aware that these estimates are weak, but finding very realistic values for these types of systems is a research on it's own. So it's out of scope for this research paper and therefore these estimates will be used. one can always re-analyse our tree with an adapted model to compensate for the estimates when more accurate estimates become available.

4 Power consumption analysis

In this section we will analyse the power consumption of 2 implementations of an algorithm. The algorithm we implemented is loosely based on "KnockD" `http://www.zeroflux.org/projects/knock/`. KnockD is the digital equivalent of a secret handshake. This piece of software allows for an action on very specific packets received on some ports. It actually intercepts the packets on a lower layer in such a way that the packets are inspected even when the destination port is closed. This application is shipped in the package manager of Open-Wrt `https://openwrt.org/` which is an open router firmware.

The implementations in listings 2 to 3 are stripped down versions of this software, because we have no means to analyse big programs quickly. The two tables resulting from the analysis of the proof trees are places after the listings. Table 6 for implementation 1 and Table 7 for implementation 2. These Tables describe the energy consumption found for each component. The proof trees from which the tables are derived can be found in the appendix A.2 for implementation 1 and appendix A.3 for implementation 2.

4.1 Implementation 1

Both implementations will check if the incoming sequence of packets matches a predefined sequence. This is done by receiving packets, querying their port number (this is called the knock) and checking if it matches the expected port number. The location of the expected input is somewhere in the ram and we can query that information with specialized functions.

The first implementation will try to receive each of the port knocks and analyse them one at a time and breaking the loop early when a unexpected port is received. But because we analyse the upper bound energy use we only analyse the happy flow trough this program. So only the flow where all iterations have been done.

```
1  X4 = C_r.get_seq_len();
2  X3 = 0;
3  while(X3 < X4) do
4          X1 = C_w.Receive();
5          X2 = C_r.Store(X1);
6          X5 = C_r.get_port(X2);
7          if (C_r.get_port_at(X3) ≡ X5) then
8                  X3 = X3 + 1;
9          else
10                 X3 = 1000
11         end if
12 end while
13 if (X3 ≢ 1000) then
14         C_w.send(1337);
15 else
16         skip;
17 end if
```

Listing 2: KnockD implementation 1

4.2 Implementation 2

The second implementation tries to save up all the ports storing them in the ram. and when the max number of iterations have been met it checks if the knocks where the right ones.

```
1  X4 = C_r.get_sequence_length();
2  X3 = 0;
3  while(X3 < X4) do
4          X1 = C_w.Receive();
5          X2 = C_r.Store(X1);
6          X3 = X3 + 1
7  end while
8  if (C_r.check_ports(X_3)) then
9          C_w.send(1337)
10 else
11         skip
12 end if
```

Listing 3: KnockD implementation 2

4.3 Prediction

From informally analysing these algorithms we would think the first implementation given by listing 2 would be the least energy consuming. The $check\_ports(X_3)$ is a very expensive operation containing a while. The same operation is emulated in implementation 1 by piggybacking of the same while that receives the messages and extracts a port.

4.4 Result table for implementation 1

n	$t_n(10^{-6}s)$	$\mathfrak{e}_r(10^{-9}J)$ *in* Γ_n	$\mathfrak{e}_w(10^{-9}J)$ *in* Γ_n	$\mathfrak{e}_{cpu}(10^{-9}J)$ *in* Γ_n
1	0	0	0	0
2	0 + 0.005	0.0 + 6.0 + (0 * 0.7375) = 6.0	-	-
3	0.005 + 0.0275	-	-	0 + 1.1
4	0.0325 + 0.0275	-	-	1.1 + 1.1
5	0.06 + 0.0275	-	-	2.2 + 1.1
6	0.0875 + 0.055	-	-	3.3 + 2.2
7	0.1425 + 1.50	-	0 + 0.1180 + 0.1425 * 10^3 * 0.820 = 116.97	-
8	1.6425 + 0.0275	-	-	5.5 + 1.1
9	1.6700 + 0.005	6.0 + 5.75 + 1.67 * 10^3 * .7375 = 1243.375	-	-
10	1.6750 + 0.0275	-	-	6.6 + 1.1
11	1.7025 + 0.0325	1243.375 + 6.0 + 0.0325 * 10^3 * .7375 = 1267.344	-	-
12	1.7350 + 0.0275	-	-	7.7 + 1.1
13	1.7625 + 0.0325	1267.344 + 6.0 + 0.06 * 10^3 * .7375 = 1317.594	-	-
14	1.7950 + 0.0275	-	-	8.8 + 1.1
15	1.8225 + 0.0275	-	-	9.9 + 2.2
16	1.8500 + 0.0275	-	-	12.1 + 1.1
17	1.8775 + 0.0275 = 1.9050	-	-	13.2 + 1.1 = 14.3
18	1.8500 + 0.0275 = 1.8775	-	-	12.1 + 1.1 = 13.2
19	0.06 + $\rho_3(X_3)$ * 1.845 + 0.0275	6.0 + $\rho_3(X_3)$ * 1261.344	0 + $\rho_3(X_3)$ * 116.97	2.2 + $\rho_3(X_3)$ * 12.1 + 1.1
20	0.0875 + $\rho_3(X_3)$ * 1.845 + 0.055	-	-	3.3 + $\rho_3(X_3)$ * 12.1 + 2.2
21	0.0925 + $\rho_3(X_3)$ * 1.845 + 1.55 = 1.6425 + $\rho_3(X_3)$ * 1.845	-	$\rho_3(X_3)$ * 116.97 + 0.4235 + 0.845 * 10^3 * 0.820	-

