

A Lisp for microcontrollers

Joel Svensson

Chalmers FP-Talk Dec 8 2023



LBM (LispBM)

- A lisp dialect for microcontrollers.
 - 32bit architectures.
 - 128KB of ram or more. Maybe a bit less even.
 - Flash storage.
 - Usually no cache between CPU and RAM.
 - Sometimes cache/accelerator between CPU and flash.
- Intended to run concurrently with a C application.

DEMO

Five years of fun, so far.

- How it began in 2018
 - SICP videos on youtube.
 - Microcontrollers at work.



- When it “took off”
 - 2022-12-09: VESC firmware version 6.0.
 - Thank you Benjamin Vedder.



Added lispbm test module (disabled by default)



vedderb committed on Jan 13, 2022

Features

- GC.
 - Stack based MS.
 - Pointer reversal MS.
- Call-CC.
- QQ
 - Quasiquotation in Lisp – Bawden.
- Macros.
- Different modes of reading.
- Message passing.
- Concurrency.
- Pattern matching.
- Byte arrays.
- Flash storage.
 - Programs and data.
- Profiler.

Retired features

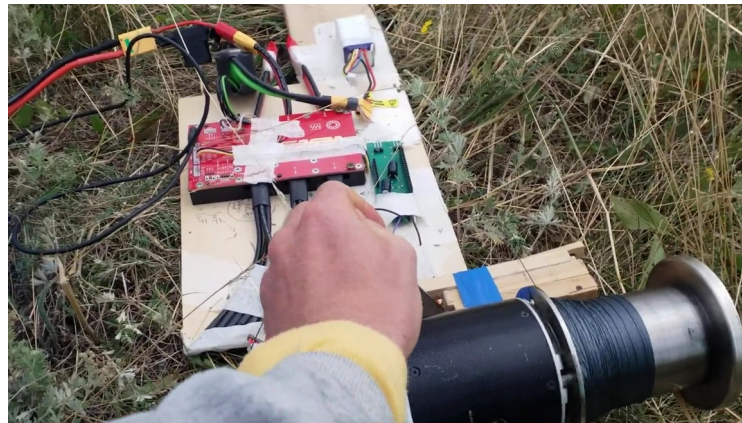
- Namespaces.
- Partial Application.
- Some other array types.
- Wait-for flags.
- Cooperative scheduling.

- Goals now:
 - Small and somewhat efficient language.
 - Sandboxed evaluation of code.
 - Add scripting capabilities to application X.
 - Buggy applications should not crash X.
- Original goals:
 - Have fun.



Luke F:

<https://www.youtube.com/watch?v=QNGDMCOsarM>



Kites for future:

<https://www.youtube.com/watch?v=pU08gltGpAs>



Alexander Krasnov:

<https://github.com/aka13-404/VSETT-LISP>







<https://github.com/leocelente/vesc-rs>
485-lispbm

<https://github.com/tonymillion/VescNi>
nebotDash

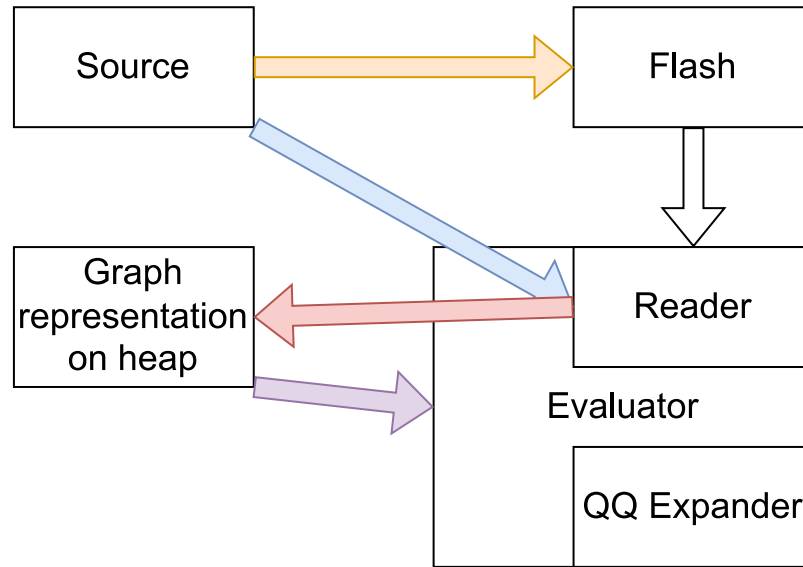
https://github.com/m365fw/vesc_m365_dash



lispBM language support

Rasmus Söderhielm |  20 installs |      (0) | Free

Overview



Overview

```
208 env.c
4370 eval_cps.c
 147 extensions.c
1319 fundamental.c
1344 heap.c
 424 lbm_channel.c
 297 lbm_c_interop.c
  52 lbm_custom_type.c
  32 lbm_flags.c
 696 lbm_flat_value.c
 457 lbm_memory.c
 118 lbm_prof.c
  79 lbm_variables.c
  51 lispbm.c
 421 print.c
 118 stack.c
 493 symrepr.c
 529 tokpar.c
11155 total
```

