The Cephalopode Project: Creating a Low-Power IoT Device Aimed at Functional Language Execution.

Carl-Johan Seger, Jeremy Pope, Henrik Valter
Department of CSE, Chalmers University of Technology, Gothenburg

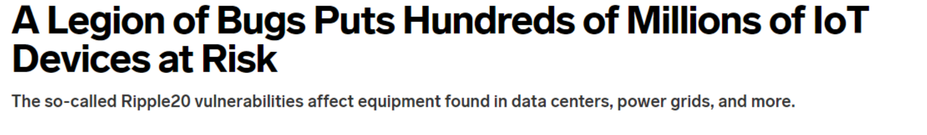
Jules Saget, Dorian Lesbre, Nicolas Nardino, ENS, Paris

March 3, 2024

INTERNET-OF-THINGS: THE GOOD

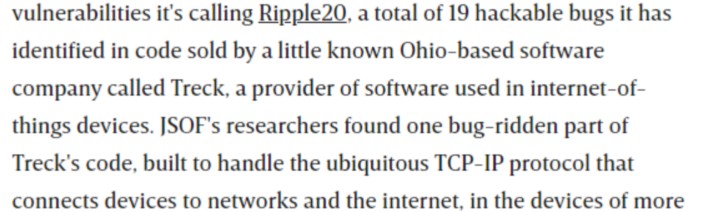
- More and more of all devices, tools, machines, vehicles, homes, industries, ..., are becoming connected to the internet
- By middle of the 2020s, 50 billion devices are expected to be connected.
- New functionality and convenience is added at a breakneck speed.
  - A “smart” lightbulb is no longer surprising….
- A modern car has hundreds of smart “things”. Homes are heading in the same direction. Industry is already there.

INTERNET-OF-THINGS: THE BAD

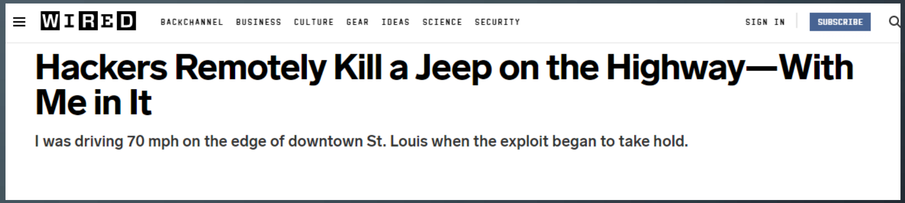


A Legion of Bugs Puts Hundreds of Millions of IoT Devices at Risk

The so-called Ripple20 vulnerabilities affect equipment found in data centers, power grids, and more.

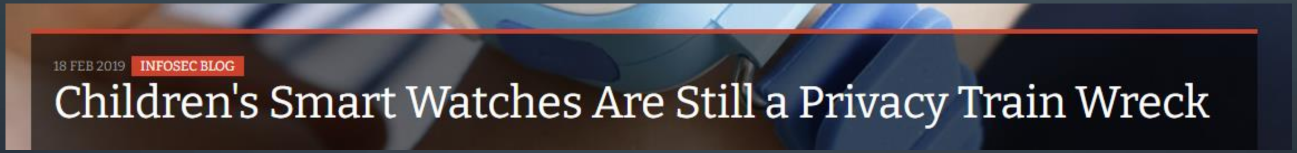


vulnerabilities it's calling Ripple20, a total of 19 hackable bugs it has identified in code sold by a little known Ohio-based software company called Treck, a provider of software used in internet-of-things devices. JSOF's researchers found one bug-ridden part of Treck's code, built to handle the ubiquitous TCP-IP protocol that connects devices to networks and the internet, in the devices of more



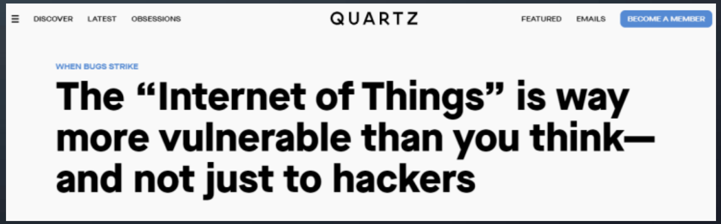
Hackers Remotely Kill a Jeep on the Highway—With Me in It

I was driving 70 mph on the edge of downtown St. Louis when the exploit began to take hold.



18 FEB 2019 INFOSEC BLOG

Children's Smart Watches Are Still a Privacy Train Wreck

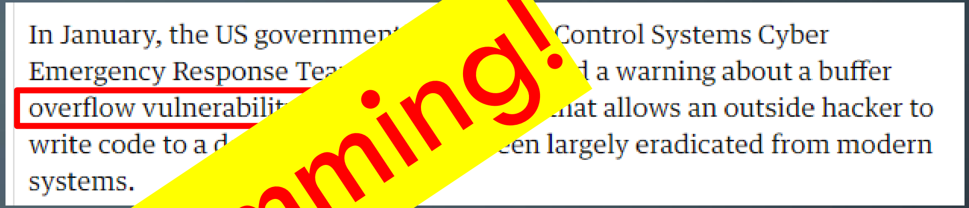


WHEN BUGS STRIKE

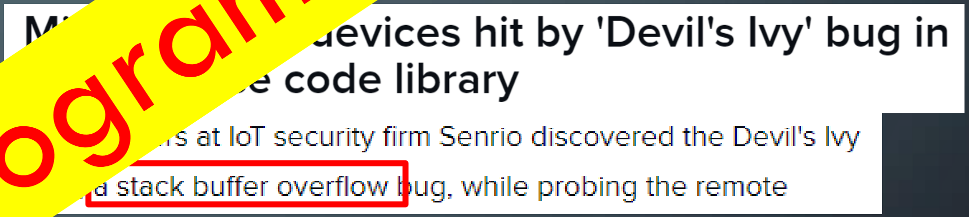
The “Internet of Things” is way more vulnerable than you think—and not just to hackers

INTERNET-OF-THINGS: THE UGLY

- Reasons for IoT Failures
  - Operating environment
    - Unexpected inputs
    - Deliberate bogus inputs
  - Integration problems
    - Unexpected interactions
  - Device configuration
    - Obscure user interface
    - Hard-coded back doors(!)
  - Connectivity
    - Robustness to lo
  - Device load
  - Inadequate testing
  - …

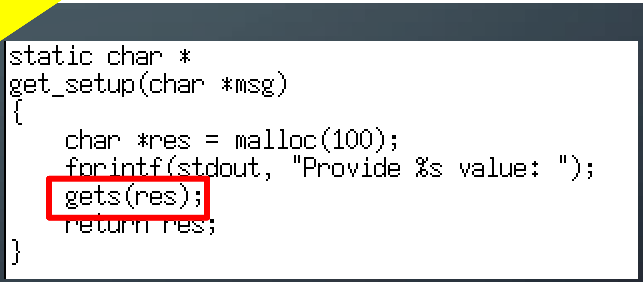


In January, the US governmen Control Systems Cyber Emergency Response Tea d a warning about a buffer overflow vulnerabilit hat allows an outside hacker to write code to a d een largely eradicated from modern systems.



M devices hit by 'Devil's Ivy' bug in e code library

s at IoT security firm Senrio discovered the Devil's Ivy a stack buffer overflow bug, while probing the remote



```
static char *
get_setup(char *msg)
{
    char *res = malloc(100);
    fprintf(stdout, "Provide %s value: ");
    gets(res);
    return res;
}
```

Low-level programming!