Table 6: Result table of the algorithm 2

4.5 Result table for implementation 2

n	$t_n(10^{-6}s)$	$\mathfrak{e}_r(10^{-9}J)$ *in* Γ_n	$\mathfrak{e}_w(10^{-9}J)$ *in* Γ_n	$\mathfrak{e}_{cpu}(10^{-9}J)$ *in* Γ_n
1	0	0	0	0
2	0 + 0.005	0.0 + 6.0 + (0 * 0.7375) = 6.0	-	-
3	0.005 + 0.0275	-	-	0 + 1.1
4	0.0325 + 0.0275	-	-	1.1 + 1.1
5	0.06 + 0.0275	-	-	2.2 + 1.1
6	0.0875 + 0.055	-	-	3.3 + 2.2
7	0.1425 + 1.50	-	0 + 0.1180 + 0.1425 * 10^3 * 0.820 = 116.97	-
8	1.6425 + 0.0275	-	-	5.5 + 1.1
9	1.6700 + 0.005	6.0 + 5.75 + 1.67 * 10^3 * .7375 = 1243.375	-	-
10	1.6750 + 0.0275	-	-	6.6 + 1.1
11	1.7025 + 0.0275 = 1.7300	-	-	7.7 + 1.1
12	0.06 + $\rho_3(X_3)$ * 1.6700 + $\rho_3(X_3)$ * 0.875 + 0.055	6.0 + $\rho_3(X_3)$ * 1237.375 + $\rho_3(X_3)$ * 9.3 + 2.2 + 0.06 * 10^{-3} * .7375	-	-
13	0.115 + $\rho_3(X_3)$ * 2.5450 + 0.055	-	-	2.2 + $\rho_3(X_3)$ * 6.6 + 2.2
14	0.170 + $\rho_3(X_3)$ * 2.5450 + 1.55 = 1.725 + $\rho_3(X_3)$ * 2.5450	52.45 + $\rho_3(X_3)$ * 1246.675	0 + 1.0164 + lc116.97 + 0.17 * 10^{-3} * 0.820 = 140.41 + lc116.97	4.4 + $\rho_3(X_3)$ * 6.6

Table 7: Result table of the algorithm 3

5 Power consumption results

From tables 6 to 7 we can calculate the energy costs of the algorithms with certain ranking functions. We kept this as abstract as possible up to now. So lets plot the results with the number of iterations as parameter.

Please note that the lines are just there to accentuate the trends. It is impossible to have a half iteration of a loop. The lines especially help to determine if the lines in Figure 3 are parallel or not, which they are not. They actually slightly diverge.

In Figure 2 one can see the time set out against the number of iterations. The number of iterations directly depends on the *get_seq_len()* function. As this is dynamic we need to plot to see if we can find an algorithm that uses the least amount of time/power with the amount of iterations you specify in your application. The formula to draw is giving by the last cell in the first column of the result tables found either in the appendix or in the analysis section.