Evaluation of expressions

```
data Exp = Num Int
         | Add Exp Exp
```

```
type Cont = Int -> Int
```

```
myExp = Add (Num 2) (Num 3)
myExp2 = Add myExp myExp
```

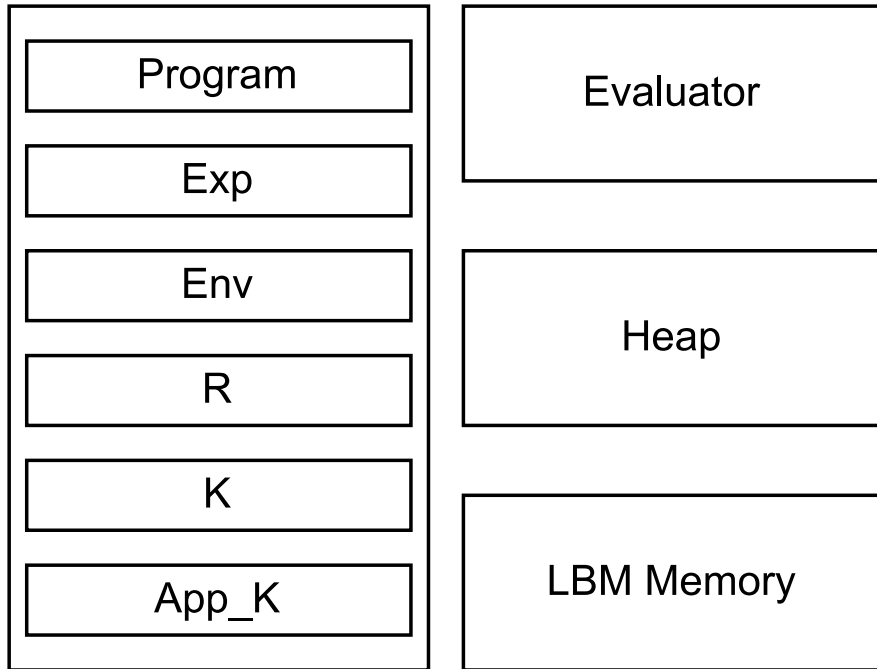
```
eval :: Cont -> Exp -> Int
eval c (Num a) = c a
eval c (Add a b) =
    eval (\v ->
          eval (\v1 -> c (v + v1)) b) a
```

Evaluation of expressions

```
*Main> eval id myExp  
5  
*Main> eval id myExp2  
10
```

```
myExp = Add (Num 2) (Num 3)  
myExp2 = Add myExp myExp
```

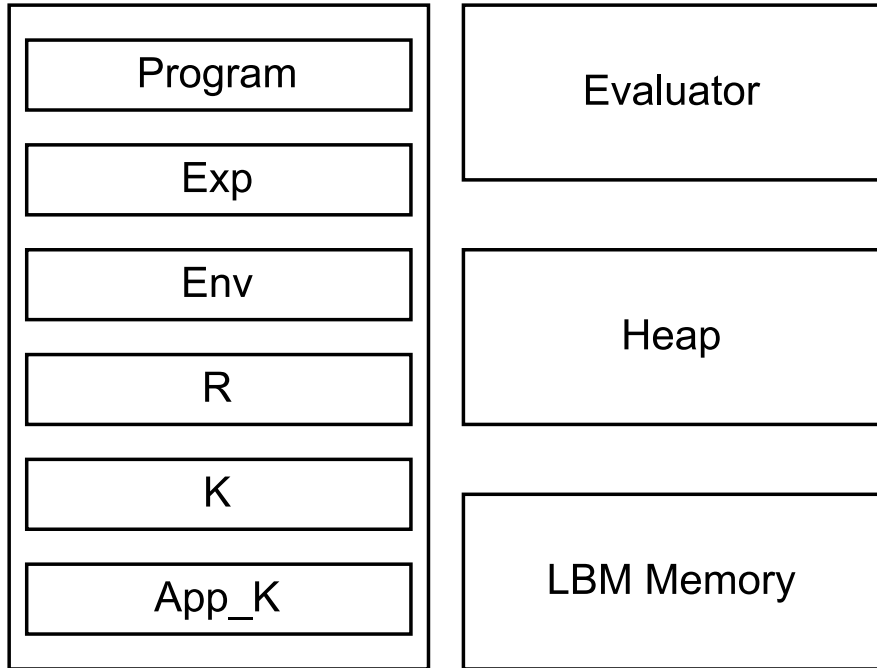
LBM Evaluator Architecture



Eval loop

```
while (true) {  
    if (App_K) {  
        apply_cont();  
    } else {  
        /* pattern match on Exp */  
    }  
}
```