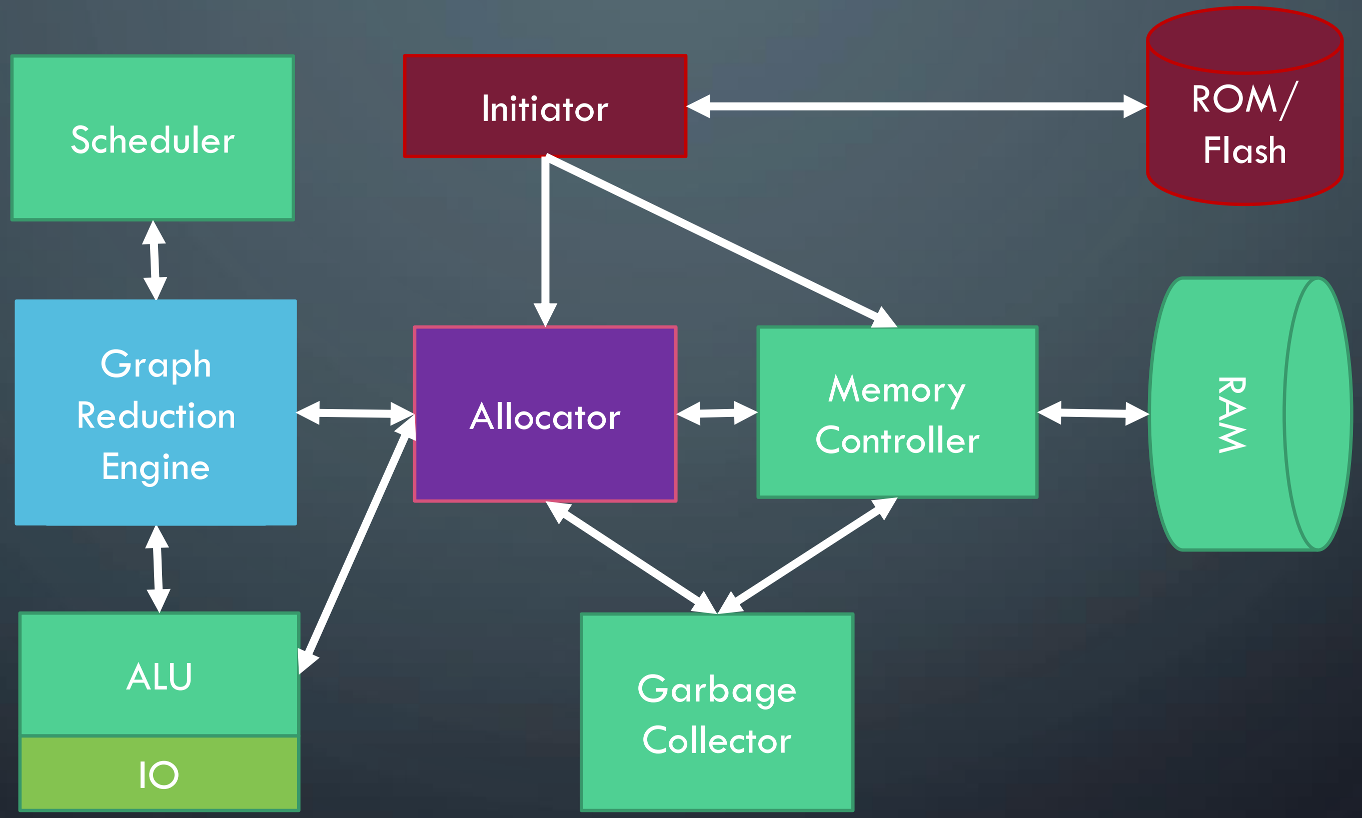
OCTOPI AND CEPHALOPODE

- The goal of the Octopi project, is to develop technology for securely programming IoT systems by high-level languages.
  - In particular, using "pure" functional languages
- Executing functional languages on IoT devices is challenging, since compute power is very limited due to energy limitations.
  - Traditional HW is very inefficient when executing functional programs.
- Idea: Design a very low-power functional language execution engine for IoT devices.
  - In order of priority (high to low): energy, security, performance.

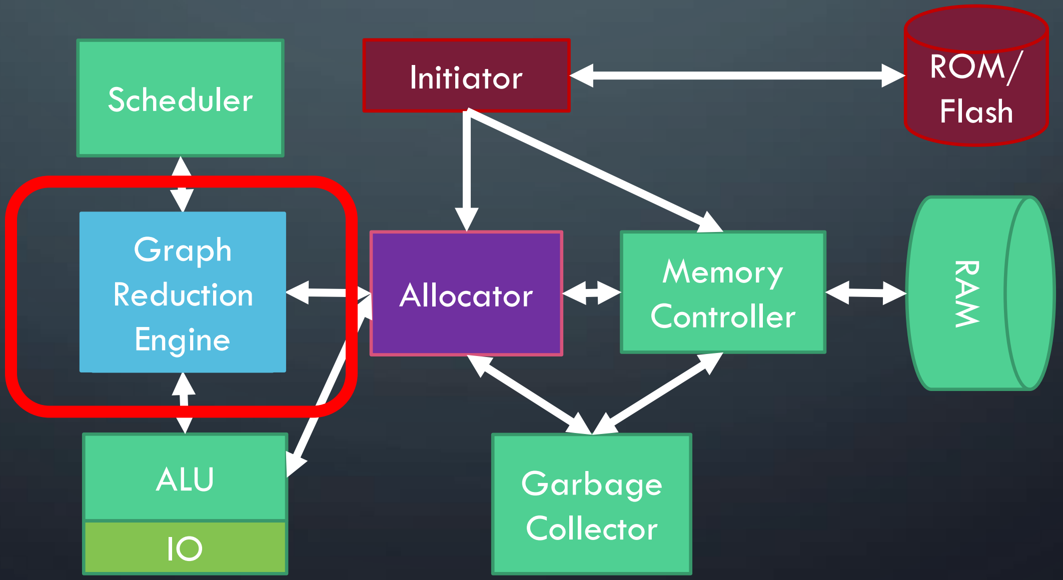
CEPHALOPODE: BASIC IDEA

- Design a (very simple) graph reduction engine
  - Extremely low power execution engine
- Incorporate security/robustness features in HW
  - Arbitrary precision arithmetic to remove overflow issues
  - Provide HW controlled process isolation
  - Remove/Reduce any possible side channels
  - Simple design to allow complete formal verification of HW
- Start with a simple version to flesh out design flow and ease exploration of different alternatives.

CEPHALOPODE ARCHITECTURE



GRAPH REDUCTION ENGINE



GRAPH REDUCTION

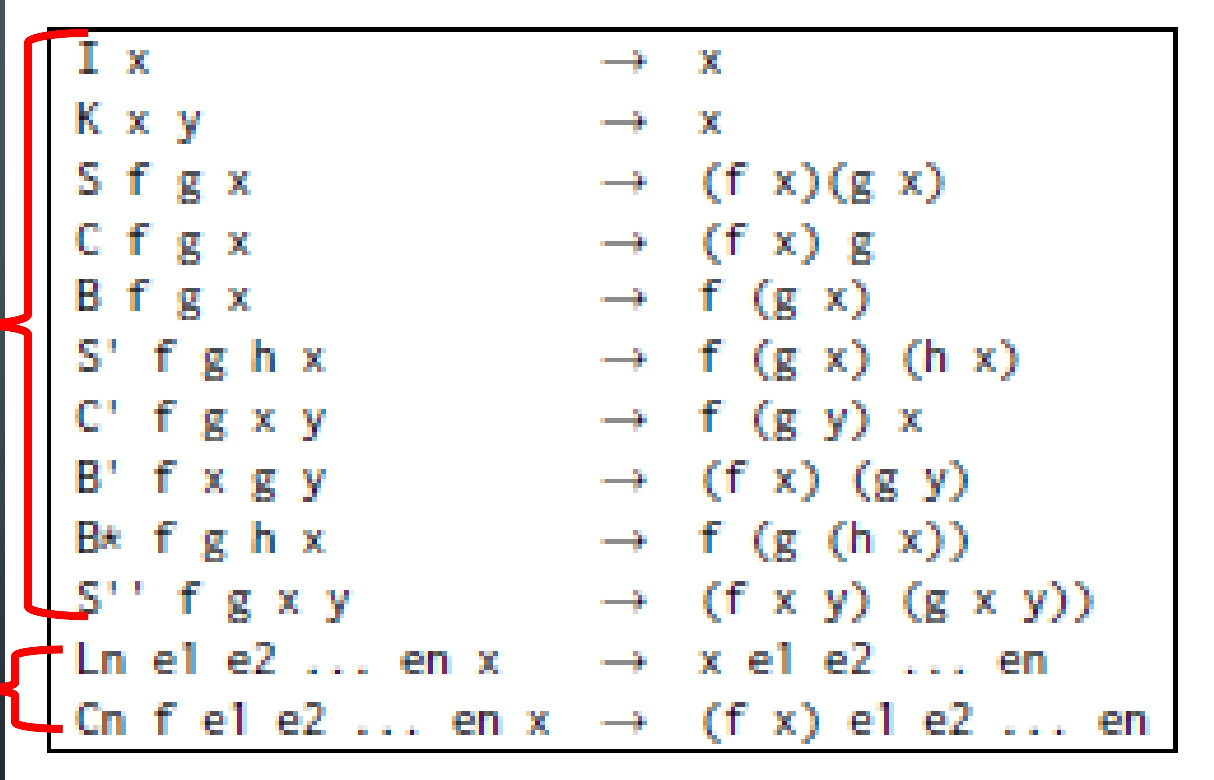
- Functional language code is (basically) lambda expressions.
    - Functional program:

        -
          ```
          let dbl x = x+x in
          let quad x = dbl x + dbl x in
          quad (12 – dbl 4)
          ```
    - Lambda expression:
        - ((λdbl. ((λquad. (quad ((- 12) (dbl 4)))) (λx. ((+ (dbl x)) (dbl x))))) (λx. ((+ x) x)))
- Evaluating lambda expressions directly is difficult.
- Compile to combinators!
    - SKI combinator expression:
        - (((S ((S (K (S I))) ((S (K K)) ((S (K (- 12))) ((S I) (K 4)))))) ((S ((S (K S)) ((S (K (S (K +)))) ((S ((S (K S)) ((S (K K)) I))) (K I))))) ((S ((S (K S)) ((S (K K)) I))) (K I)))) ((S ((S (K +)) I)) I))

COMBINATORS USED IN CEPHALOPODE

Traditional Set of Fixed Combinators

Infinite Family of Paramterized Combinators



I x	→	x
K x y	→	x
S f g x	→	(f x)(g x)
C f g x	→	(f x) g
B f g x	→	f (g x)
S' f g h x	→	f (g x) (h x)
C' f g x y	→	f (g y) x
B' f x g y	→	(f x) (g y)
B* f g h x	→	f (g (h x))
S'' f g x y	→	(f x y) (g x y))
Ln e1 e2 ... en x	→	x e1 e2 ... en
Cn f e1 e2 ... en x	→	(f x) e1 e2 ... en

BENEFITS OF PARAMETRIZED COMBINATORS

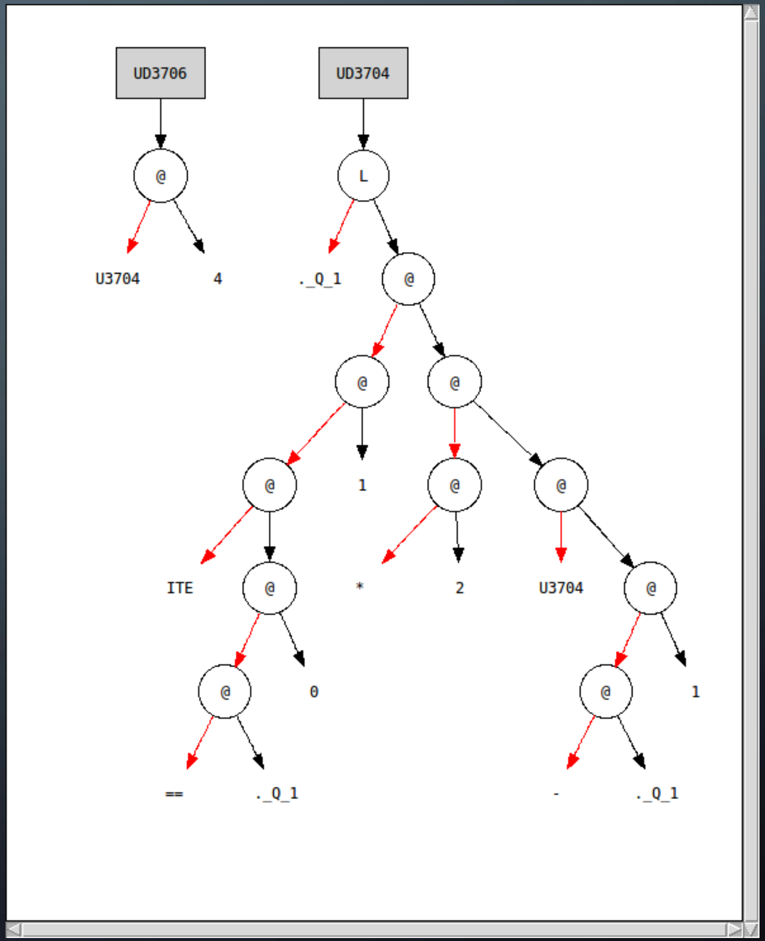
Program	Number of of combinators		Size of ROM image	
	Fixed	Extended	Fixed	Extended
Factorial	13	5(3)	44	28
Dot product	34	13(7)	97	55
Matrix multiplication	45	24(8)	134	88
Neural Ntwk. eval	92	53(16)	267	185
Min. 3-D distance	166	90(19)	405	240

IMPLEMENTATION

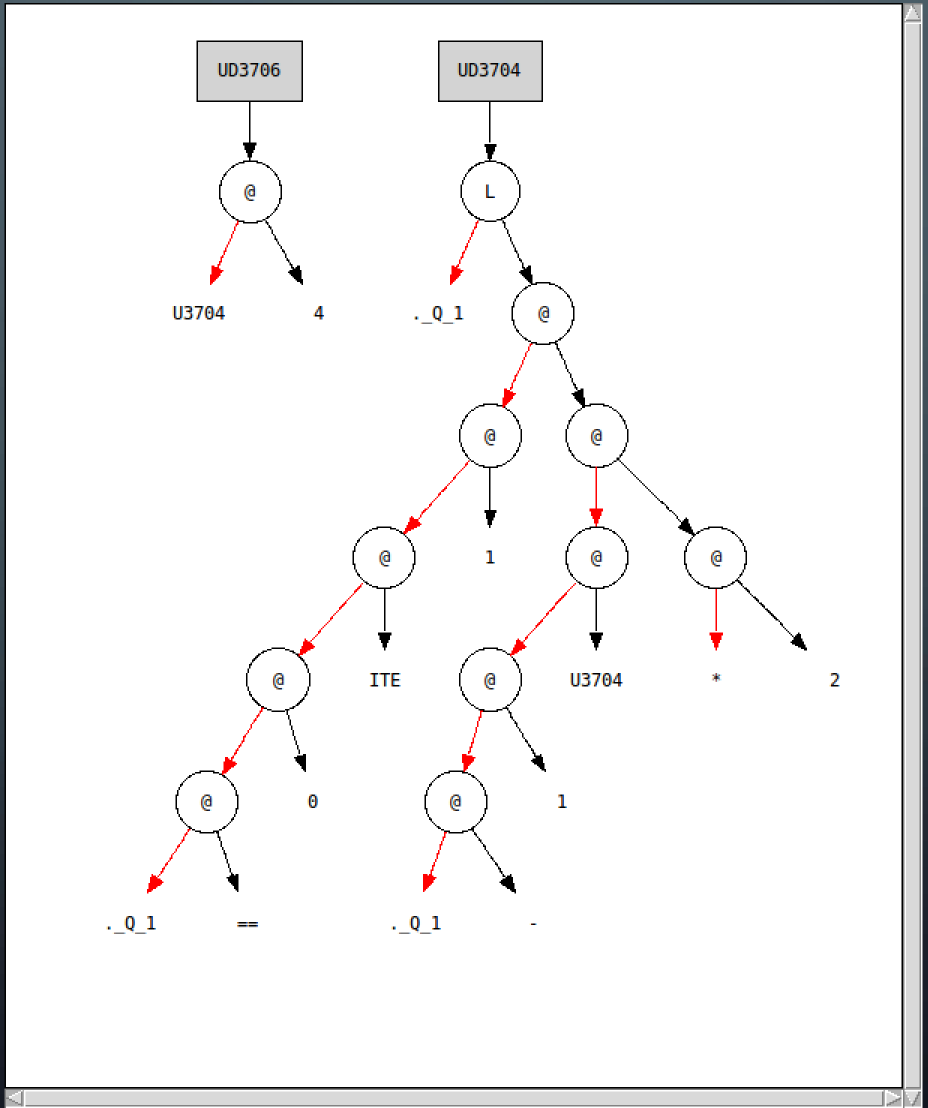
- The graph represents both program and data
    - Every item (graph node) is of the same size
    - Makes memory management much easier but leads to “waste”
- Use back-pointer in node while traversing the spine
    - No evaluation stack needed
    - Switching evaluation process requires only two operations:
        - Swap the top of reduction graph
        - Swap the current location on left spine
    - An aborted graph reduction is resumed with almost no overhead

