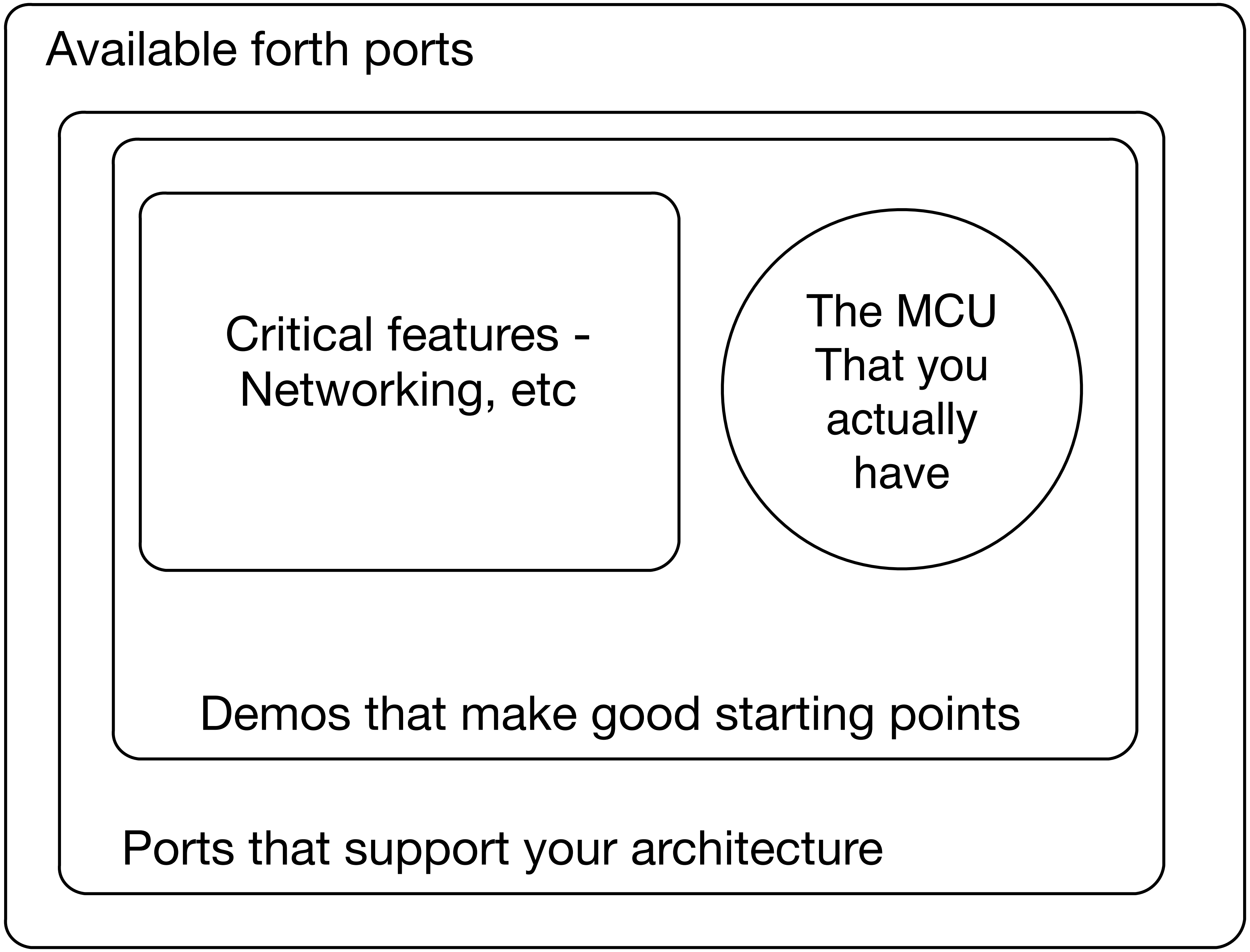


Coin cell Forth with SockPuppet and MPE

The core problem



Sockpuppet 1/2

- 2011: I look around for an open source forth for Cortex-M
- Initial port (2012) was Riscy-pygness forth to the Stellaris Cortex-M chips. It was primitive and missing lots of stuff. MPE was available for the Mac.
- Clock configuration sucked.
- Cortex-M system call layer is pretty clean - How about a simple system call layer?
- 2016 - Integrated into MPE Forth.   Becoming generic-ised.