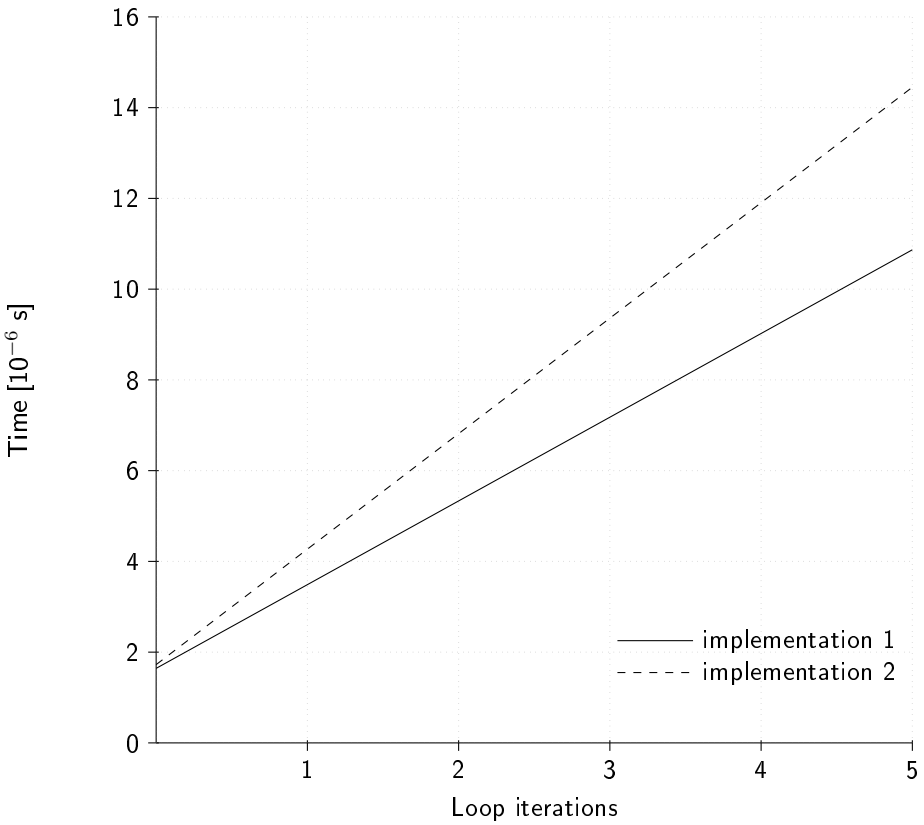

Figure 2: The amount of iterations versus the duration of the 2 implementations

In figure 3 we plot these same iterations against the power consumption. The plotted equation is the sum of the bottom cells of the last 3 columns of the Tables found in Section 4.

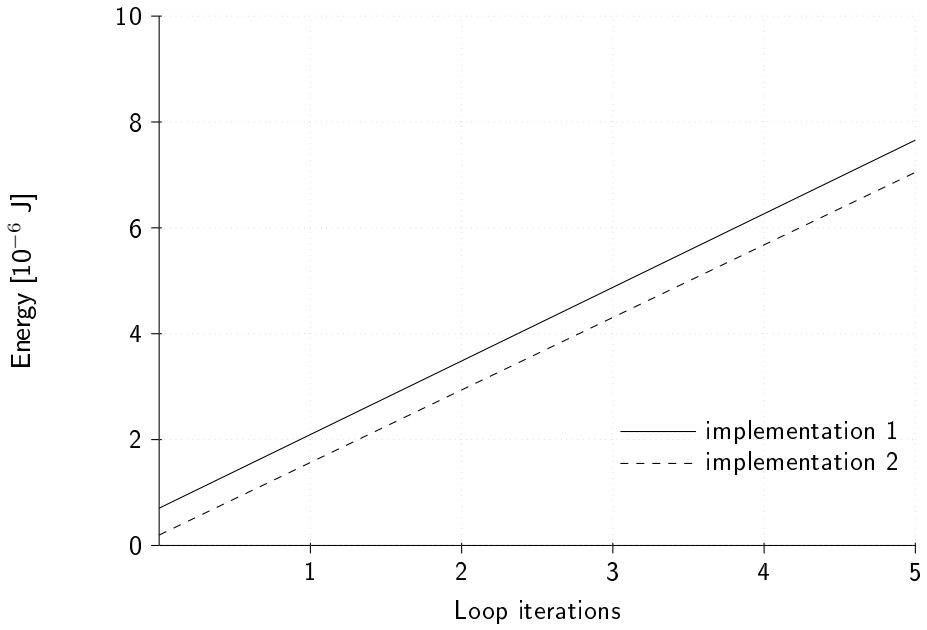

Figure 3: The amount of iterations versus the energy consumption of the 2 implementations

The plots in figure 2 to 3 tell us that algorithm implementation 2 is less power consuming. But the second implementation takes more time to run.

This is contradictory with what we informally guessed. Although the difference isn't that big, the fact that the second algorithm can be proven to use less power is a indication that we need to do these analysis systematically and not by intuition. Especially when these kind on analysis can be done automatically the result will even be more reliable as human error can be ruled out. If the implementation is correct.

6 Conclusion

The method we are using to determine the power consumption analysis statically is very comprehensible. The problem lies not in applying the technique or creating a suitable model, but in the time it takes to do so. We spend a large part of our research time getting realistic estimates for our particular system. Unfortunately this part cannot be automated. Because of this you need hardware experts that understand the hardware and can either estimate, measure or calculate the energy consumption. However if this method becomes a world standard, hardware companies will be easier persuaded to give a component description with each piece of hardware they release.

The analysis part can be automated and research is being done to make a tool that does the semantic analysis and calculates the energy consumption with a corresponding model. The automation of the analysis itself will greatly reduce the time it takes to get results. It would take far less time to learn to use this tool than to learn the analysis and executing it by hand.

Another problem with the current method is that it uses a toy language that is very similar to While. It is not very expressive and analysing complex code is almost impossible. Extending this technique to a language like C however is equally impossible without further research. It requires taking a expressive subset of a contemporary language that has very strong semantics.

Maybe future research should take a look at formalization of C currently being researched by Freek Wiedijk [7].

So the focus of future research should be focused on usability rather than making more proofs of concept. This technique will only become widely used if it is easy applicable, and if it analyses the real code and not some translation into another domain specific language. A solution would be to have an compiler translating it to a language on which the semantics are defined. However, this could create an other problem. You never know how a compiler will translate the code, it is error prone and time consuming work and will be too expensive for larger projects to maintain.

There is still a long road ahead for this new technique to become accepted as a world standard. However the technique is very promising and brings the best solution for a very important problem.

References

[1] Atheros. AR9331 Highly-Integrated and Cost Effective IEEE 802.11n 1x1 2.4 GHz SoC for AP and Router Platforms, December 2010.