EXAMPLE OF COMPILATION

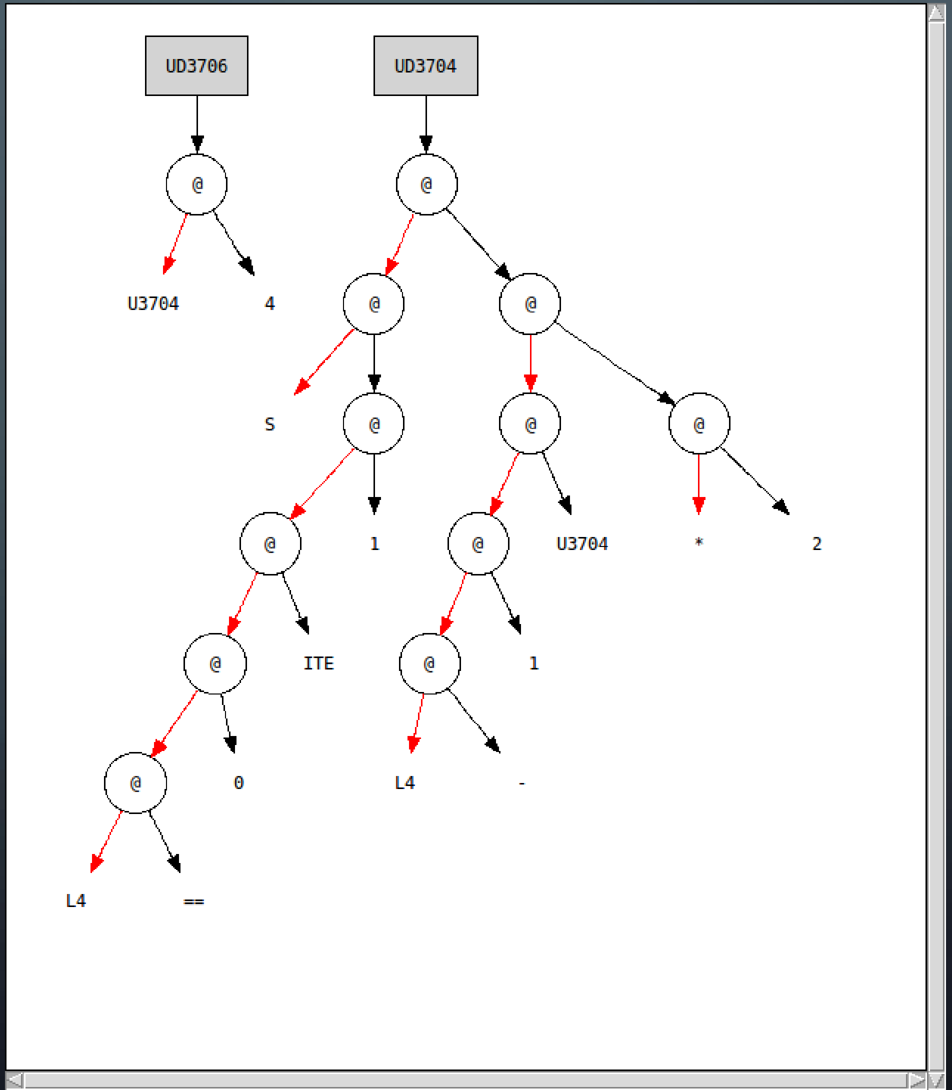
```
forward_declare {pow2 :: int -> int};
let pow2 n =
  n=0 => 1
        | 2*(pow2 (n-1))
;

let main = pow2 4;
```