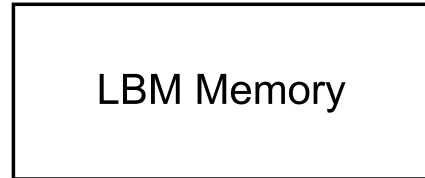
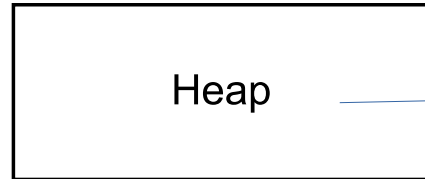
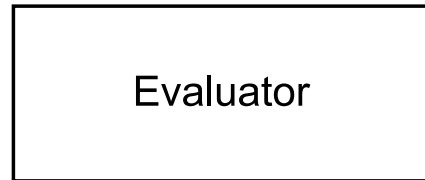
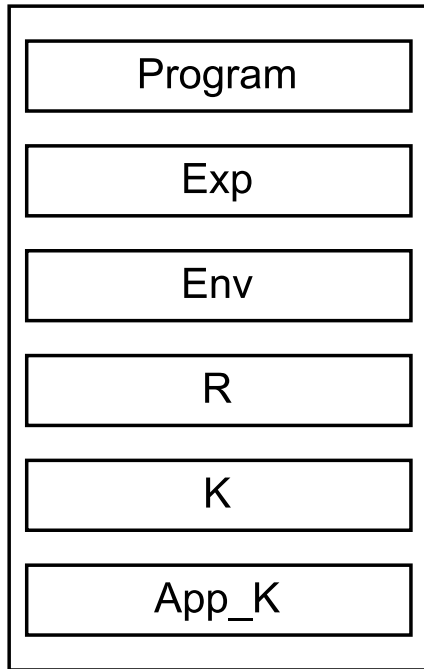
LBM Evaluator Architecture



Evaluate this!

```
(define a 10)
(define b 20)
(+ a b)
```

LBM Evaluator Architecture

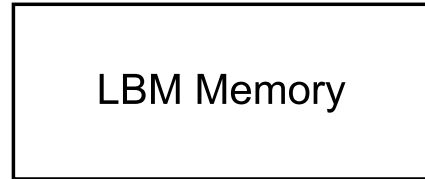
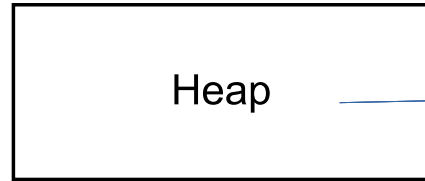
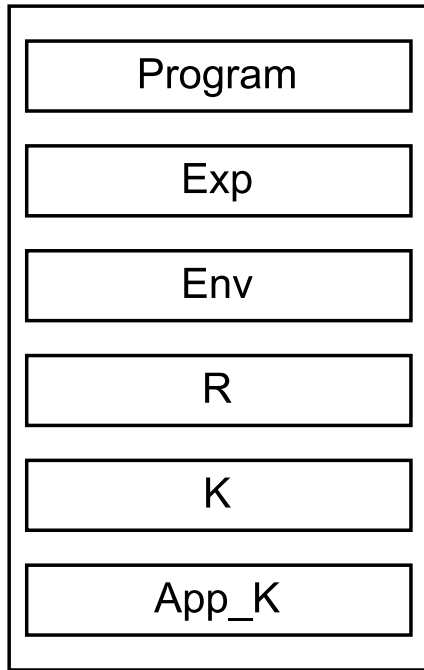


Evaluate this!

```
(define a 10)  
(define b 20)  
(+ a b)
```

→ `((define a 10) (define b 20) (+ a b))`

LBM Evaluator Architecture



Evaluate this!

```
(define a 10)
(define b 20)
(+ a b)
```

→ `((define a 10) (define b 20) (+ a b))`

```
Program <= ((define b 20) (+ a b))
Exp <= (define a 10)
Env <= Nil
R <= Nil
K <= Done
App_K <= False
```

```
Program <= ((define b 20) (+ a b))
Exp <= (define a 10)
Env <= Nil
R <= Nil
K <= Done
App_K <= False
```

```
Program = ((define b 20) (+ a b))
Exp = 10
Env = Nil
R <= 10
K = Done | a | do_define
App_K <= True
```

eval_define

eval 10

apply_cont

```
Program = ((define b 20) (+ a b))
Exp <= 10
Env = Nil
R = Nil
K <= Done | a | do_define
App_K = False
```

```
Program = ((define b 20) (+ a b))
Exp = 10
Env = Nil
R = 10
K <= Done
App_K = True
```

```
Program = ((define b 20) (+ a b))  
Exp = 10  
Env = Nil  
R = 10  
K <= Done  
App_K = True
```

