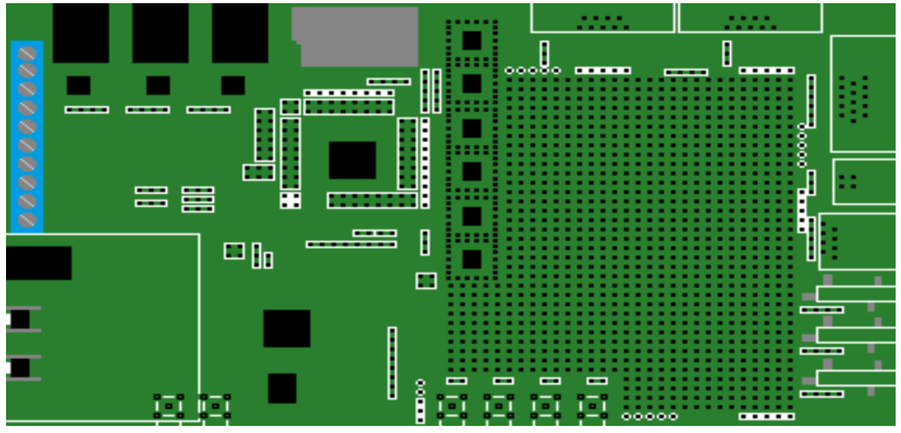
EVB001 Studio



by Stefan Mauerhofer

Goals

- An environment for the C-compiler
- Better visual representation
- Better understanding of the EVB001 and GA144
- Portable C++ code running under Windows and Linux
- No third-part royalities by using open-source libraries (Boost, Gtkmm, ...)

Drawbacks

- Lack of the interactivity enjoyed with Forth
- Fixed for EVB001 and GA144
- No command line interface

Connceting to the EVB001

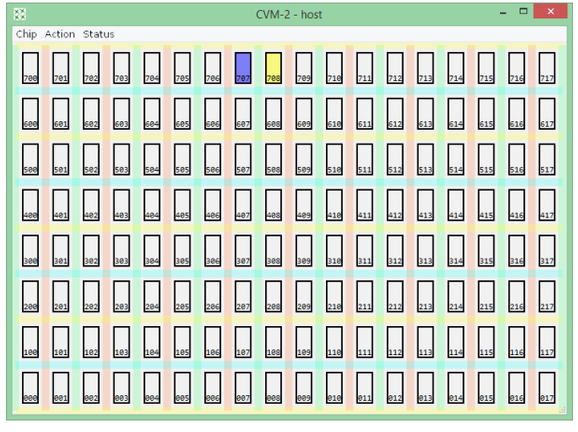
- Connecting via USB FTDI serial lines
- Serial parameter like in AF (arrayForth)
- Boost-asio and multi-threaded programming

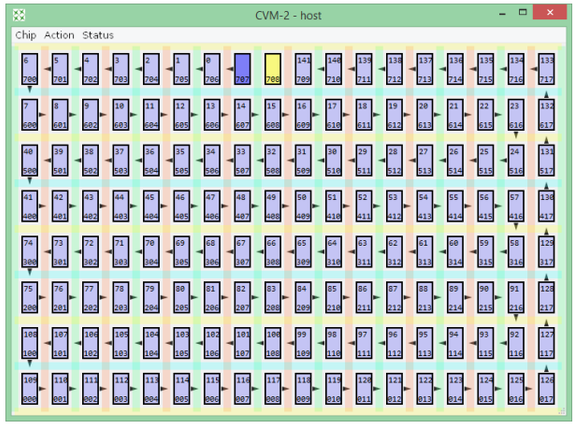
Reseting the host chip

- Set jumper 1-3 on J20
- Open serial line to serial port A
- Setting RTS to low
- Waiting a few milliseconds
- Setting RTS to high