[2] Iyad Al Khatib Rassul Ayani Gerald and Q Maguire Jr. Wireless LAN Access Points Uplink and Downlink Delays: Packet Service-Time Comparison. In *Proceedings of The 16th Nordic Teletraffic Seminar (NTS-16), August*, pages 253–264, 2002.

[3] U Liqat, S Kerrison, A Serrano, K Georgiou, P Lopez-Garcia, N Grech, MV Hermenegildo, and K Eder. Energy Consumption Analysis of Programs based on XMOS ISA-Level Models.

[4] MIPS Technologies. MIPS32 24Kc Processor Core Datasheet, 2008.

[5] Paolo Parisen Toldin, Rody Kersten, Bernard van Gastel, and Marko van Eekelen. Soundness Proof for a Hoare Logic for Energy Consumption Analysis. Technical Report ICIS–R13009, Radboud University Nijmegen, October 2013.

[6] Paolo Parisin Toldin, Rody Kersten, Bernard van Gastel, and Marko van Eekelen. A Generic Type System for Energy Consumption Analysis, 2013.

[7] Freek Wiedijk. Formalizing the C99 standard in HOL, Isabelle and Coq, 2010.

[8] Zentel. A3S56D30FTP, A3S56D40FTP, 256M Double Data Rate Synchronous DRAM, 2010.

A Proof trees

A.1 Example

Start

$$\dfrac{\dfrac{}{\{\Gamma_1;t_1;\rho_1\}\ \ 0\ \ \{\Gamma_1;t_1;\rho_1\}}\,N \qquad \Gamma_2=\Gamma_1[C_{cpu}::\mathfrak{e}+=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_1;t_1;\rho_1\}\ \ X_1=0\{\Gamma_2;t_2=t_1+C_{cpu}::\mathfrak{T}_a;\rho_2\}}\,A \qquad \{\Gamma_2;t_2;\rho_2\}\ \ \mathit{Rest}_1\ \ \{\Gamma_7;t_7;\rho_4\}}{\{\Gamma_1;t_1;\rho_1\}\ \ X_1=0;\mathit{Rest}_1\ \ \{\Gamma_7;t_7;\rho_4\}}\,C$$

Rest_1

$$\dfrac{\dfrac{\dfrac{}{\{\Gamma_2;t_2;\rho_2\}\ \ 20\ \ \{\Gamma_2;t_2;\rho_2\}}\,N \qquad \Gamma_3=\Gamma_2[C_{cpu}::\mathfrak{e}+=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_2;t_2;\rho_2\}\ \ X_2=20\{\Gamma_3;t_3=t_2+C_{cpu}::\mathfrak{T}_a;\rho_3\}}\,A \qquad \{\Gamma_3;t_3;\rho_3\}\ \ \mathit{While}\ \ \{\Gamma_7;t_7;\rho_4\}}{\{\Gamma_2;t_2;\rho_2\}\ \ X_2=20;\mathit{While}\ \ \{\Gamma_7;t_7;\rho_4\}}\,C$$

While_1

$$\dfrac{\{\Gamma_3;t_3;\rho_3\}\ \ \mathit{Cond}\ \ \{\Gamma_4;t_4;\rho_3\} \qquad \Gamma_5=\Gamma_4[C_{cpu}::\mathfrak{e}+=C_{cpu}::\mathfrak{E}_w] \qquad \{\Gamma_5;t_4;\rho_3\}\ \ \mathit{Body}\ \{\Gamma_7;t_7;\rho_4\}}{\{\Gamma_3;t_3;\rho_3\}\ \ \mathit{while}\ (X_1<X_2)\ \mathit{do}\ X_1=C_s.m()\ \mathit{end\ while}\ \ \{\Gamma_7;t_7;\rho_4\}}$$

Cond

$$\dfrac{\dfrac{}{\{\Gamma_3;t_3;\rho_3\}\ \ X_1\ \ \{\Gamma_3;t_3;\rho_3\}}\,V \qquad \dfrac{}{\{\Gamma_3;t_3;\rho_3\}\ \ X_2\ \ \{\Gamma_3;t_3;\rho_3\}}\,V \qquad \Gamma_4=\Gamma_3[C_{cpu}::\mathfrak{e}+=C_{cpu}::\mathfrak{E}_e]}{\{\Gamma_3;t_3;\rho_3\}\ \ X_1<X_2\{\Gamma_4;t_4=t_3+C_{cpu}::\mathfrak{T}_e;\rho_3\}}$$

Body

$$\dfrac{\dfrac{\Gamma_6=\Gamma_5[C_s::s\leftarrow\delta_f(C_i::s),C_s::\tau\leftarrow t_5,C_s::\mathfrak{e}+=C_s::\mathfrak{E}_m+C_s::td(t_5)]}{\{\Gamma_5;t_5;\rho_3\}\ \ C_s.m()\{\Gamma_6;t_6=t_5+C_s::\mathfrak{T}_m;\rho 3\}}\,CF \qquad \Gamma_7=\Gamma_6[C_{cpu}::\mathfrak{e}+=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_5;t_5=t_4+C_{cpu}::\mathfrak{T}_w;\rho_3\}X_1=C_s.m()\{\Gamma_7;t_7=t_6+C_{cpu}::\mathfrak{T}_a;\rho_4\}}\,A$$