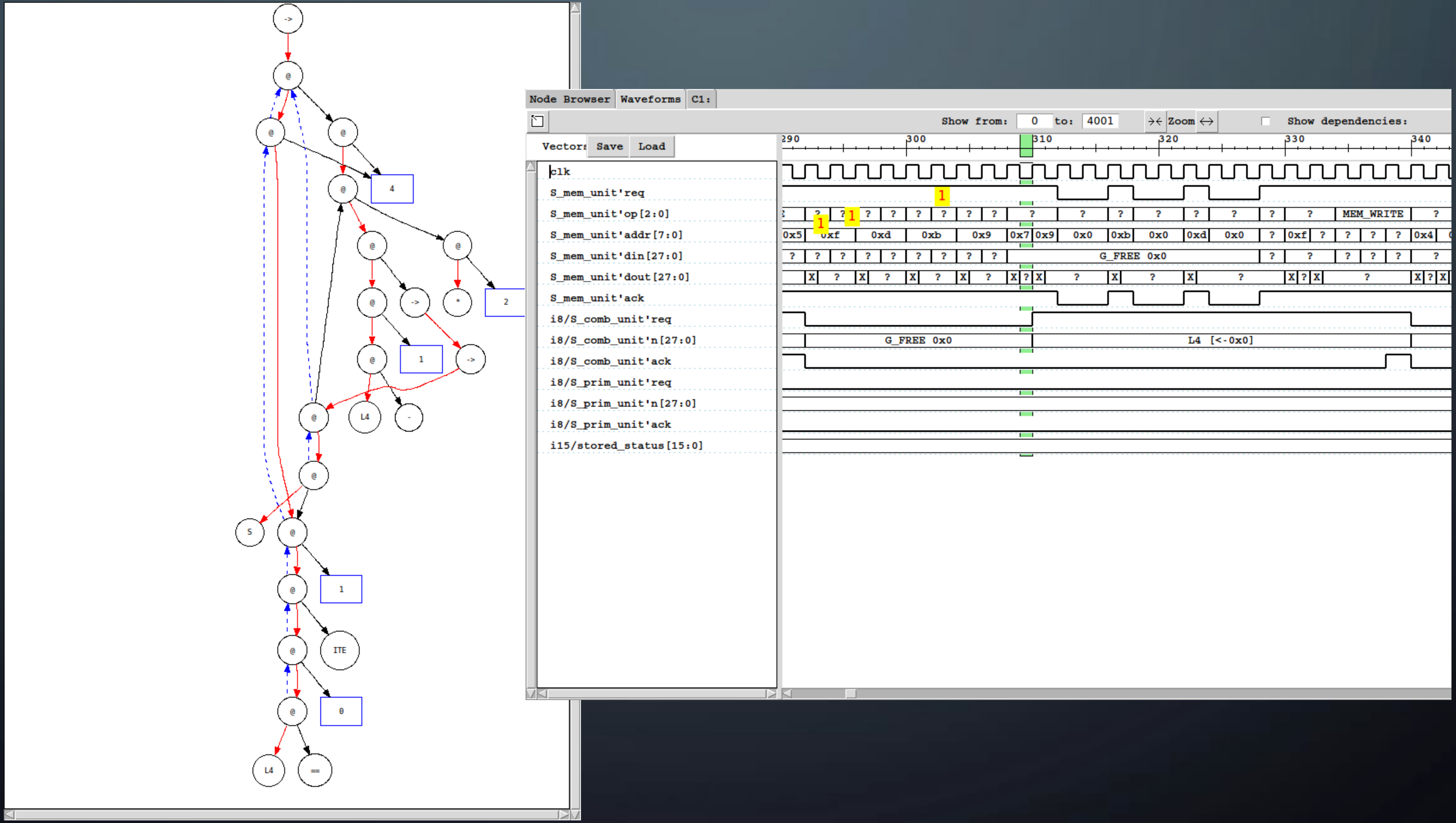
AFTER STRICTNESS TRANSFORMATION



COMPILED TO COMBINATORS

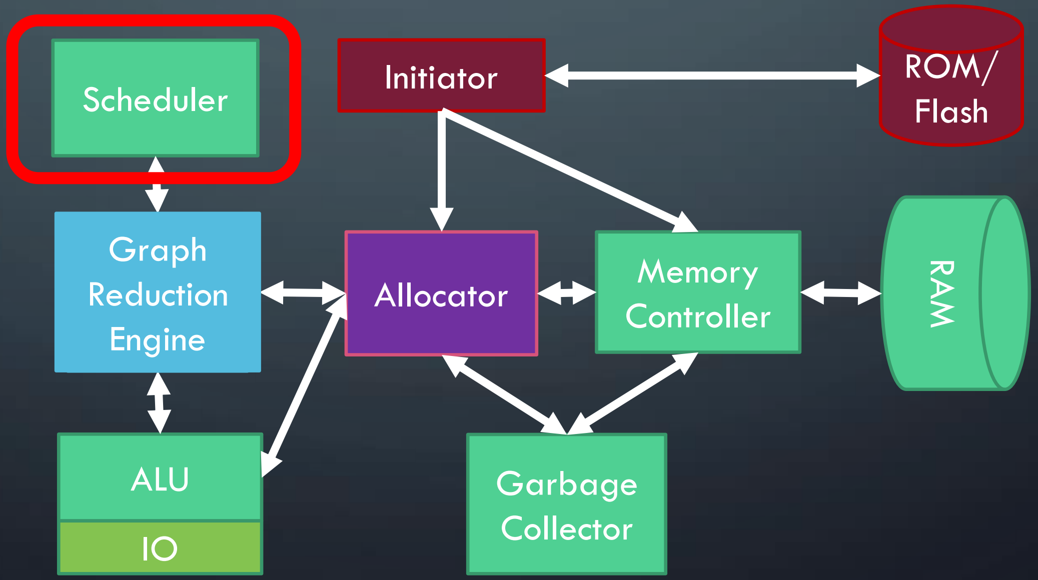


SNAPSHOT DURING REDUCTION



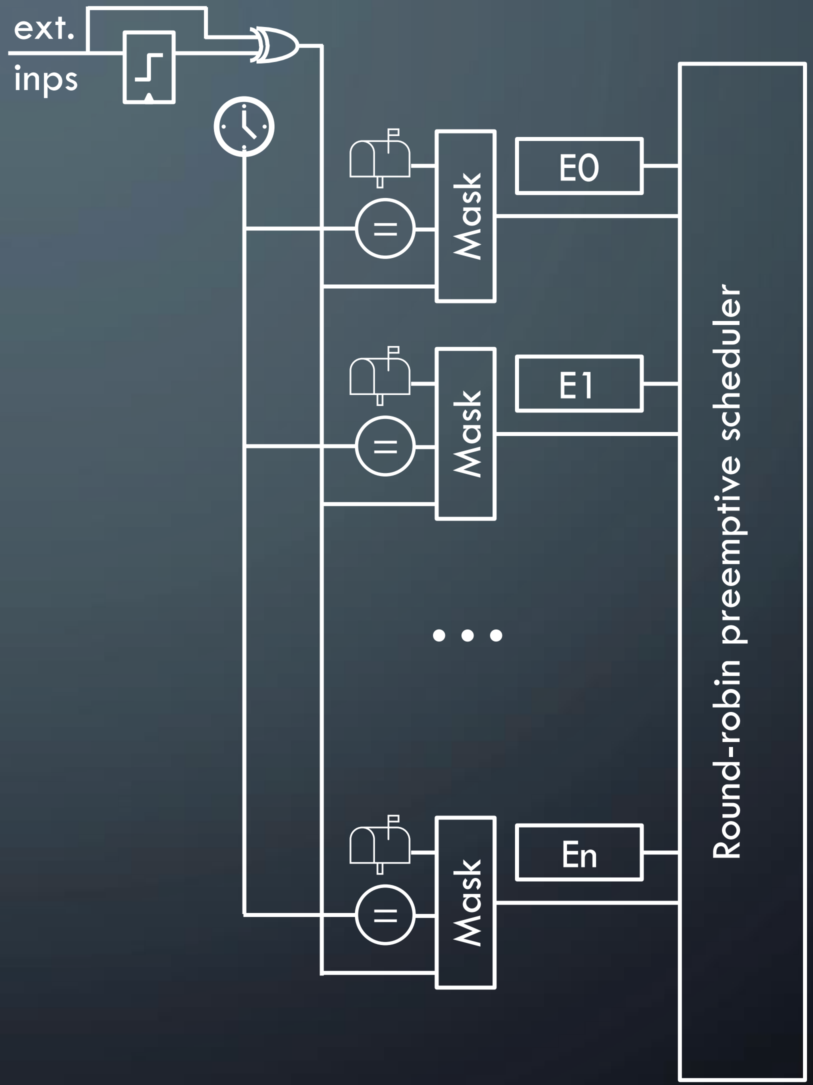
Show from: 0 to: 4001

SCHEDULER

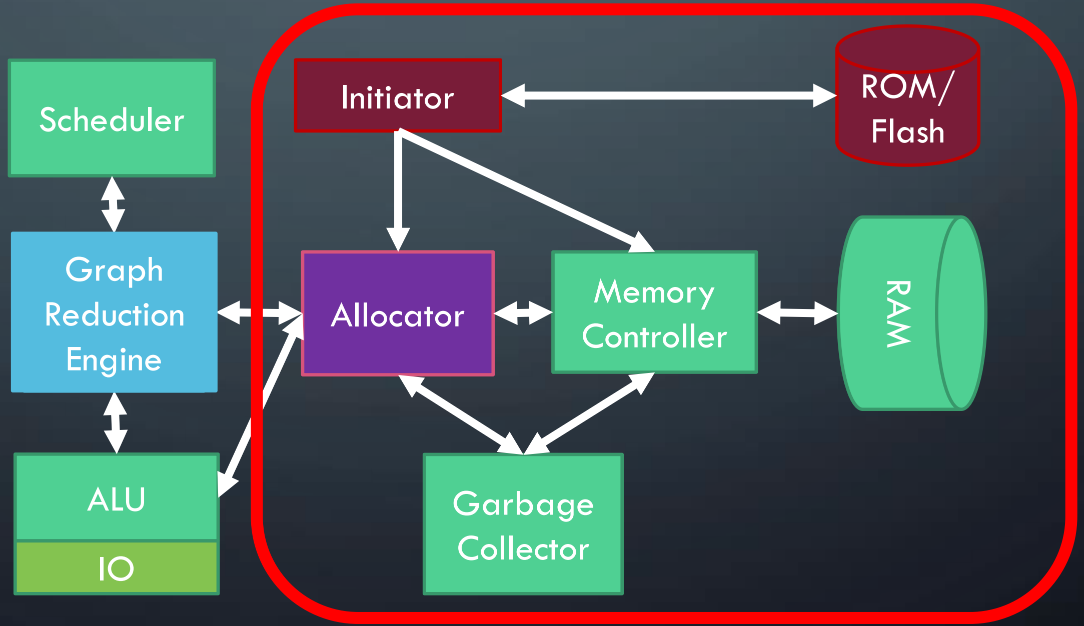

Scheduler
Initiator
ROM/ Flash
Graph Reduction Engine
Allocator
Memory Controller
RAM
ALU
IO
Garbage Collector

SCHEDULER

- HW supported processes.
  A process consists of:
  - An evaluation reduction graph
  - An event time
  - An input “mailbox” as a graph node representing list of msgs
  - A mask used to select activation condition(s)
- Cephalopode uses a round-robin pre-emptive scheduler
- When a graph is fully evaluated, it is removed from the scheduler.



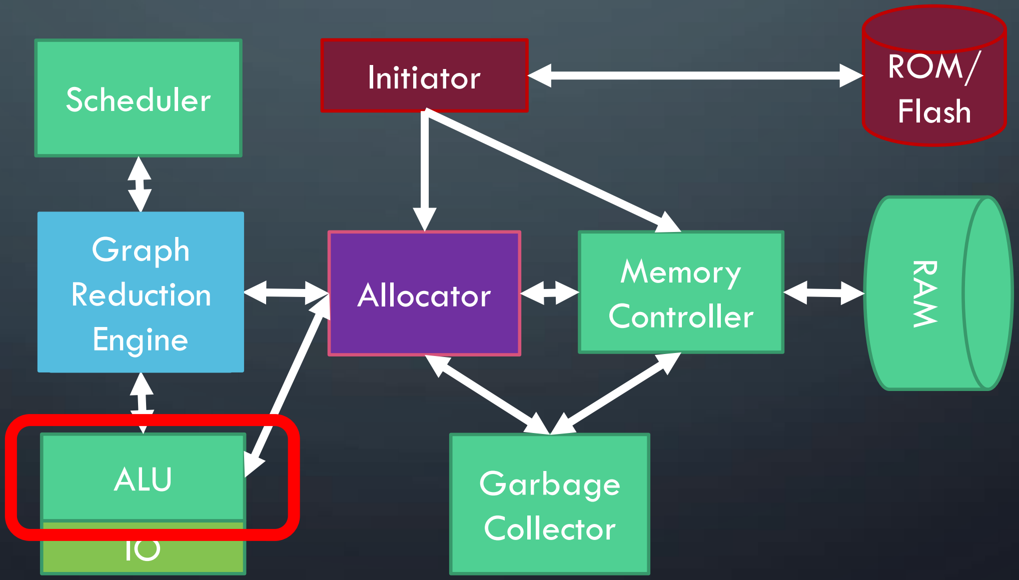
MEMORY SUBSYSTEM



MEMORY SYSTEM

- Copy ROM to RAM since graph will be modified.
- Memory protection between processes automatic since no memory read/write operations available.
    - Messages are fully reduced so will not change.
- Perform concurrent garbage collection
    - G.C. starts by taking a snapshot of the memory.
        - Instead of copying all the nodes, a copy-on-first-write is used.
            - Doubles the memory needed!
    - Now a mark-and-sweep is performed on the snapshot.
    - When sweep is completed, free nodes are returned to free list.
    - Give reduction engine priority to minimize memory contention

ARITHMETIC UNIT



CEPHALOPODE: ARITHMETIC

- To limit overflow issues, all arithmetic is mixed fixed and arbitrary precision.
- An arb. prec. number is represented as a (little-endian) list of chunks.
- Conversion:
  - Integers automatically converted to/from arb. precision when needed.
  - Efficiency of fixed, safety of arbitrary precision.

```
inline function negate: x:index -> z:index {
    var alen   l, i;
    var Aint   carry, wx, wz;
    var bit    mx, mz;

    z = do arb_rf_new;
    l = do arb_rf_length x;
    do arb_rf_alloc z (l+1);
    carry = 1;
    for(i = 0; i < l; i++) {
        wx = ~(do arb_rf_get x i);
        wz = do adder wx carry 0;
        do arb_rf_set z i wz;
        mx = MSB(wx); mz = MSB(wz);
        carry = if (mx & ~mz) then 1 else 0;
    }
    if( ~(MSB(wz) == MSB(wx)) ) {
        do arb_rf_set z l
            (if MSB(wz) then 0 else -1);
    } else {
        do arb_rf_pop z;
        do arbi_trim z;
    }
    return;
}
```

Example of Bifröst code.