Sockpuppet 2/2

- Since the original implementation, simpler approaches
  - Launcher-based implementations for Cortex-Ms.
  - Shared memory/interconnect - first byte of RAM is architecturally at 0x2000:0000
- More powerful

Since 2012

- Minimalist ports - launcher based
- No more bit-banding - switch to Cortex-M atomic operations.
- Get rid of SVC call wrappers for C functions.
- MPU support for forth threads.
- Scheduler improvements for MPE forth.
- Ports to three different product families

Arm Cortex-M0/3/4/7

- 32-Bit Architecture
- Low-latency Prioritized Interrupt controller - NVIC
- Sophisticated Debug
- M0: Simplified and Ultra-low power.
- M4: DSP and Floating point
- M7: Higher performance, better power efficiency.

What the C-M is good at

- Interrupt/Exception/Fault handlers
    - Low-Latency with prioritized handling
    - The hardware does all the stacking and gives you a ready-to-go execution environment (if you are written in C!)
- Supervisor/User Separation
    - Multiple system stacks
    - Memory Protection unit

Forth Challenges

- Interrupt handlers
  - Handlers have to setup a Forth environment
    - Data stack
    - User area
- Machine initialization - Tedious and easy to get wrong.
- Somebody else already did the work

Bootstrapping Forth

- Forth is an excellent rapid-prototyping and debugging environment
    - Direct access to device memory
- Forth is a dangerous rapid-prototyping and debugging environment
    - Direct access to device memory
    - Expect a lot of crashes due to typos

MPE Forth

- Compiler-based
- Supported on on Linux, Mac, and windows
- Good documentation.
- Hobbyist compatible licensing.

Foundations:AAPCS 1/2

- Well-defined by ARM: IHI0042F_aapcs.pdf
- Registers R0-R3 for callee-parameters, R0-1 for return values. Additional args go on stack - usually not needed.
- R12 is a inter-procedure call scratch register - must be preserved.
- Interrupt architecture is compatible with the ABI.

Foundations: AAPCS 2/2

```
CODE CALL1--N ( addr arg0 -- n )
  mov r0, tos
  ldr tos, [ psp ], # 4
  orr tos, tos, # 1 \ set Thumb bit
  push { psp, link }
  blx tos
  pop  { psp, link }
  mov tos, r0
  next,
END-CODE
```

R0
R1
R2
R3
R12
LR
PC
xPSR

https://github.com/rbsexton/cm3forthtools/blob/master/aapcs.fth

Forth SVC Calls

```
\ *************************************************
\ SVC 0: Return the version of the API in use.
\ *************************************************
CODE API-Version  ( -- n )
   svc #0                           ( Call Supervisor)
   str tos, [ psp, # -4 ] ! ( Push TOS)
   mov tos, r0                      ( return value)
   next,
END-CODE
```

Note: Cortex-M3 Pushes R0-R4, R12, LR, PC, xPSR automatically at SVC entry. MPE Forth can generate this code automatically

https://github.com/rbsexton/sockpuppet/blob/master/forth/SysCalls.fth

Porting to the Gecko1/5

- Silabs Tiny Gecko - A Cortex-M3 device.
- Start with the UART Equivalent of hello_world()
- Forth UART Drivers without hardware init.
- Partitioning