Repeat earlier steps

apply_cont

```
Program <= ((+ a b))  
Exp <= (define b 20)  
Env <= Nil  
R <= Nil  
K <= Done  
App_K <= False
```

```
Program <= Nil
Exp <= (+ a b)
Env <= Nil
R <= Nil
K <= Done
App_K = False
```

eval_app

```
Program <= Nil
Exp <= +
Env <= Nil
R <= Nil
K <= Done | APPLICATION_START (a b)
App_K = False
```

eval +

```
Program <= Nil
Exp <= +
Env <= Nil
R <= +
K <= Done | APPLICATION_START (a b)
App_K = True
```

apply_cont

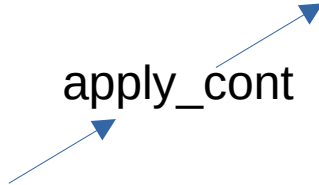
```
Program <= Nil
Exp <= a
Env <= Nil
R <= +
K <= Done | + | APPLICATION_ARGS (b)
App_K = False
```

```
Program <= Nil
Exp <= a
Env <= Nil
R <= +
K <= Done | + | APPLICATION_ARGS (b)
App_K = False
```

lookup a

```
Program <= Nil
Exp <= a
Env <= Nil
R <= 10
K <= Done | + | APPLICATION_ARGS (b)
App_K = True
```

apply_cont



```
Program <= Nil
Exp <= b
Env <= Nil
R <= 10
K <= Done | + | 10 | APPLICATION_ARGS Nil
App_K = False
```

lookup b

```
Program <= Nil
Exp <= b
Env <= Nil
R <= 20
K <= Done | + | 10 APPLICATION_ARGS Nil
App_K = True
```



Program <= Nil

Exp <= b

Env <= Nil

R <= 30

K <= Done

App_K = True

```

// (define sym exp)
static void eval_define(eval_context_t *ctx) {
    lbm_value args = get_cdr(ctx->curr_exp);
    lbm_value key, rest_args;
    get_car_and_cdr(args, &key, &rest_args);
    lbm_value val_exp, rest_val;
    get_car_and_cdr(rest_args, &val_exp, &rest_val);
    lbm_uint *sptr = stack_reserve(ctx, 2);
    if (lbm_is_symbol(key) && lbm_is_symbol_nil(rest_val)) {
        lbm_uint sym_val = lbm_dec_sym(key);
        sptr[0] = key;
        if (sym_val >= RUNTIME_SYMBOLS_START) {
            sptr[1] = SET_GLOBAL_ENV;
            if (ctx->flags & EVAL_CPS_CONTEXT_FLAG_CONST) {
                stack_push(&ctx->K, MOVE_VAL_TO_FLASH_DISPATCH);
            }
            ctx->curr_exp = val_exp;
            return;
        }
    }
    error_at_ctx(ENC_SYM_EERROR, ctx->curr_exp);
}


```

```
static void cont_set_global_env(eval_context_t *ctx){  
  
    lbm_value key;  
    lbm_value val = ctx->r;  
  
    lbm_pop(&ctx->K, &key);  
    lbm_value new_env;  
    // A key is a symbol and should not need to be remembered.  
    WITH_GC(new_env, lbm_env_set(*lbm_get_env_ptr(),key,val));  
  
    *lbm_get_env_ptr() = new_env;  
    ctx->r = val;  
  
    ctx->app_cont = true;  
  
    return;  
}
```


Pattern match on Exp

- Symbol
 - Look it up
- (Special-form $e_1 \dots e_n$)
 - define, lambda. The built-in syntax essentially
- $(x e_1 \dots e_n)$ - General application form
 - Closure, Continuation (from call/cc), Fundamental-operation,
 - Extension, something I call an “apply_fun”.
- Anything else

```
while (true) {  
  if (App_K) {  
    apply_cont();  
  } else {  
    /* pattern match on Exp */  
  }  
}
```



```
static const evaluator_fun
evaluators[] =
{
    eval_quote,
    eval_define,
    eval_progn,
    eval_lambda,
    eval_if,
    eval_let,
    eval_and,
    eval_or,
    eval_match,
    eval_receive,
    eval_receive_timeout,
    eval_callcc,
    eval_atomic,
    eval_selfevaluating, // macro
    eval_selfevaluating, // cont
    eval_selfevaluating, // closure
    eval_cond,
    eval_app_cont,
    eval_var,
    eval_setq,
    eval_move_to_flash,
    eval_loop,
};
```