Booting the host chip

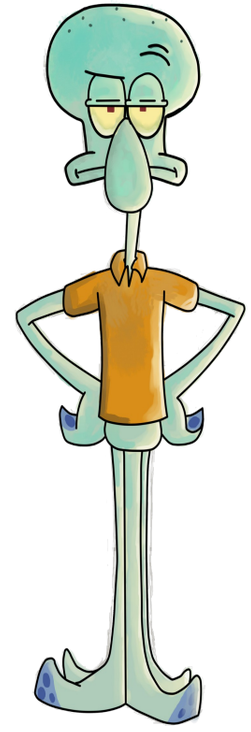
- Open serial line & reset host chip
- Send boot frame to node 708 via port A
- Send boot data to node 707 to install Kraken in this node
- Span a tentacle across the whole chip
- After success the whole chip is now under control





What is Kraken and Tentacle?

- Kraken is a generator for sequences controlling other nodes via port access only
- Tentacle is a path of nodes
- A Kraken can access all the nodes in its tentacle
- A node in the tentacle has 2 constraints:
  - P is set to the port to its predecessor
  - B is set to the port to its successor



Port controlling sequences

- Based on induction mathematics
- Starting condition
- Rule from state n to state n+1

Basic definitions

- a = A[@p >r]]
- b = A[begin @p !b unext]]
- c = A[begin @b !p unext]]
- d = A[@b !p]]
- p(x) = a x b
- q(x) = a x c

Write 1 word transaction

- w1(0, x) = x
- w1(n+1, x) = p(3n) w1(n, x)
- Seqence length = 3n+1
- Example:
  - w1(0, x) = x
  - w1(1, x) = p(0) w(0, x) = a 0 b x
  - w1(2, x) = p(3) w(1, x) = a 3 b a 0 b x
  - w1(3, x) = p(6) w(2, x) = a 6 b a 3 b a 0 b x

Write 2 word transaction

- w2(0, x, y) = x y
- w2(n+1, x, y) = p(3n+1) w1(n, x, y)
- Seqence length = 3n+2
- Example:
  - w2(0, x, y) = x y
  - w2(1, x, y) = p(1) w(0, x, y) = a 1 b x y
  - w2(2, x, y) = p(4) w(1, x, y) = a 4 b a 1 b x y
  - w2(3, x, y) = p(7) w(2, x, y) = a 7 b a 4 b a 1 b x y

Write transaction

- w(0, m-1, x{m}) = x{m} = x[0] x[1] ... x[m-1]
- w(n+1, m-1 x{m}) = p(3n+m-1) w(n, m-1, x{m})
- Seqence length = 3n+m
- Example:
  - w(0, 2, x{3}) = x[0] x[1] x[2] = x0 x1 x2
  - w(1, 2, x{3}) = p(2) w(0, 2, x{3}) = a 2 b x0 x1 x2
  - w(2, 2, x{3}) = p(5) w(1, 2, x{3}) = a 5 b a 2 b x0 x1 x2
  - w(3, 2, x{3}) = p(8) w(2, 2, x{3}) = a 8 b a 5 b a 2 b x0 x1 x2

Write & read 1 word transaction

- wr1(0, x) = x -> y
- wr1(n+1, x) = p(4n) wr1(n, x) q(k-1)
- Seqence length (write & read) = 4n+2
- Example:
  - wr1(0, x) = x -> y
  - wr1(1, x) = p(0) wr1(0, x) d = a 0 b x d -> y
  - wr1(2, x) = p(4) wr1(1, x) d = a 4 b a 0 b x d d -> y
  - wr1(3, x) = p(8) wr1(2, x) d = a 8 b a 4 b a 0 b x d d d -> y

Write & read transaction

- wr(0, m-1, x{m}, k-1) = x{m} -> y{k}
- wr(n+1, m-1 x{m}, k-1)
  = p(6n+m-1) wr(n, m-1, x{m}, k-1) q(k-1)
- Seqence length (write & read) = 6n+m+k
- Example:
  - wr(0, 0, x{1}, 1) = x[0] -> y[0] y[1] = x0 -> y0 y1
  - wr(1, 0, x{1}, 1) = p(0) wr(0, 0, x{1}, 0) q(0)
    = a 0 b x0 a 1 c -> y0 y1
  - wr(2, 0, x{1}, 1) = p(6) wr(1, 0, x{1}, 0) q(0)
    = a 6 b a 0 b x0 a 1 c a 1 c -> y0 y1