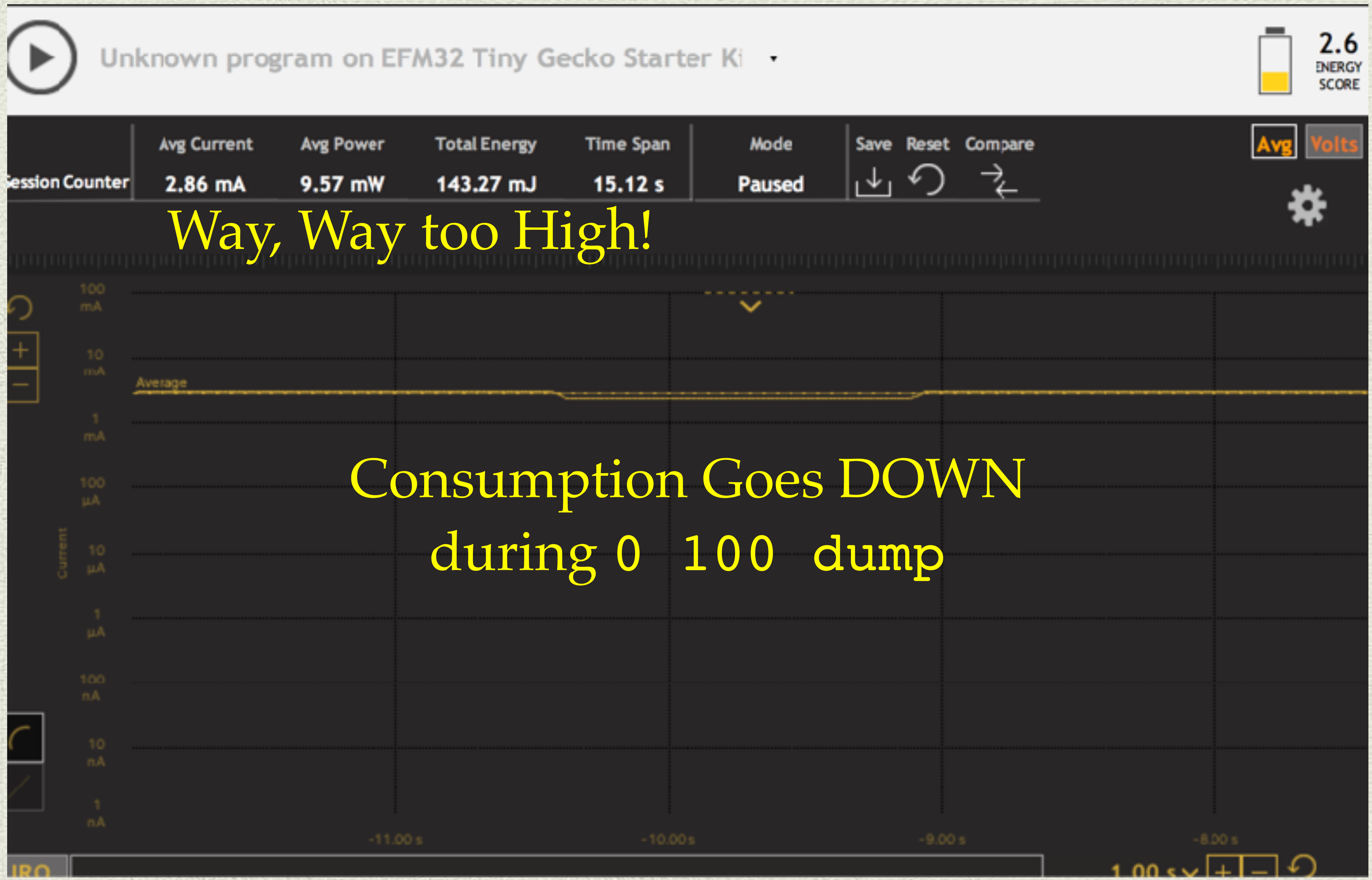
https://github.com/rbsexton/gecko/

Minimal Port #1

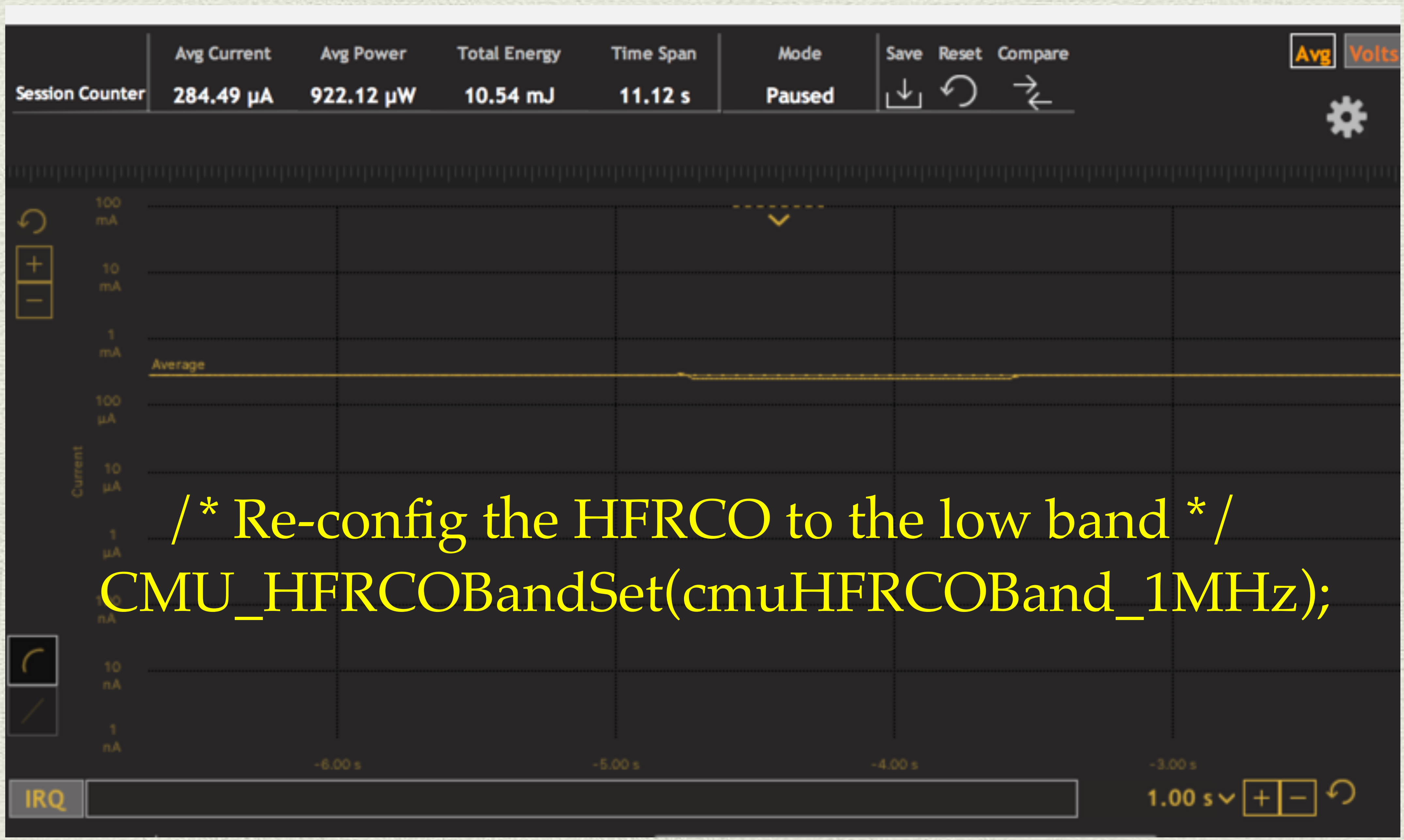
- Launcher Initializes the hardware.
- Single-Threaded forth polls the UART status register.

```
$8 equ LEUART_STATUS
bit4 equ LEUART_STATUS_TXBL

$28 equ LEUART_TXDATA

internal

: (seremit) \ char base --
\ *G Transmit a character on the given UART.
  begin
    dup LEUART_STATUS + @ LEUART_STATUS_TXBL and \ Tx FIFO full test
  until
  LEUART_TXDATA + !
;
```

Minimal #1a - 2.6maA!



Minimal #1b- 280uA



Adding WFI

- Basic power control - Forth executes WFI to stall the CPU while waiting for an event.
- The most basic wake event is a UART character
- Requires a shared data between between the supervisor and forth

Shared Data 1/3

- The tricky part is passing the address of the shared structure.
  - Option One - Use the Sockpuppet API to retrieve it.
  - Option Two - Pass it over to forth at startup time.

```
// Let Forth set its own stack pointer.
  LaunchUserAppNoSP( (long unsigned int *) 0x2000, (uint32_t *) &theshareddata);
```

```
.global LaunchUserAppNoSP

LaunchUserAppNoSP:
        ldr r2, [ r0, #4 ] /* The initial PC */
        mov r0, r1
    bx  r2
```

Shared Data 2/3

- Forth must catch it and save it for later.