STATUS

CEPHALOPODE: STATUS

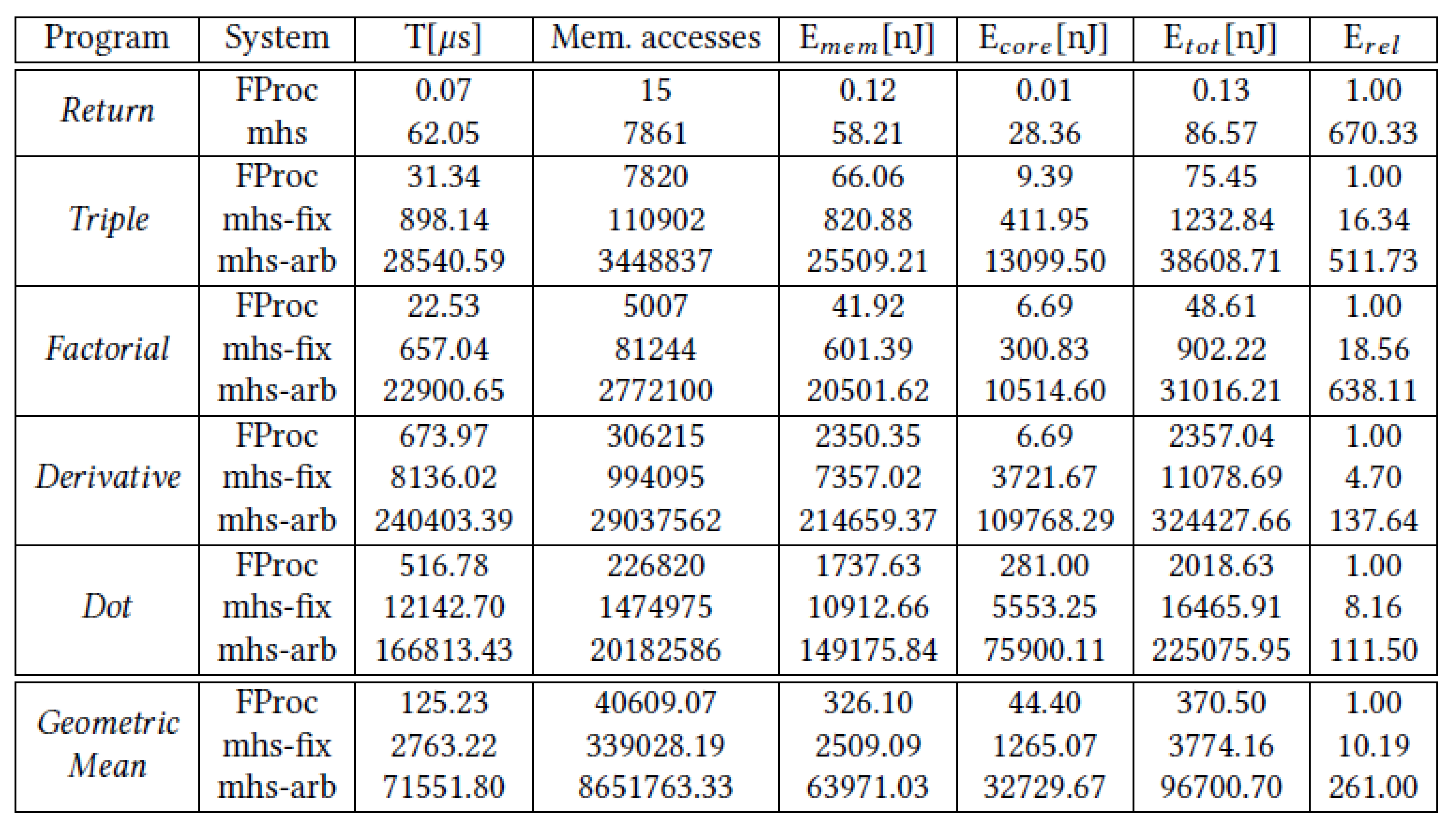
- The second version of Cephalopode completed and has been synthesized and is being evaluated
  - Designed inside the VossII design environment and mapped to RTL which is then going through a traditional synthesis flow.
- A (simple) compiler is in place allowing fl programs to be compiled to ROM images.
  - Almost no optimization in place in the compiler

RESULTS

EVALUATION

- Compare Cephalopode with MicroHaskell (MHS) running on an IoT RISC-V core.
  - MicroHaskel is a new light-weight Haskell compiler + runtime system developed by Lennart Augustsson that uses graph reduction and a fixed set of combinators.
- Energy usage evaluation:
  - Both cores synthesized using the ASAP7 7-nm finFET predictive PDK and standard cell ASIC library [9] with Cadence Genus version 18.14.
  - The library transistors are characterized at the TT corner at a 0.7-V supply voltage and a temperature of 25◦C.
  - Clock frequency: 100 MHz. (iOT device!)
  - A set of benchmarks written in fl & Haskell were developed

INITIAL RESULTS



Program	System	T[μs]	Mem. accesses	E_{mem}[nJ]	E_{core}[nJ]	E_{tot}[nJ]	E_{rel}
Return	FProc	0.07	15	0.12	0.01	0.13	1.00
	mhs	62.05	7861	58.21	28.36	86.57	670.33
Triple	FProc	31.34	7820	66.06	9.39	75.45	1.00
	mhs-fix	898.14	110902	820.88	411.95	1232.84	16.34
	mhs-arb	28540.59	3448837	25509.21	13099.50	38608.71	511.73
Factorial	FProc	22.53	5007	41.92	6.69	48.61	1.00
	mhs-fix	657.04	81244	601.39	300.83	902.22	18.56
	mhs-arb	22900.65	2772100	20501.62	10514.60	31016.21	638.11
Derivative	FProc	673.97	306215	2350.35	6.69	2357.04	1.00
	mhs-fix	8136.02	994095	7357.02	3721.67	11078.69	4.70
	mhs-arb	240403.39	29037562	214659.37	109768.29	324427.66	137.64
Dot	FProc	516.78	226820	1737.63	281.00	2018.63	1.00
	mhs-fix	12142.70	1474975	10912.66	5553.25	16465.91	8.16
	mhs-arb	166813.43	20182586	149175.84	75900.11	225075.95	111.50
Geometric Mean	FProc	125.23	40609.07	326.10	44.40	370.50	1.00
	mhs-fix	2763.22	339028.19	2509.09	1265.07	3774.16	10.19
	mhs-arb	71551.80	8651763.33	63971.03	32729.67	96700.70	261.00

INITIAL RESULTS RELATIVE TO CEPHALOPODE

Program	System	Cycles	Memory accesses	Energy memory	Energy core	Total energy
Tripple	Ceph.	1	1	1	1	1
	mHS fix	28	14	12	44	16
	mHS arb.	911	441	386	1394	512
Factorial	Ceph.	1	1	1	1	1
	mHS fix	29	16	14	44	19
	mHS arb	1016	554	489	1572	638
Derivative	Ceph.	1	1	1	1	1
	mHS fix	12	3	3	556	5
	mHS arb	357	95	91	16407	138
Dot	Ceph.	1	1	1	1	1
	mHS fix	23	7	6	20	8
	mHS arb	323	1412	86	270	111

INITIAL RESULTS RELATIVE TO CEPHALOPODE

Program	System	Cycles	Memory accesses	Energy memory	Energy core	Total energy
Tripple	Ceph.	1	1	1	1	1
	mHS fix	28	14	12	44	16
	mHS arb.	911	441	386	1394	512
Factorial	Ceph.	1	[illegible]	[illegible]	[illegible]	[illegible]
	mHS fix	29	[illegible]	[illegible]	[illegible]	[illegible]
	mHS arb	1016	[illegible]	[illegible]	[illegible]	[illegible]
Derivative	Ceph.	1	[illegible]	[illegible]	[illegible]	[illegible]
	mHS fix	12	[illegible]	[illegible]	[illegible]	[illegible]
	mHS arb	357	95	91	16407	138
Dot	Ceph.	1	1	1	1	1
	mHS fix	23	7	6	20	8
	mHS arb	323	1412	86	270	111

~20x faster than mHS for fixed precision

~500x faster than mHS for arb. prec.