Extending the tentacle

- let t[n] be tentacles last node
- set B in t[n] to next node t[n+1]
- send a focusing call or jump to t[n+1]
- make t[n+1] tantacle's last node

Setup a node

- A = 0 (1)
- write memory x 64 (65)
- write A (1)
- write B (1)
- write return stack (9) (15)
- write parameter stack (13)
- jump to start (1)
- 3n + 97

Booting the host & target chip

- Boot via port A and node 708 and reach out to node 300
- Reset target chip through 500.17 (low then high)
- Install boot frame code in node 300
- Send boot frame with bridge code to 10300
- Install bridge code in 300
- Continue spreading tentacle into target chip

Resetting the target chip

- Reset target chip through 500.17 (low then high)
- C++ Code:

```
bool Manager::reset_target()
{
  size_t segment;
  if (!activate_node(500, segment)) {return false;}
  long data[4];
  data[0] = ops_[RemoteOp::write_a];
  data[1] = F18A_IO;
  data[2] = ops_[RemoteOp::store_a];
  data[3] = F18A_PIN17_LO;
  if (!w(segment, 4, data)) {return false;}
  connection::sleep(50);
  if (!w2(segment, ops_[RemoteOp::store_a], F18A_PIN17_HI)) {return false;}
  connection::sleep(20);
  return true;
}
```

Boot code for node 300

- Boot code:

```
# 0 org
: dly ( b)  !b  40. for unext ;  ( ~88 ns)
: 1bt ( b)  dup dly  x10000. xor dly ;
: 18o ( w-0)  x30000. dly  8. for begin
   -while  x30003. 1bt  : rise
      2* -if  x20003. 1bt  2* *next drop ;
      then  x20002. 1bt  2* *next drop ;
   then  x30002. 1bt  rise ;
: off  io b!  x20002. dly  x10001. !b ;
: trns up a! @ for @ 18o next off await ;
```

Write boot code to node 10300

- C++ code:

```
long data[F18A_RAM_SIZE+5];

// write ram of node 300
pos = 0;
data[pos++] = ops_[RemoteOp::set_a_to_0];
data[pos++] = ops_[RemoteOp::write_rstack];
data[pos++] = F18A_RAM_SIZE-1;
data[pos++] = ops_[RemoteOp::write_mem_loop];
for (size_t i=0; i<F18A_RAM_SIZE; ++i) {
  data[pos++] = boot_module.ram().data(i);
}
if (!src_proc->find_label("trns", 1, addr)) {
  event_list_.push_back("Manager::build_sync_bridge : cannot find label 'trns' in node "+ boost::lexical_cast<std::string>(host_node));
  return false;
}
data[pos++] = f18a::State::addr_to_jump(addr);
if (!w(segment, pos, data)) {
  return false;
}
```

Bridge code for node 300 & 10300

- Bridge code:

```
# 0 org
: ?lo  begin @b inv -until ;
: dly ( b)  40. for unext !b ;  ( ~88 ns)
: 1bt ( b)  dup dly  x10000. xor dly ;
: zro  x10001. dly ;
: wpd  x10001. !b ;
: 18o ( w - 0)
   17. for begin
   -while x30003. 1bt 2* next  drop wpd ;
   then   x30002. 1bt 2* next  drop wpd ;
: 18i  ( HiZ) dup xor !b
   17. for  begin @b -until
   begin @b inv -until  inv 2. and 2/  a 2* xor a!  next
   a up a! !  ( ack) x30001. 1bt wpd
: idl  # --lu alit a! ..  @ @b -if  ( wire) drop 18i ;
   then  ( port) drop 18o  begin @b -until  ?lo idl ;
: ent  io b!  wpd  ?lo idl ;
```