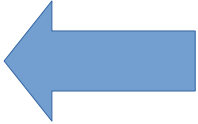
```
static const apply_fun fun_table[] =
{
    apply_setvar,
    apply_read,
    apply_read_program,
    apply_read_eval_program,
    apply_spawn,
    apply_spawn_trap,
    apply_yield,
    apply_wait,
    apply_eval,
    apply_eval_program,
    apply_send,
    apply_ok,
    apply_error,
    apply_map,
    apply_reverse,
    apply_flatten,
    apply_unflatten,
    apply_kill,
    apply_sleep,
};
```

```
const fundamental_fun fundamental_table[] =
    {fundamental_add,
      fundamental_sub,
      fundamental_mul,
      fundamental_div,
      fundamental_mod,
      fundamental_eq,
      fundamental_not_eq,
      fundamental_num_eq,
      fundamental_num_not_eq,
      fundamental_lt,
      fundamental_gt,
      fundamental_leq,
      fundamental_geq,
      fundamental_not,
      fundamental_gc,
      fundamental_self,
      fundamental_set_mailbox_size,
      fundamental_cons,
      fundamental_car,
      fundamental_cdr,
      fundamental_list,
      ...
```

```
static const cont_fun continuations[NUM_CONTINUATIONS] =
{ advance_ctx, // CONT_DONE
  cont_set_global_env,
  cont_bind_to_key_rest,
  cont_if,
  cont_progn_rest,
  cont_application_args,
  cont_and,
  cont_or,
  cont_wait,
  cont_match,
  cont_application_start,
  cont_eval_r,
  cont_set_var,
  cont_resume,
  cont_closure_application_args,
  cont_exit_atomic,
  cont_read_next_token,
  cont_read_append_continue,
  cont_read_eval_continue,
  cont_read_expect_closepar,
  cont_read_dot_terminate,
  cont_read_done,
  cont_read_quote_result,
  cont_read_commaat_result,
  cont_read_comma_result,
  cont_read_start_array,
  cont_read_append_array,
  cont_map,
  cont_match_guard,
  cont_terminate,
  cont_progn_var,
  cont_setq,
  cont_move_to_flash,
  cont_move_val_to_flash_dispatch,
  cont_move_list_to_flash,
```

```
  cont_close_list_in_flash,
  cont_qq_expand_start,
  cont_qq_expand,
  cont_qq_append,
  cont_qq_expand_list,
  cont_qq_list,
  cont_kill,
  cont_loop,
  cont_loop_condition,
};
```

```
while (true) {
  if (App_K) {
    apply_cont();
  } else {
    /* pattern match on Exp */
  }
}
```



Values

- 4 value types:
 - 28Bit Integers, unsigned and signed, characters and symbols.
 $B_{31}..B_4T_1T_0G0$ - B is the value.
- Lots of pointer types (Boxed values)
 - $T_5T_4T_3T_2T_1T_0B_{25}..B_2G1$ - B is an index into the heap.
 - 32Bit values.
 - 64 Bit values.
 - Float.
 - Double.

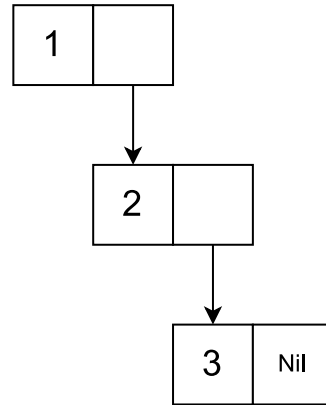
Memory

- Heap
- Buffer memory - “LBM_Memory”
- Flash Storage

Heap

- An array of cells with two fields
 - The car and the cdr. fst/snd. Each 32bit.

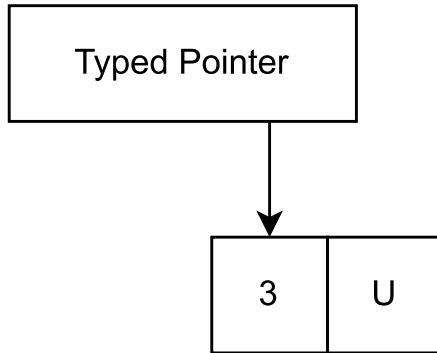
(list 1 2 3)