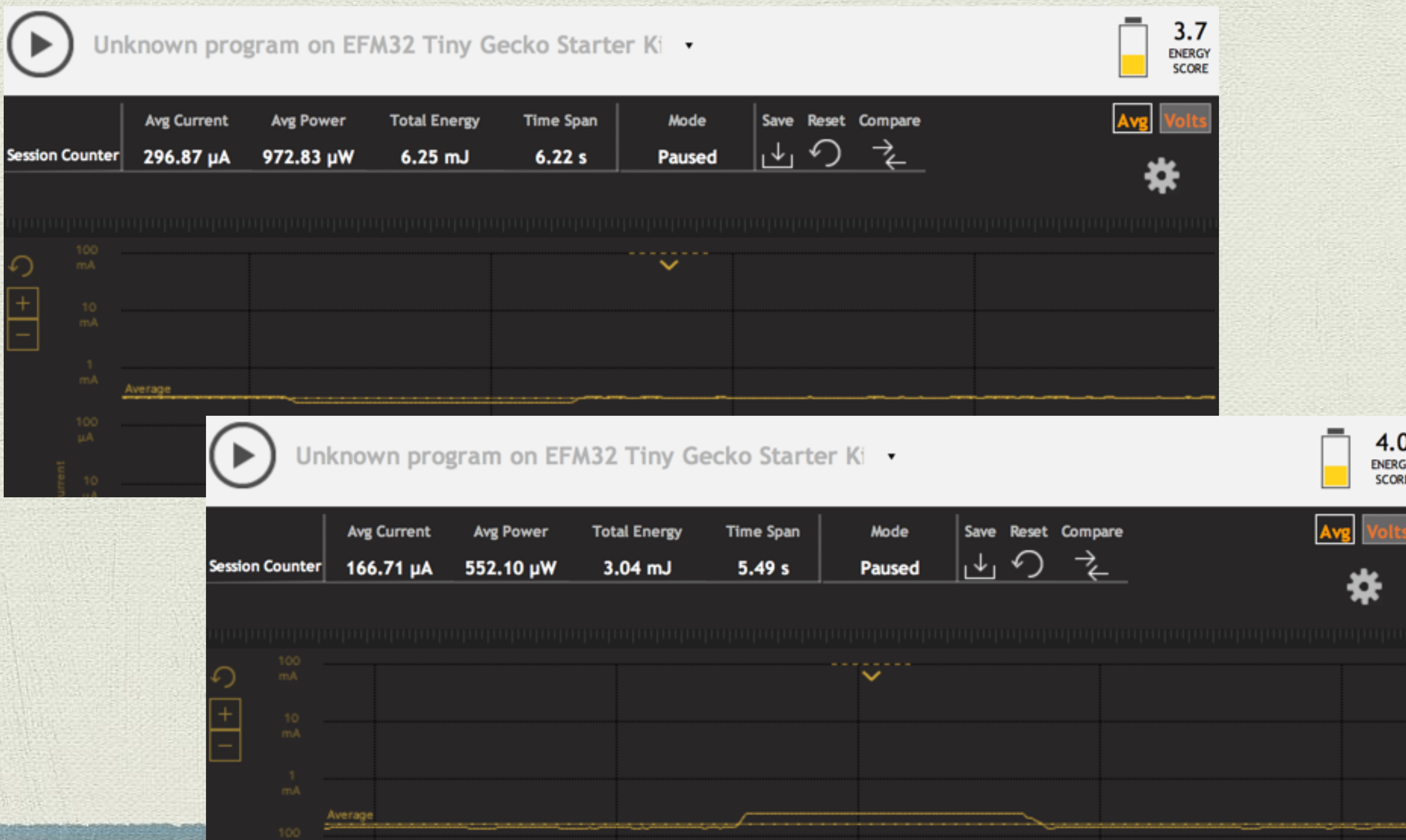
```
udata \ This has got to be part of udata
create icroot 4 allot \ Values are cleaner, but they're part of IDATA..
cdata

code get_icroot
    str tos, [ psp, # -4 ] !
    mov r7, r0
    next,
end-code

: StartCortex    \ -- ; never exits
    INIT-R0 SP_process sys!  2 control sys! \ switch to SP_process
    REAL-INIT-S0 set-sp            \ Allow for cached TOS and guard space
    get_icroot \ Do this before anything else tampers with R0.
    icroot !
    INIT-U0 up!  CLD1 @ execute
  again
;
```

Shared Data 3/3

- Forth must catch it and save it for later.