INITIAL RESULTS RELATIVE TO CEPHALOPODE

Program	System	Cycles	Memory accesses	Energy memory	Energy core	Total energy
Tripple	Ceph. mHS fix mHS arb.	1 28 911	1 14 441	1 12 [illegible]	1 44 [illegible]	1 16 [illegible]
Factorial	Ceph. mHS fix mHS arb	1 29 1016	1 16 554	[illegible]	[illegible]	[illegible]
Derivative	Ceph. mHS fix mHS arb	1 12 357	1 3 95	[illegible]	[illegible]	[illegible]
Dot	Ceph. mHS fix mHS arb	1 23 323	1 7 1412	1 6 86	1 20 270	1 8 111

~10x fewer memory accesses than fixed format mHS

~500x fewer memory accesses than arb. prec. mHS

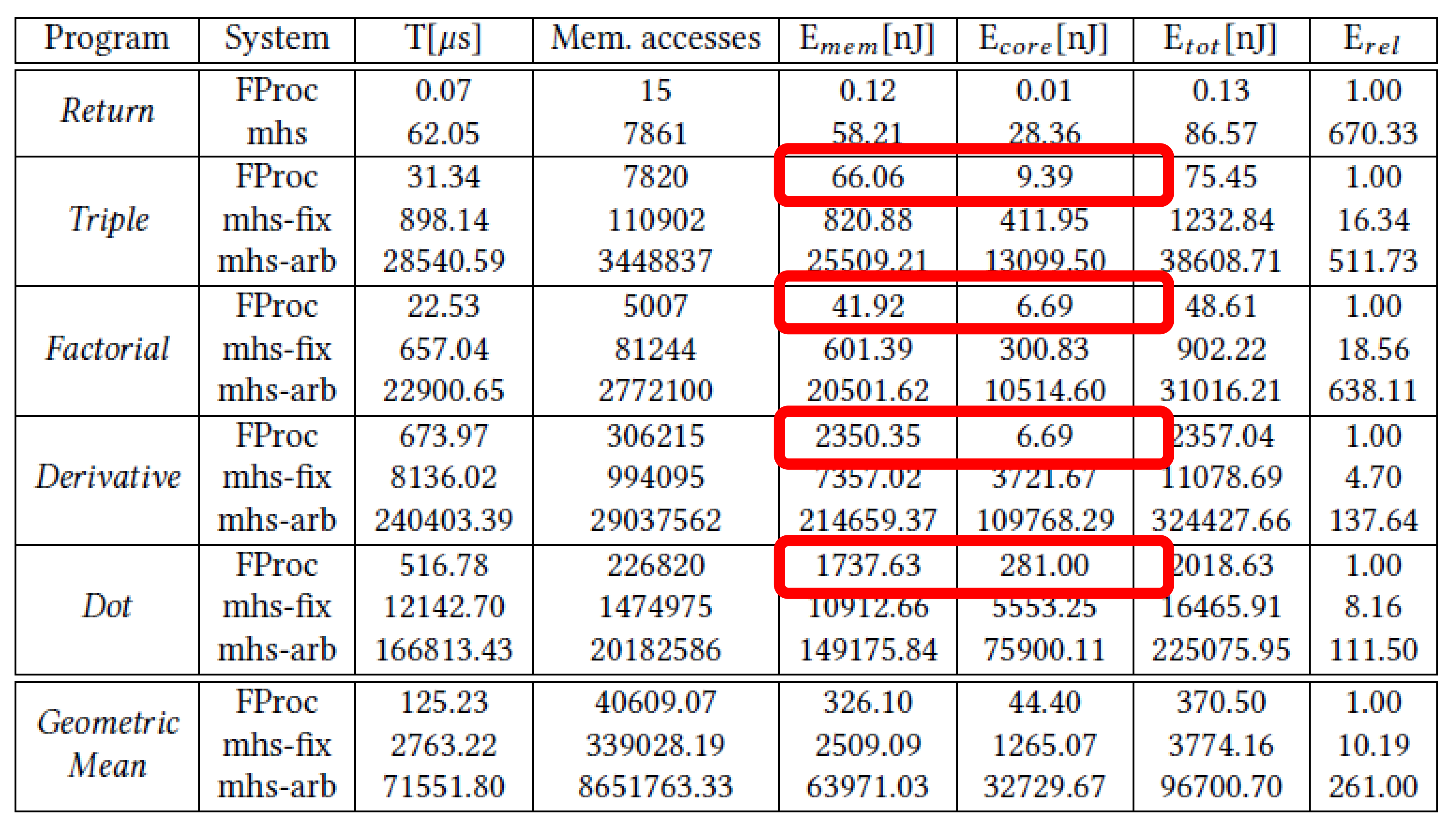
INITIAL RESULTS RELATIVE TO CEPHALOPODE

Program	System	Cycles	Memory accesses	Energy memory	Energy core	Total energy
Tripple	Ceph. mHS fix mHS arb.	1 28 911	1 14 441	1 12 386	1 44 1394	1 16 512
						1 19 638
	mHS arb	357	95	91	18407	1 5 138
Dot	Ceph. mHS fix mHS arb	1 23 323	1 7 1412	1 6 86	1 20 270	1 8 111