Heap

- A 32bit unsigned

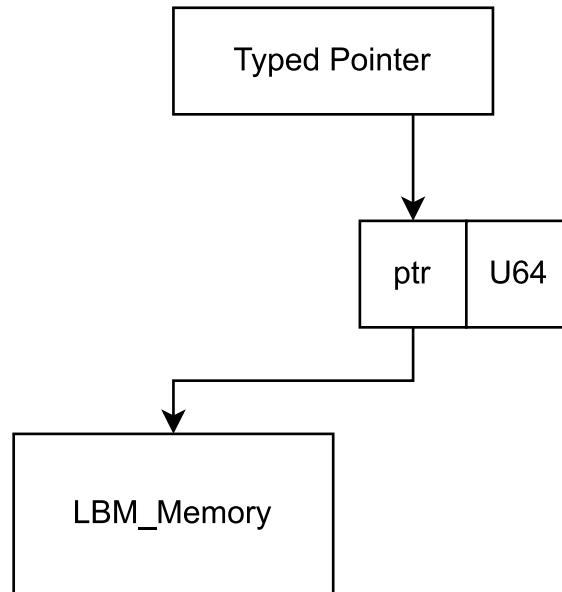
3u32



Heap

- A 64bit unsigned

3u64



LBM_Memory

- A memory where N 32 bit words can be allocated and freed. Malloc/free style.
- Lisp values that are larger than a heap cell:
 - (pointer_into_lbm_mem . special_id_symbol)
 - GC calls free on these when not needed.
 - GC does not recurse into values stored in lbm memory.

Flash Storage

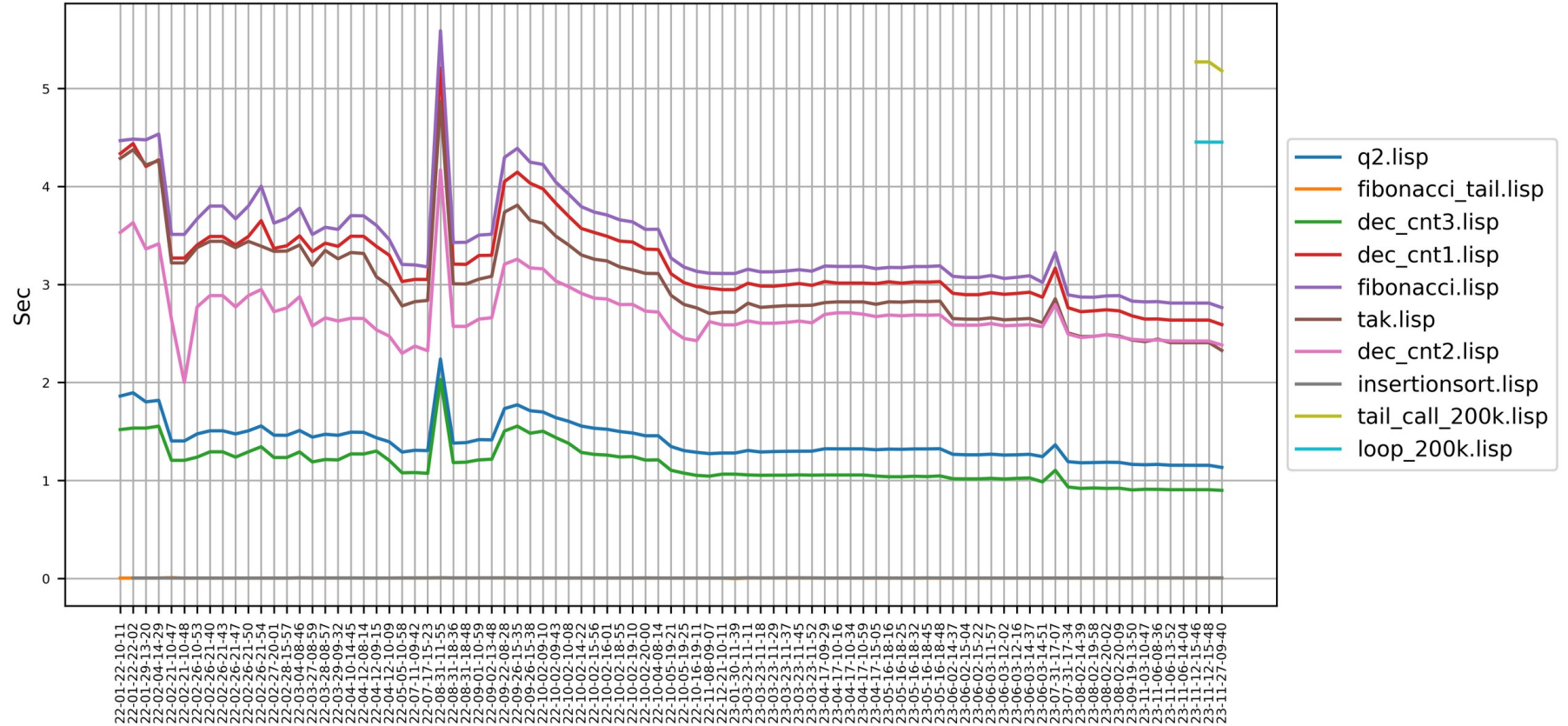
- Cleared in blocks
- Typically value 0xFF in a cleared byte
- Bits can flipped to 0 but not back to 1 individually.