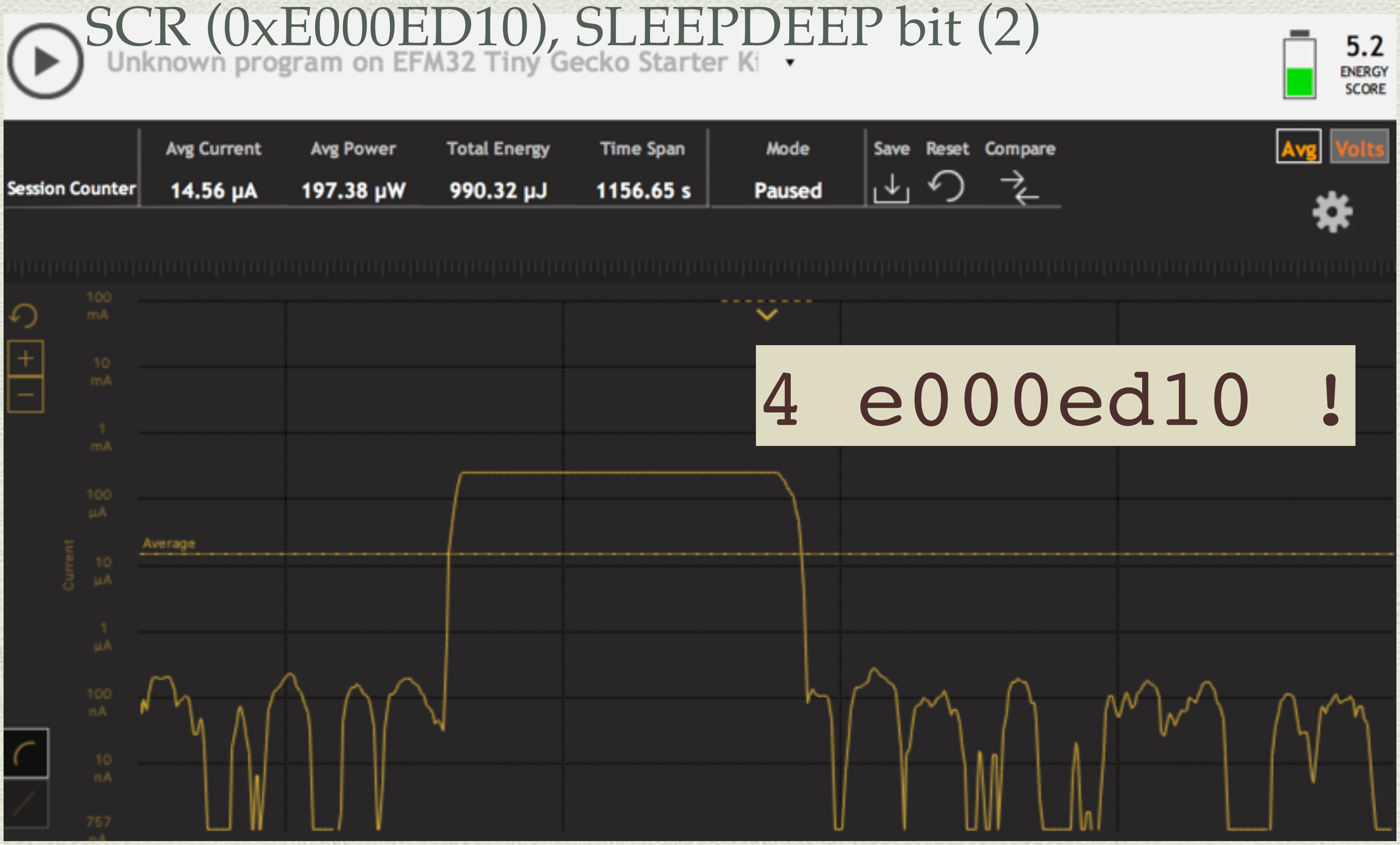
```
: (serkey?) \ -- t/f
\ *G Return true if the IRQ handler has dropped off a payload.
   icroot @ u0rxdata @ -1 <> \ Rx
;

: (serkey)  \ -- char
\ *G Wait for a character to come available on the given UART and
\ ** return the character.
  begin
   (serkey?)
   dup false = if [ tasking? ] [if]  pause  [else] [asm wfi asm] [then] then
  until
  di
  icroot @ u0rxdata dup
   c@ swap -1 swap !  \ Fetch the result, then reset it.
  ei
;
```