A.2 Implementation 1

Start

$$\dfrac{\dfrac{\dfrac{\Gamma_2 = \Gamma_1[C_r :: s \leftarrow \delta_f(C_i :: s), Cr :: \tau \leftarrow t_1, C_r :: \mathfrak{e}+ = C_r :: \mathfrak{E}_{read} + Cr :: td(t_1)]}{\{\Gamma_1; t_1; \rho_1\}\ \ C_r.get\_seq\_len()\ \{\Gamma_2; t_2 = t_1 + C_r :: \mathfrak{T}_{read}; \rho_1\}}\ CF \qquad \Gamma_3 = \Gamma_2[C_{cpu} :: \mathfrak{e}+ = \mathfrak{E}_a]}{\{\Gamma_1; t_1; \rho_1\}\ \ X_4 = C_r.get\_seq\_len()\ \{\Gamma_3; t_3 = t_2 + C_{cpu} :: \mathfrak{T}_a; \rho_2\}}\ A \qquad \{\Gamma_3; t_3; \rho_2\}\ \ Rest_1\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}{\{\Gamma_1; t_1; \rho_1\}\ \ X_4 = C_r.get\_seq\_len(); Rest_1\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}\ C$$

Rest_1

$$\dfrac{\dfrac{}{\{\Gamma_3; t_3; \rho_2\}\ \ 0\ \ \{\Gamma_3; t_3; \rho_2\}}\ N \qquad \Gamma_4 = \Gamma_3[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_3; t_3; \rho_2\}\ \ X_3 = 0\ \{\Gamma_4; t_4 = t_3 + C_{cpu} :: \mathfrak{T}_a; \rho_3\}}\ A \qquad \{\Gamma_4; t_4; \rho_3\}\ While\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}{\{\Gamma_3; t_3; \rho_2\}\ \ X_3 = 0; While\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}\ C$$

While

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\Gamma_4; t_4; \rho_3\}\ \ X_3\ \ \{\Gamma_4; t_4; \rho_3\}}\ V \qquad \dfrac{}{\{\Gamma_4; t_4; \rho_3\}\ \ X_4\ \ \{\Gamma_4; t_4; \rho_3\}}\ V \qquad \Gamma_5 = \Gamma_4[C_{cpu} :: \mathfrak{e}+ = \mathfrak{E}_e]}{\{\Gamma_4; t_4; \rho_3\}\ \ X_3 < X_4\ \{\Gamma_5; t_5 = t_4 + C_{cpu} :: \mathfrak{T}_e; \rho_3\}}\ E \qquad \Gamma_6 = \Gamma_5[C_{cpu} :: \mathfrak{e}+ = \mathfrak{E}_w]] \qquad WBody_1}{\{\Gamma_4; t_4; \rho_3\}\ \ while(X3\ <\ X4)\ do\ WBody_1\ end\ while\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ W \qquad \{\Gamma_{17}; t_{17}; \rho_7\}\ \ AWhile_1\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}{\{\Gamma_4; t_4; \rho_3\}\ \ while(X3\ <\ X4)\ do\ WBody_1\ end\ while;\ AWhile_1\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}\ C$$

WBody_1

$$\dfrac{\dfrac{\dfrac{\Gamma_7 = \Gamma_6[C_w :: s \leftarrow \delta_f(C_w :: s), C_w :: \tau \leftarrow t_6, C_r :: \mathfrak{e}+ = C_w :: \mathfrak{E}_{recv} + C_w :: td(t_6)]}{\{\Gamma_6; t_6; \rho_3\}\ \ C_w.Receive()\ \{\Gamma_7; t_7 = t_6 + C_w :: \mathfrak{T}_{recv}; \rho_3\}}\ CF \qquad \Gamma_8 = \Gamma_7[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_6; t_6; \rho_3\}\ \ X_1 = C_w.Receive()\ \{\Gamma_8; t_8 = t_7 + C_{cpu} :: \mathfrak{T}_a; \rho_4\}}\ A \qquad \{\Gamma_8; t_8; \rho_4\}\ \ WBody_2\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}{\{\Gamma_6; t_6 = t_5 + C_{cpu} :: \mathfrak{T}_w; \rho_3\}\ X_1 = C_w.Receive(); WBody_2\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ C$$

WBody_2

$$\dfrac{\dfrac{\dfrac{\Gamma_9 = \Gamma_8[C_r :: s \leftarrow \delta_f(C_r :: s), C_r :: \tau \leftarrow t_8, C_r :: \mathfrak{e}+ = C_r :: \mathfrak{E}_{Store} + C_r :: td(t_8)]}{\{\Gamma_8; t_8; \rho_4\}\ \ C_r.store(X_1)\ \{\Gamma_9; t_9 = t_8 + C_r :: \mathfrak{T}_{store}; \rho_4\}}\ CF \qquad \Gamma_{10} = \Gamma_9[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_8; t_8; \rho_4\}\ \ X_2 = C_r.store(X_1)\ \{\Gamma_{10}; t_{10} = t_9 + C_{cpu} :: \mathfrak{T}_a; \rho_5\}}\ A \qquad \{\Gamma_{10}; t_{10}; \rho_5\}\ \ WBody_3\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}{\{\Gamma_8; t_8; \rho_4\}\ \ X_2 = C_r.store(X_1); WBody_3\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ C$$