Flash Storage

```
lbm_flash_status lbm_allocate_const_cell(lbm_value *res)  
lbm_flash_status lbm_write_const_raw(lbm_uint *data, lbm_uint n, lbm_uint *res)  
lbm_flash_status write_const_cdr(lbm_value cell, lbm_value val)  
lbm_flash_status write_const_car(lbm_value cell, lbm_value val)
```

Performance over time

STM32F4 160MHz



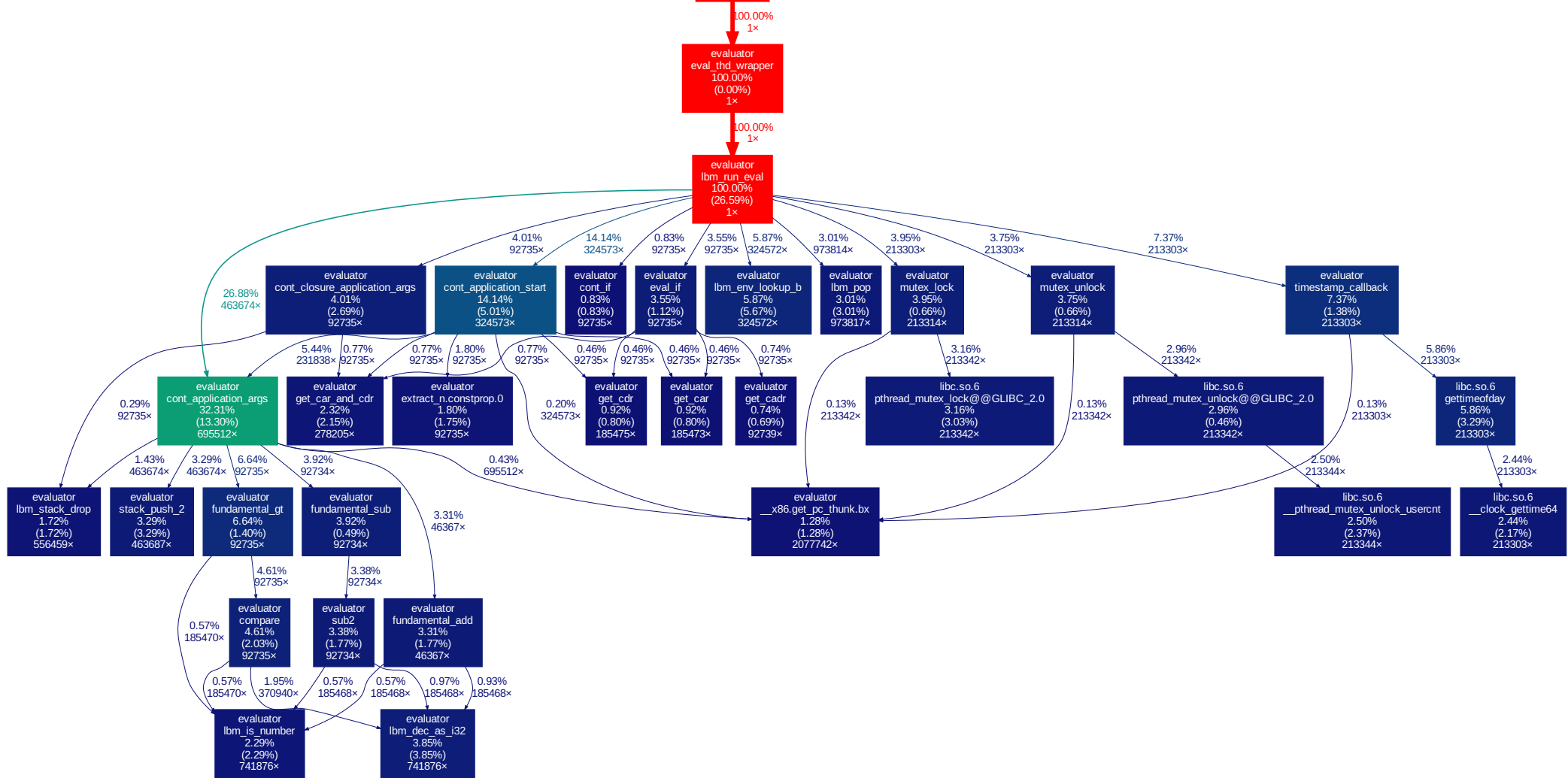
Nfib

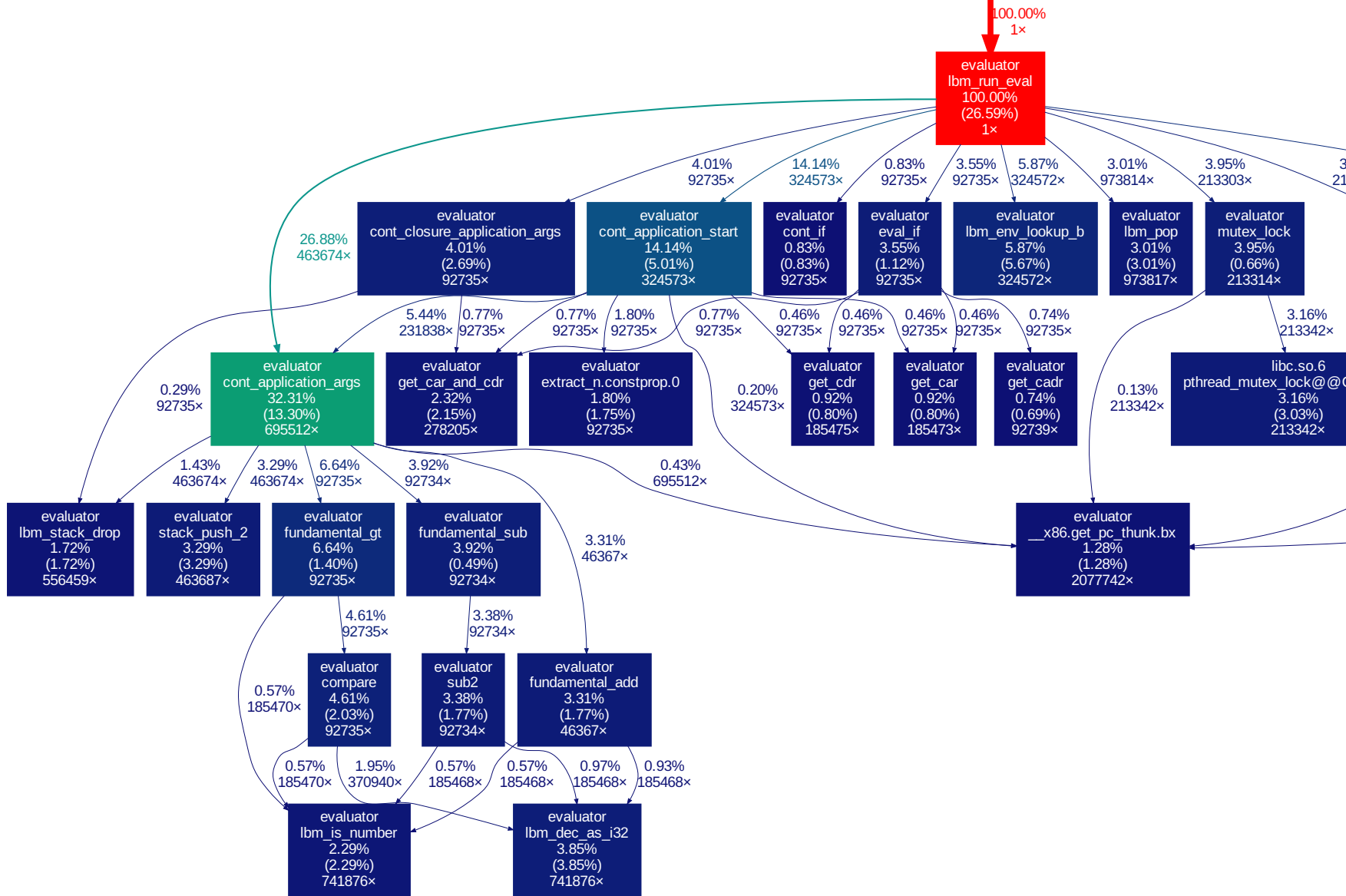
- 2.3Mnfib/s (32bit binary on I7-10700)
- 3.1Mnfib/s (64bit binary on I7-10700)

Nfib

- 2.3Mnfib/s (32bit binary on I7-10700)
- 3.1Mnfib/s (64bit binary on I7-10700)

- ~8Mnfib/s (Lennart's combinators on M1)
- ~10Mnfib/s (Lennart's combinators on M1 when we talked to him again moments later)

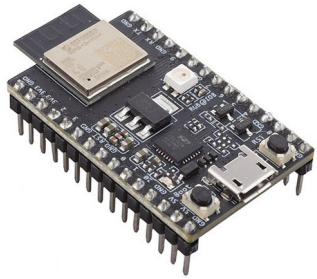




TODO

- Look over application and all those special cases.
- Compilation of some kind?
- Maybe some MicroHS inspiration for the GC?
 - The bitmap, the lazy sweep...

To try it out



Buy: esp32c3-devkitm-mini-1 (< 10\$€€)

Download: https://vesc-project.com/vesc_tool