WFI Improvements 1/2



WFI Improvements 2/2

- This part lets us enter ultra-low power EM2 with Cortex-M SCR (0xE000ED10), SLEEPDEEP bit (2)



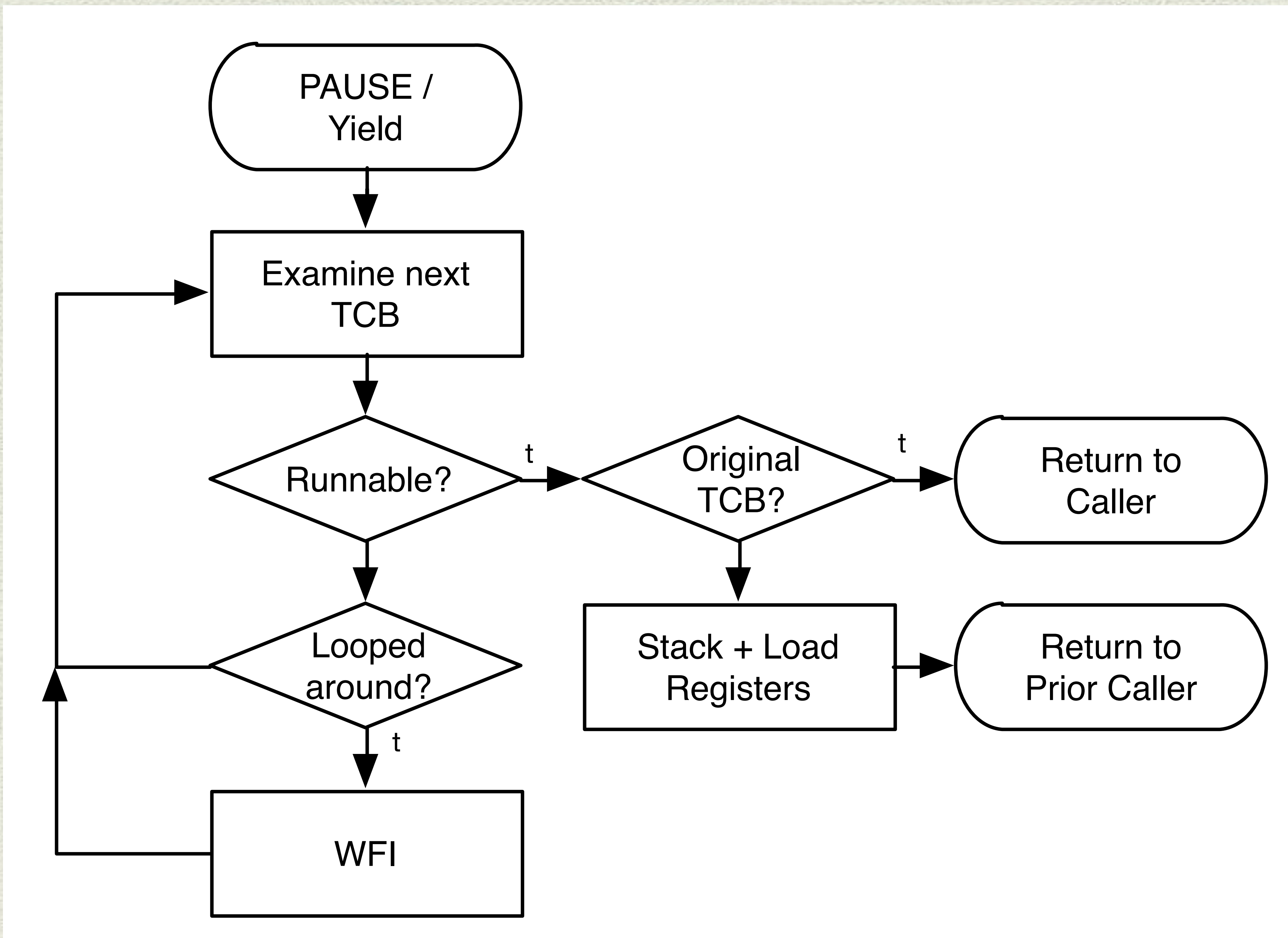
Pitfalls

- Your programmer might not erase/init as you expect - I program the forth section first, then the launcher.
- Consider writing a script to stitch the halves together.
- Working on low level IO bringup is tedious - work in the smallest possible steps.