Write boot frame to node 10300 via 300

- C++ code:

```
pos = 1;
// send boot frame to node 10300 via node 300
// boot frame conatins bridge code for node 10300
if (!dest_proc->find_label("ent", 0, addr)) {
  event_list_.push_back("Manager::build_sync_bridge : cannot find label 'ent' in node "+ boost::lexical_cast<std::string>(target_node));
  return false;
}
data[pos++] = addr;  // start address of node
data[pos++] = 0;  // address to write
data[pos++] = F18A_RAM_SIZE; // size of memory block to write
for (size_t i=0; i<F18A_RAM_SIZE; ++i) {
  data[pos++] = dest_module.ram().data(i);
}
data[0] = pos-2; // size of data transfered to target node (minus 1)
if (!w(segment, pos, data)) {
  return false;
}
```

Write bridge code to node 300

- C++ code:

```
// install bridge code in node 300
pos = 0;
data[pos++] = ops_[RemoteOp::set_a_to_0];
data[pos++] = ops_[RemoteOp::write_rstack];
data[pos++] = F18A_RAM_SIZE-1;
data[pos++] = ops_[RemoteOp::write_mem_loop];
for (size_t i=0; i<F18A_RAM_SIZE; ++i) {
  data[pos++] = src_module.ram().data(i);
}
if (!src_proc->find_label("ent", 0, addr)) {
  event_list_.push_back("Manager::build_sync_bridge : cannot find label 'ent' in node "+ boost::lexical_cast<std::string>(host_node));
  return false;
}
data[pos++] = f18a::State::addr_to_jump(addr);
if (!w(segment, pos, data)) {
  return false;
}
```

Example with LED

- Digital output (10517.17)
- Analog output (10117.ao)
- Duty cycle

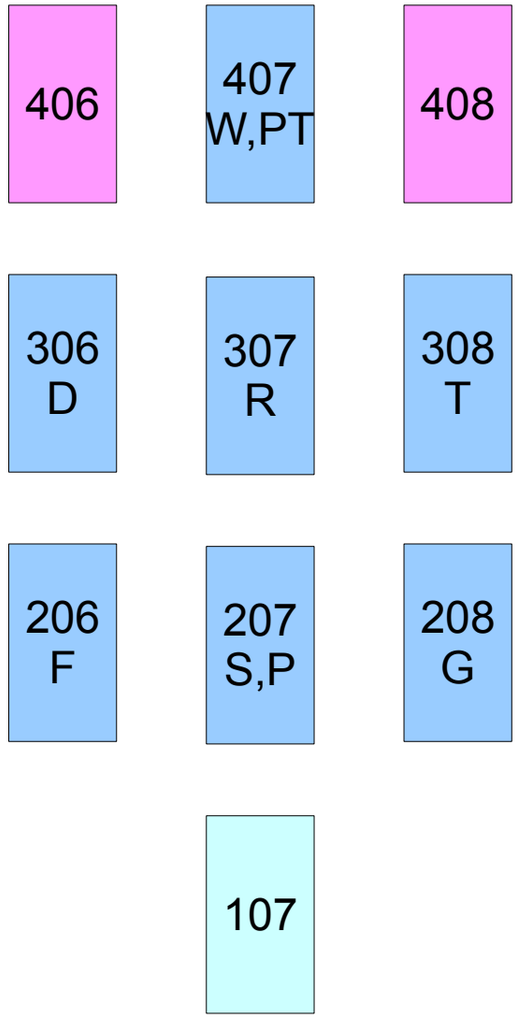
C-compiler

- C virtual machine (CVM) has 2 parents:
  - Charley Shattuck
  - Stefan Mauerhofer
- GA144 node design
- CVM instruction set
- C compiler written in C++ (using Boost Spirit)