WBody_3

$$\dfrac{\dfrac{\dfrac{\Gamma_{11} = \Gamma_{10}[C_r :: s \leftarrow \delta_f(C_r :: s), C_r :: \tau \leftarrow t_{10}, C_r :: \mathfrak{e}+ = C_r :: \mathfrak{E}_{get\_port} + C_r :: td(t_{10})]}{\{\Gamma_{10}; t_{10}; \rho_5\}\ \ C_r.get\_port(X_2)\ \{\Gamma_{11}; t_{11} = t_{10} + C_r :: \mathfrak{T}_{get\_port}; \rho_5\}}\ CF \qquad \Gamma_{12} = \Gamma_{11}[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_{10}; t_{10}; \rho_5\}\ \ X_5 = C_r.get\_port(X_2)\ \{\Gamma_{12}; t_{12} = t_{11} + C_{cpu} :: \mathfrak{T}_a; \rho_6\}}\ A \qquad \{\Gamma_{12}; t_{12}; \rho_6\}\ \ if_1\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}{\{\Gamma_{10}; t_{10}; \rho_5\}\ \ X_5 = C_r.get\_port(X_2); if_1\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ C$$

if_1

$$\dfrac{\dfrac{\dfrac{\Gamma_{13} = \Gamma_{12}[C_r :: s \leftarrow \delta_f(C_r :: s), C_r :: \tau \leftarrow t_{12}, C_r :: \mathfrak{e}+ = C_r :: \mathfrak{E}_{get\_port} + C_r :: td(t_{12})]}{\{\Gamma_{12}; t_{12}; \rho_6\}\ \ C_r.get\_port(X_3)\ \{\Gamma_{13}; t_{13} = t_{12} + C_r :: \mathfrak{T}_{get\_port}; \rho_6\}}\ CF \quad \dfrac{}{\{\Gamma_{13}; t_{13}; \rho_6\}\ \ X_5\ \ \{\Gamma_{13}; t_{13}; \rho_6\}}\ V \quad \Gamma_{14} = \Gamma_{13}[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_e]}{\{\Gamma_{12}; t_{12}; \rho_6\}\ \ C_r.get\_port(X_3) \equiv X_5\ \{\Gamma_{14}; t_{14} = t_{13} + C_{cpu} :: \mathfrak{T}_e; \rho_6\}}\ E \quad if_1^t \\ \Gamma_{15} = \Gamma_{14}[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_{it}] \quad if_1^f}{\{\Gamma_{12}; t_{12}; \rho_6\}\ \ if\ (C_r.get\_port(X_3) \equiv X_5)\ then\ if_1^t\ else\ if_1^f\ end\ if\ \ \{\Gamma_{17}; t_{17}; \rho_6\}}\ I$$

if_1^t

$$\dfrac{\dfrac{\dfrac{}{\{\Gamma_{15}; t_{15}; \rho_6\}\ \ X_3\ \ \{\Gamma_{15}; t_{15}; \rho_6\}}\ V \quad \dfrac{}{\{\Gamma_{15}; t_{15}; \rho_6\}\ \ 1\ \ \{\Gamma_{15}; t_{15}; \rho_6\}}\ N \quad \Gamma_{16} = \Gamma_{15}[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_e]}{\{\Gamma_{15}; t_{15}; \rho_6\}\ \ X_3 + 1\ \{\Gamma_{16}; t_{16} = t_{15} + C_{cpu} :: \mathfrak{T}_e; \rho_6\}}\ E \quad \Gamma_{17} = \Gamma16[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_a]}{\{\{\Gamma_{15}; t_{15} = t_{14} + C_{cpu} :: \mathfrak{T}_e; \rho_6\} X_3 = X_3 + 1\ \{\Gamma_{17}; t_{17} = t_{16} + C_{cpu} :: \mathfrak{T}_a; \rho_7\}}\ A$$

if_1^f

$$\dfrac{\dfrac{}{\{\Gamma_{15}; t_{15}; \rho_6\}\ \ 1000\ \ \{\Gamma_{15}; t_{15}; \rho_6\}}\ N \quad \Gamma_{18} = \Gamma_{15}[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_a]}{\{\{\Gamma_{15}; t_{15} = t_{14} + C_{cpu} :: \mathfrak{T}_e; \rho_6\} X_3 = 1000\ \{\Gamma_{18}; t_{18} = t_{15} + C_{cpu} :: \mathfrak{T}_a; \rho_7\}}\ A$$