~10x less energy than fixed format mHS

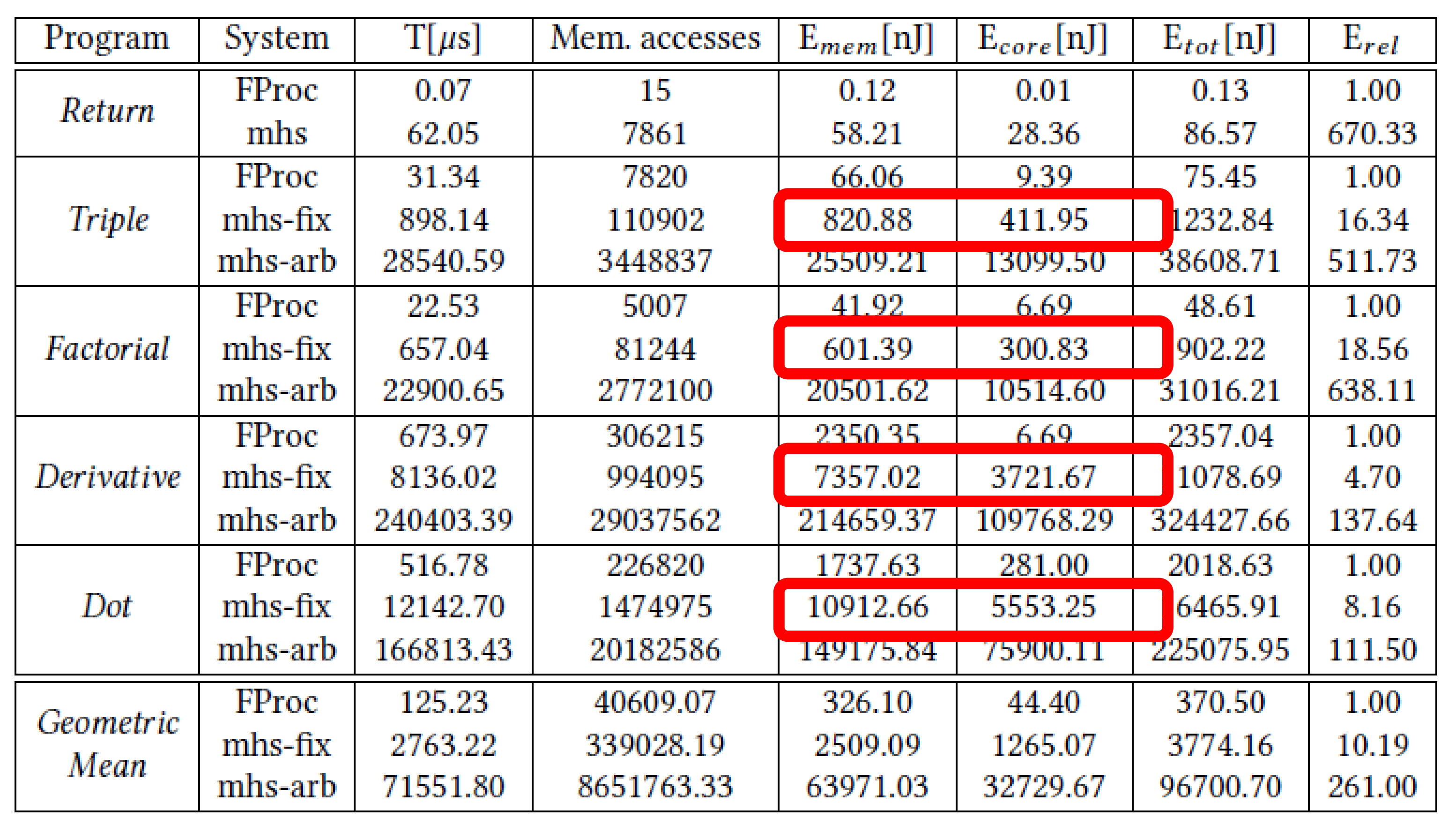
~300 less energy than arb. prec. mHS

RESULTS IN MORE DEPTH: MEMORY VS CORE CEPHALOPODE



Program	System	T[μs]	Mem. accesses	E$_{mem}$[nJ]	E$_{core}$[nJ]	E$_{tot}$[nJ]	E$_{rel}$
Return	FProc	0.07	15	0.12	0.01	0.13	1.00
	mhs	62.05	7861	58.21	28.36	86.57	670.33
Triple	FProc	31.34	7820	66.06	9.39	75.45	1.00
	mhs-fix	898.14	110902	820.88	411.95	1232.84	16.34
	mhs-arb	28540.59	3448837	25509.21	13099.50	38608.71	511.73
Factorial	FProc	22.53	5007	41.92	6.69	48.61	1.00
	mhs-fix	657.04	81244	601.39	300.83	902.22	18.56
	mhs-arb	22900.65	2772100	20501.62	10514.60	31016.21	638.11
Derivative	FProc	673.97	306215	2350.35	6.69	2357.04	1.00
	mhs-fix	8136.02	994095	7357.02	3721.67	11078.69	4.70
	mhs-arb	240403.39	29037562	214659.37	109768.29	324427.66	137.64
Dot	FProc	516.78	226820	1737.63	281.00	2018.63	1.00
	mhs-fix	12142.70	1474975	10912.66	5553.25	16465.91	8.16
	mhs-arb	166813.43	20182586	149175.84	75900.11	225075.95	111.50
Geometric Mean	FProc	125.23	40609.07	326.10	44.40	370.50	1.00
	mhs-fix	2763.22	339028.19	2509.09	1265.07	3774.16	10.19
	mhs-arb	71551.80	8651763.33	63971.03	32729.67	96700.70	261.00

RESULTS IN MORE DEPTH: MEMORY VS CORE MICROHASKELL ON RISC-V



Program	System	T[μs]	Mem. accesses	E_{mem}[nJ]	E_{core}[nJ]	E_{tot}[nJ]	E_{rel}
Return	FProc	0.07	15	0.12	0.01	0.13	1.00
	mhs	62.05	7861	58.21	28.36	86.57	670.33
Triple	FProc	31.34	7820	66.06	9.39	75.45	1.00
	mhs-fix	898.14	110902	820.88	411.95	1232.84	16.34
	mhs-arb	28540.59	3448837	25509.21	13099.50	38608.71	511.73
Factorial	FProc	22.53	5007	41.92	6.69	48.61	1.00
	mhs-fix	657.04	81244	601.39	300.83	902.22	18.56
	mhs-arb	22900.65	2772100	20501.62	10514.60	31016.21	638.11
Derivative	FProc	673.97	306215	2350.35	6.69	2357.04	1.00
	mhs-fix	8136.02	994095	7357.02	3721.67	1078.69	4.70
	mhs-arb	240403.39	29037562	214659.37	109768.29	324427.66	137.64
Dot	FProc	516.78	226820	1737.63	281.00	2018.63	1.00
	mhs-fix	12142.70	1474975	10912.66	5553.25	6465.91	8.16
	mhs-arb	166813.43	20182586	149175.84	75900.11	225075.95	111.50
Geometric Mean	FProc	125.23	40609.07	326.10	44.40	370.50	1.00
	mhs-fix	2763.22	339028.19	2509.09	1265.07	3774.16	10.19
	mhs-arb	71551.80	8651763.33	63971.03	32729.67	96700.70	261.00

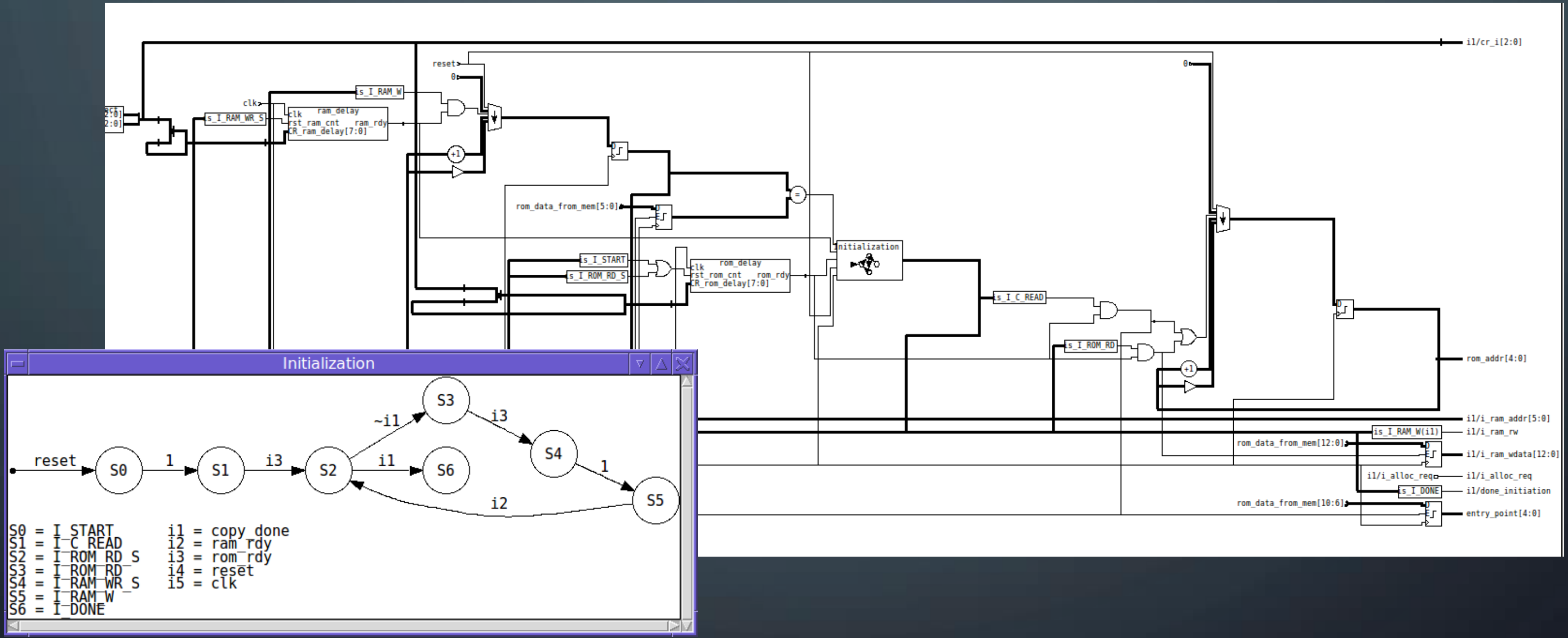
DISCUSSION

- What about state-of-the-art functional compiler?
  - If we manage to get a runtime environment of GHC to run on the tiny RISC-V processor we are using, we estimate
    - GHC unoptimized uses about the same energy as Cephalopode if we let GHC use fixed arithmetic
    - GHC unoptimized uses about 2x more power than Cephalopode
- How much do we pay for using a high-level language?
  - C code or Rust or…?
- Benchmarking the full system is challenging:
  - Need to compare with a (partial) operating system

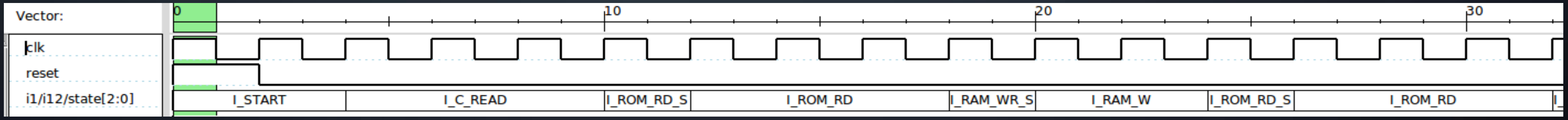
CONCLUSIONS & FUTURE WORK

- Custom hardware for functional language execution aimed at low-power IoT devices can provide
  - At least an order of magnitude reduced energy consumption compared with running functional language on traditional CPU.
  - Process isolation, memory safety and overflow protection to drastically reduce common IoT software problems.
- Cephalopode only scratches the surface– Rich field of research to be explored!
  - Better design -- Reduce memory traffic critical
  - Compiler technology – Different set of combinators?

THANK YOU



```
S0 = I_START       i1 = copy_done
S1 = I_C_READ      i2 = ram_rdy
S2 = I_ROM_RD_S    i3 = rom_rdy
S3 = I_ROM_RD      i4 = reset
S4 = I_RAM_WR_S    i5 = clk
S5 = I_RAM_W
S6 = I_DONE
```


Vector:
0
10
20
30
clk
reset
i1/i12/state[2:0]
I_START
I_C_READ
I_ROM_RD_S
I_ROM_RD
I_RAM_WR_S
I_RAM_W
I_ROM_RD_S
I_ROM_RD
I_