CVM

- Used nodes:
  - 207 program control, memory and stack interface (P, S)
  - 206 frame management (F)
  - 208 global variables (G)
  - 307 binary arithmetic and control extension (R)
  - 306 extended arithmetic (D)
  - 308 comparison and unary arithmetic (T)
  - 407 interface to GA144 chip (W, PT)

CVM node layout



CVM-2 register

Abbr	Description	Node	Location
S	stack pointer	207	top of stack
P	program counter	207	register A
F	frame pointer	206	register A
R	result register	307	register A
D	double register	306	register A
T	temporary register	308	register A
G	global base register	208	register A
W	word register	407	top of stack
PT	port register	407	register A

CVM instruction set

Pattern	Min	Max	Description
`0xxx xxxx xxxx xxxx`	`0`	`0x7fff`	call
`1001 0xxx xxxx xxxx`	`-0x400`	`0x3ff`	branch (signed offset)
`1001 1xxx xxxx xxxx`	`-0x400`	`0x3ff`	conditional branch (signed offset)
`1010 1000 xxxx xxxx`	`0`	`0xff`	enter stack frame
`1010 1001 xxxx xxxx`	`0`	`0xff`	cleanup stack frame
`1010 1010 xxxx xxxx`	`0`	`0xff`	store local
`1010 1011 xxxx xxxx`	`0`	`0xff`	store parameter
`1010 1100 xxxx xxxx`	`0`	`0xff`	load local
`1010 1101 xxxx xxxx`	`0`	`0xff`	load parameter
`1010 1110 xxxx xxxx`	`0`	`0xff`	address of local
`1010 1111 xxxx xxxx`	`0`	`0xff`	address of parameter
`1011 100x xxxx xxxx`	`0`	`0x1ff`	offset to global variable
`1011 101x xxxx xxxx`	`0`	`0x1ff`	mem[r = r+off] = pop(). Note that the register r contains the address of the memory location just used.
`1011 110x xxxx xxxx`	`0`	`0x1ff`	r = mem[r+off]
`1011 111x xxxx xxxx`	`0`	`0x1ff`	offset to address. r = r+off
`1101 xxxx xxxx xxxx`	`-0x800`	`0x7ff`	signed literal

CVM instructions 8xxx

Opcode	Mnemonic	Description
`800e n`	`plit`	Push literal n onto the stack
`8014 n`	`llit`	Move literal n to R
`8018`	`pop`	Pop R from the stack
`801a`	`push`	Push R onto the stack
`802e`	`ret`	Return from call (without changing the frame pointer)
`8030`	`xs`	Exchange stack pointer with R
`8032`	`xp`	Exchange program counter with R
`8034`	`tjmp`	Table jump. Add R to P and add R again with the value at that location
`8037`	`pc`	Move program counter to R

CVM instructions Axxx

Opcode	Mnemonic	Description
a033	xf	Exchange frame pointer with r
a038	term	Set P to 1 and wait for a stimulus
a03c	wait	Wait for a stimulus
a8xx	enter xx	Build a stack frame with xx locals
a9xx	exit xx	Exit stack frame, restore the frame pointer and return from call. then cleanup stack by adding xx to stack pointer
aaxx	stl xx	Move R to local xx
abxx	stp xx	Move R to parameter xx-2
acxx	ldl xx	Move local xx to R
adxx	ldp xx	Move parameter xx-2 to R
aexx	lal xx	Move address of local xx to R
afxx	lap xx	Move address of parameter xx-2 to R

CVM instructions Bxxx

Opcode	Mnemonic	Description
b008	ld	Load R with cell at address in R
b00c	st	Store top of stack at address in R
b010	ldph	Push cell at address in R onto stack
b029	xg	Exchange global pointer with R
b02e	ldx	Load R with cell at extended address in pop(), R. pop 1 cell
b035	stx	Store top of stack at extended address in second of stack, R. pop 2 cells and leave R unchanged
b03d	mk	Detach (mask) memory controller (used for break. Stops virtual machine)
b800-b9ff	glob xxx	Move address of global xxx to R
ba00-bbff	sto xxx	Store top of stack at address in R = R + xxx
bc00-bdff	ldo xxx	Load R with cell at address in R + xxx
be00-bfff	la xxx	Load address. R = R + xxx