AWhile

$$\dfrac{\dfrac{\dfrac{}{\{\Gamma_{17}; t_{17}; \rho_7\}\ \ X_3\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ V \quad \dfrac{}{\{\Gamma_{17}; t_{17}; \rho_7\}\ \ 1000\ \ \{\Gamma_{17}; t_{17}; \rho_7\}}\ N \quad \Gamma_{19} = \Gamma_{17}[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_e]}{\{\Gamma_{17}; t_{17}; \rho_7\}\ \ X_3 \not\equiv 1000\ \{\Gamma_{19}; t_{19} = t_{17} + C_{cpu} :: \mathfrak{T}_e; \rho_7\}}\ E \quad if_2^t \\ \Gamma_{20} = \Gamma_{19}[C_{cpu} :: \mathfrak{e}+ = C_{cpu} :: \mathfrak{E}_{it}] \quad if_2^f}{\{\Gamma_{17}; t_{17}; \rho_7\}\ if(X_3 \not\equiv 1000)\ then\ if_2^t\ else\ if_2^f end\ if\ \ \{\Gamma_{21}; t_{21}; \rho_7\}}\ I$$

if_2^t

$$\dfrac{\Gamma_{21} = \Gamma_{20}[C_w :: s \leftarrow \delta_f(C_w :: s), C_w :: \tau \leftarrow t_{20}, C_w :: \mathfrak{e}+ = C_w :: \mathfrak{E}_{send} + C_w :: td(t_{20})]}{\{\Gamma_{20}; t_{20} = t_{19} + C_{cpu} :: \mathfrak{T}_{it}; \rho_7\}\ C_w.send()\ \{\Gamma_{21}; t_{21} = t_{20} + C_w :: \mathfrak{T}_{send}; \rho_7\}}\ CF$$

if_2^f

$$\{\Gamma_{20}; t_{20} = t_{19} + C_{cpu} :: \mathfrak{T}_{it}; \rho_7\}\ skip\ \ \{\Gamma_{20}; t_{20}; \rho_7\}$$

A.3 Implementation 2

Start

$$\dfrac{\dfrac{\Gamma_2 = \Gamma_1[C_r :: s \leftarrow \delta_f(C_i :: s), Cr :: \tau \leftarrow t_1, C_r :: \mathfrak{e}+= C_r :: \mathfrak{E}_{read} + Cr :: td(t_1)]}{\{\Gamma_1; t_1; \rho_1\}\ \ C_r.get\_seq\_len()\ \{\Gamma_2; t_2 = t_1 + C_r :: \mathfrak{T}_{read}; \rho_1\}}\ CF \qquad \Gamma_3 = \Gamma_2[C_{cpu} :: \mathfrak{e}+= \mathfrak{E}_a]}{\{\Gamma_1; t_1; \rho_1\}\ \ X_4 = C_r.get\_seq\_len()\ \{\Gamma_3; t_3 = t_2 + C_{cpu} :: \mathfrak{T}_a; \rho_2\}}\ A \qquad \{\Gamma_3; t_3; \rho_2\}\ \ Rest_1\ \{\Gamma_{19}; t_{19}; \rho_5\}}{\{\Gamma_1; t_1; \rho_1\}\ \ X_4 = C_r.get\_seq\_len(); Rest_1\ \{\Gamma_{19}; t_{19}; \rho_5\}}\ C$$

Rest_1

$$\dfrac{\dfrac{}{\{\Gamma_3; t_3; \rho_2\}\ \ 0\ \ \{\Gamma_3; t_3; \rho_2\}}\ N \qquad \Gamma_4 = \Gamma_3[C_{cpu} :: \mathfrak{e}+= C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_3; t_3; \rho_2\}\ \ X_3 = 0\ \{\Gamma_4; t_4 = t_3 + C_{cpu} :: \mathfrak{T}_a; \rho_3\}}\ A \qquad \{\Gamma_4; t_4; \rho_3\}\ While\ \{\Gamma_{19}; t_{19}; \rho_5\}}{\{\Gamma_3; t_3; \rho_3\}\ \ X_3 = 0; While\ \{\Gamma_{19}; t_{19}; \rho_5\}}\ C$$

While

$$\dfrac{\dfrac{\dfrac{}{\{\Gamma_4; t_4; \rho_3\}\ \ X_3\ \ \{\Gamma_4; t_4; \rho_3\}}\ V \qquad \dfrac{}{\{\Gamma_4; t_4; \rho_3\}\ \ X_4\ \ \{\Gamma_4; t_4; \rho_3\}}\ V \qquad \Gamma_5 = \Gamma_4[C_{cpu} :: \mathfrak{e}+= \mathfrak{E}_e]}{\{\Gamma_4; t_4; \rho_3\}\ \ X_3 < X_4\ \{\Gamma_5; t_5 = t_4 + C_{cpu} :: \mathfrak{T}_e; \rho_3\}}\ E \qquad \Gamma_6 = \Gamma_5[C_{cpu} :: \mathfrak{e}+= \mathfrak{E}_w]] \qquad WBody_1}{\{\Gamma_4; t_4; \rho_3\}\ \ while(X3\ <\ X4)\ do\ WBody_1\ end\ while\ \{\Gamma_{11}; t_{11}; \rho_5\}}\ W \qquad \{\Gamma_{11}; t_{11}; \rho_5\}\ AWhile_{if}\ \{\Gamma_{19}; t_{19}; \rho_5\}}{\{\Gamma_4; t_4; \rho_3\}\ \ while(X3\ <\ X4)\ do\ WBody_1\ end\ while;\ AWhile_1\ \{\Gamma_{19}; t_{19}; \rho_5\}}\ C$$

WBody_1

$$\dfrac{\dfrac{\Gamma_7 = \Gamma_6[C_w :: s \leftarrow \delta_f(C_w :: s), C_w :: \tau \leftarrow t_6, C_r :: \mathfrak{e}+= C_w :: \mathfrak{E}_{recv} + C_w :: td(t_6)]}{\{\Gamma_6; t_6; \rho_3\}\ \ C_w.Receive()\ \{\Gamma_7; t_7 = t_6 + C_w :: \mathfrak{T}_{recv}; \rho_3\}}\ CF \qquad \Gamma_8 = \Gamma_7[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_6; t_6; \rho_3\}\ \ X_1 = C_w.Receive()\ \{\Gamma_8; t_8 = t_7 + C_{cpu} :: \mathfrak{T}_a; \rho_4\}}\ A \qquad \{\Gamma_8; t_8; \rho_4\}\ \ WBody_2\ \{\Gamma_{11}; t_{11}; \rho_5\}}{\{\Gamma_6; t_6 = t_5 + C_{cpu} :: \mathfrak{T}_w; \rho_3\}\ X_1 = C_w.Receive(); WBody_2\ \{\Gamma_{11}; t_{11}; \rho_5\}}\ C$$

WBody_2

$$\dfrac{\dfrac{\Gamma_9 = \Gamma_8[C_w :: s \leftarrow \delta_f(C_w :: s), C_r :: \tau \leftarrow t_8, C_w :: \mathfrak{e}+= C_w :: \mathfrak{E}_{Store} + C_r :: td(t_8)]}{\{\Gamma_8; t_8; \rho_4\}\ \ C_r.store(X_1)\ \{\Gamma_8; t_9 = t_8 + C_r :: \mathfrak{T}_{store}; \rho_4\}}\ CF \qquad \Gamma_{10} = \Gamma_9[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_8; t_8; \rho_4\}\ \ X_2 = C_r.store(X_1)\ \{\Gamma_{10}; t_{10} = t_9 + C_{cpu} :: \mathfrak{T}_a; \rho_5\}}\ A \qquad \{\Gamma_{10}; t_{10}; \rho_5\}\ \ WBody_3\ \{\Gamma_{11}; t_{11}; \rho_5\}}{\{\Gamma_8; t_8; \rho_4\}\ \ X_2 = C_r.store(X_1); WBody_3\ \{\Gamma_{11}; t_{11}; \rho_5\}}\ C$$

WBody_3

$$\dfrac{\{\Gamma_{10}; t_{10}; \rho_5\}\ \ X3 + 1\ \ \{\Gamma_{10}; t_{10}; \rho_5\} \qquad \Gamma_{11} = \Gamma_{10}[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma_{10}; t_{10}; \rho_5\}\ \ X3 = X3 + 1\ \{\Gamma_{11}; t_{11} = t_{10} + C_{cpu} :: \mathfrak{T}_a; \rho_5\}}\ A$$

AWhile$_{if}$

$$\frac{\begin{array}{cc} if_{cond} & if^{t} \\ \Gamma_{13} = \Gamma_{12}[C_{cpu} :: \mathfrak{e} + C_{cpu} :: \mathfrak{E}_{it}] & if^{f} \end{array}}{\{\Gamma_{11}; t_{11}; \rho_5\} \;\; if(C_r.check\_ports(X_3)) \; then \; C_w.send(1337) \; else \; skip \; end \; if \;\; \{\Gamma_{19}; t_{19}; \rho_5\}} \; I$$

if$_{cond}$

$$\frac{\Gamma_{12} = \Gamma_{11}[C_r :: s \leftarrow \delta_f(C_i :: s), Cr :: \tau \leftarrow t_{11}, C_r :: \mathfrak{e}+ = C_r :: \mathfrak{E}_{check\_ports} + Cr :: td(t_{11})]}{\{\Gamma_{11}; t_{11}; \rho_5\} \;\; C_r.check\_ports(X_3) \; \{\Gamma_{12}; t_{12} = t_{11} + C_r :: \mathfrak{T}_{check\_ports}; \rho_5\}} \; CF$$

if^{t}

$$\frac{\Gamma_{14} = \Gamma_{13}[C_w :: s \leftarrow \delta_f(C_w :: s), C_w :: \tau \leftarrow t_{13}, C_w :: \mathfrak{e}+ = C_w :: \mathfrak{E}_{send} + C_w :: td(t_{13})]}{\{\Gamma_{13}; t_{13} = t_{12} + C_{cpu} :: \mathfrak{T}_{it}; \rho_5\} C_w.send(1337) \{\Gamma_{14}; t_{14} = t_{13} + C_w :: \mathfrak{T}_{send}; \rho_5\}}$$

if^{f}

$$\{\Gamma_{13}; t_{13} = t_{12} + C_{cpu} :: \mathfrak{T}_{it}; \rho_5\} \; skip \;\; \{\Gamma_{13}; t_{13}; \rho_5\}$$