CVM instructions Cxxx

Opcode	Mnemonic	Description
c82d	sl	Shift left top of stack R+1 times
c82f	sr	Signed shift right top of stack R+1 times
c831	usr	Unsigned shift right top of stack R+1 times
c833	add	Add R and top of stack. R = R + pop()
c834	and	Bitwise and of R and top of stack. R = R & pop()
c835	xor	Bitwise exclusive or of R and top of stack. R = R ^ pop()
c836	or	Bitwise or of R and top of stack. R = R | pop()
c838	cx	Atomic compare and exchange.

CVM instructions Exxx

Opcode	Mnemonic	Description
e00f	zext	Set D to 0
e013	addc	Add R, top of stack and carry (in D). Result in R and carry in D
e01d	ldd	Move D to R
e01e	std	Move R to D
e020	xd	Exchange D with R
e022	lshc	16 bit shift left of R with carry (in D). Result in R and carry in D
e026	rshc	16 bit shift right of R with carry (in D). Result in R and carry in D
e02b	sext	Extend the sign in R into D
e030	umul	16x16 bit unsigned multiplication of R and top of stack. Hi result in R, lo result in D

CVM instructions Fxxx

Opcode	Mnemonic	Description
f006	sub	Subtract R from top of stack. R = pop() - R
f00a	inc	Increment R by 1
f00c	dec	Decrement R by 1
f00d	eq	equal (pop() == R)
f00e	eq0	Equal to 0 (or logical not) (set R to 1 if R was 0 otherwise set R to 0)
f010	false	Set R = 0
f011	true	Set R = 1
f013	ne	Not equal (pop() != R)
f014	ne0	Not equal to 0 (R != 0) (set R to 1 if R was not 0 otherwise set R to 0)
f017	ugt	Unsigned greater than (pop() > R)
f01b	gt	Greater than (pop() > R)
f01c	gt0	Greater than 0 (R > 0)
f01d	ge	Greater or equal than (pop() >= R)
f01e	ge0	Greater or equal than 0 (R >= 0)

CVM instructions Fxxx cont.

Opcode	Mnemonic	Description
f01f	ule	Unsigned lower or equal than (pop() <= R)
f023	le	Lower or equal than (pop() <= R)
f024	le0	Lower or equal than 0 (R <= 0)
f025	lt	Lower than (pop() < R)
f026	lt0	Lower than 0 (R < 0)
f027	ult	Unsigned lower than (pop() < R)
f029	uge	Unsigned greater or equal than (pop() >= R)
f02b	mul2	Shift left R by 1
f02c	udiv2	Unsigned shift right R by 1
f02d	div2	Arithmetic shift right R by 1
f02f	abs	Absolute value of R in R
f031	neg	Negate R
f032	nop	No operation (do nothing)
f033	not	Bitwise invert of R

CVM instructions Fxxx cont.

Opcode	Mnemonic	Description
f01f	ule	Unsigned lower or equal than (pop() <= R)
f023	le	Lower or equal than (pop() <= R)
f024	le0	Lower or equal than 0 (R <= 0)
f025	lt	Lower than (pop() < R)
f026	lt0	Lower than 0 (R < 0)
f027	ult	Unsigned lower than (pop() < R)
f029	uge	Unsigned greater or equal than (pop() >= R)
f02b	mul2	Shift left R by 1
f02c	udiv2	Unsigned shift right R by 1
f02d	div2	Arithmetic shift right R by 1
f02f	abs	Absolute value of R in R
f031	neg	Negate R
f032	nop	No operation (do nothing)
f033	not	Bitwise invert of R

CVM remaining instructions

Opcode	Mnemonic	Description
`f034`	`xt`	Exchange temp register T with R
`f035`	`ldt`	Move T to R
`f036`	`stt`	Move R to T
`f037`	`bcnt`	Count 1 bits in R
`0000-7fff`	`call xxxx`	Call function at address xxxx and save return address on the stack
`9000-97ff`	`br xxx`	Unconditional branch with offset xxx
`9800-9fff`	`cbr xxx`	Conditional branch (if R == 0) with offset xxx
`dxxx`	`lit xxx`	Set R to the signed literal xxx

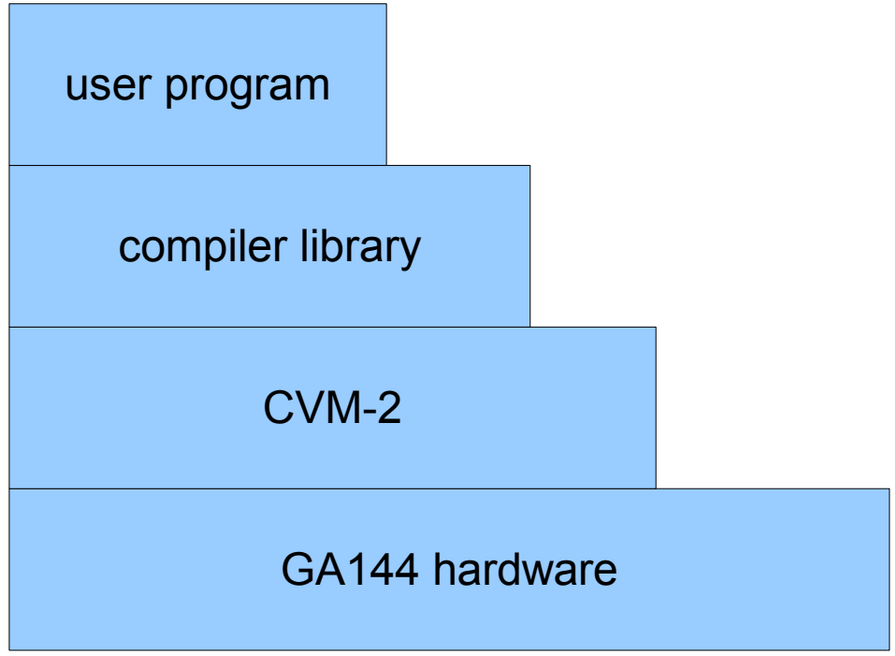
CVM stack frame

- Parameter are put on the stack from left to right
- Variadic arguments are not allowed (e.g. printf)
- The place of a result depends on its size
  - 1: register R
  - 2: register D, R (hi, lo)
  - >2: parameter on the stack (before first parameter)

CVM stack frame

Offset to F	Content	Assembler
n+x+2	result (opt)	ldp n+x
...	...	...
n+3	result (opt)	ldp n+1
n+2	parameter 0	ldp n
n+1	parameter 1	ldp n-1
...	...	...
3	parameter n-1	ldp 1
2	parameter n	ldp 0
1	return address	ldp -1
0 (F ->)	old F	ldp -2
-1	local 0	ldl 0
-2	local 1	ldl 1
...	...	...
-m	local m	ldl m

C environment architecture



C compiler library

- Not all C primitives are implemented in the CVM
- A CVM assembler is integrated into the CVM
- Missing functions implemented in CVM assembler + C

C system component example

```
__system long load_long(void* src)
{
  __asm(
    "cvm2",
    "ldp 0; ld; xd;"
    "ldp 0; inc; ld;"
  );
}

__system void store_long(long src, void* dest)
{
  __asm(
    "cvm2",
    "ldp 1; push; ldp 0; st;"
    "ldp 2; push; ldp 0; inc; st;"
  );
}
```

Studio TODO list

- Complete C compiler
- Integrate C compiler & editor
- Access to EVB001 SRAM & flash
- Bootstream generator
- Program loader