Now What? Tasks!

- Modern embedded systems are event and interrupt driven.
- When nothing is runnable, you can enter a low-power state.
- Blocking code is simpler - no need to keep state.
- Forth is well-suited to tasking.
  - Saving context takes only 4 registers, much less than RTOS
  - Task are normal words that call `PAUSE`

High Efficiency Tasking



A Scheduler with WFI

```
CODE pause   \ -- ;
  mov r6, up   \ Use r6 as the working copy
l: [schedule]next
  ldr     r6, [ r6, # 0 tcb.link ]\ get next task
  ldr     r5, [ r6, # 0 tcb.status ]  \ inspect status
  cmp     r5, # 0        \ 0 = not running
  b .ne   [schedule]run
  cmp     r6, up \ No work?  WFI
  it .eq
    wfi
  b  [schedule]next
l: [schedule]run
  cmp r6, up \ If we've come back to ourselves, just return.
  it .eq
    bx lr
  push    { r7, r9, r12, link }       \ stack registers
  str     rsp, [ up, # 0 tcb.ssp ]\ save SP in TCB
  mov up, r6 \ Load up the new task pointer.
  \ run selected task - sp, up, rp, ip
  ldr     rsp, [ up, # 0 tcb.ssp ]\ restore SSP
  pop     { r7, r9, r12, link }       \ restore registers
```

https://github.com/rbsexton/cm3forthtools/blob/master/pause.fth

Disaster!

The thread that calls PAUSE never sleeps!
We need a way to wake it up

Problem: When can the multi-tasker safely call WFI?
Answer: When there are no running tasks

SockPuppet Integration

- Thumb-2 provides very clean SVC interface
- This is a binary ABI.   Develop on one platform, run on another (may require memory map adjustments)
- Conceptually, it resembles a traditional BIOS.

Integrating SockPuppet

```
SVC_Handler:
   tstlr,#0x4      @ Figure out which stack
   iteeq
   mrseq r0,msp    @ Main stack
   mrsne r0,psp    @ Process/Thread Stack
   push { r4, lr }
   mov r4, r0 @ We'll over-write R0, so stash it in r4.
   ldr  r1, [r0,#24]  @ Get the stacked PC
   ldrb r1, [r1,#-2]  @ Extract the svc call number
   ldr  r2,=syscall_table
   ldr  r12, [r2, r1, LSL #2]
   ldm r4, { r0-r3 } @ Pull function args from the stack.
   blx r12
   stm r4, { r0-r1 } @ Support 64-bit return values.
   pop { r4, pc }
```

https://github.com/rbsexton/sockpuppet/blob/master/sapi/svchandler.S

System Call Handlers

The system call handler must stop the task - no other safe way

```
/// @parameters
/// @R0 - Stream Number
/// @R1 - The Character in question.
/// @returns in R0 - Result - 0 for success.
/// @ 1 for blocked - Thread must yield/pause
bool __SAPI_02_PutChar(int stream, uint8_t c, unsigned long *tcb){
   int ret;
   switch ( stream ) {
      default:
         return(console_leuart_putchar(c, tcb));
      }
   return(ret);
   }
```

You only need three

- SAPI Defines 16 reserved vectors. You only need three to get to a working system
  - GetChar - KEY
  - CharsAvail - KEY?
  - Putchar - EMIT
- CR and TYPE are also defined, but can be emulated with forth code. High-performance systems can benefit from implementing TYPE and CR

Driver needs state

```
typedef struct {
   unsigned long *tcb;
   bool blocked_tx;
   bool blocked_rx;
   } sIOBlockingData;

    int free = ringbuffer_addchar(&rb_tx,c);
    // If maxing out, tell the caller to yield.
    if ( free == 0 ) { // Let it fill up.
        connection_state[0].tcb = tcb;
        connection_state[0].blocked_tx = true;
        if ( tcb) forth_thread_stop(&connection_state[0]);
        return(true);
        }
    else return(false);
    }
```

TCB status byte

Bit	When set	When Reset
0	Task is running	Task is halted
1	Message pending but not read	No messages
2	Event triggered	No events
3	Event handler has been run	No events (reset by user)
4..	User defined	User defined

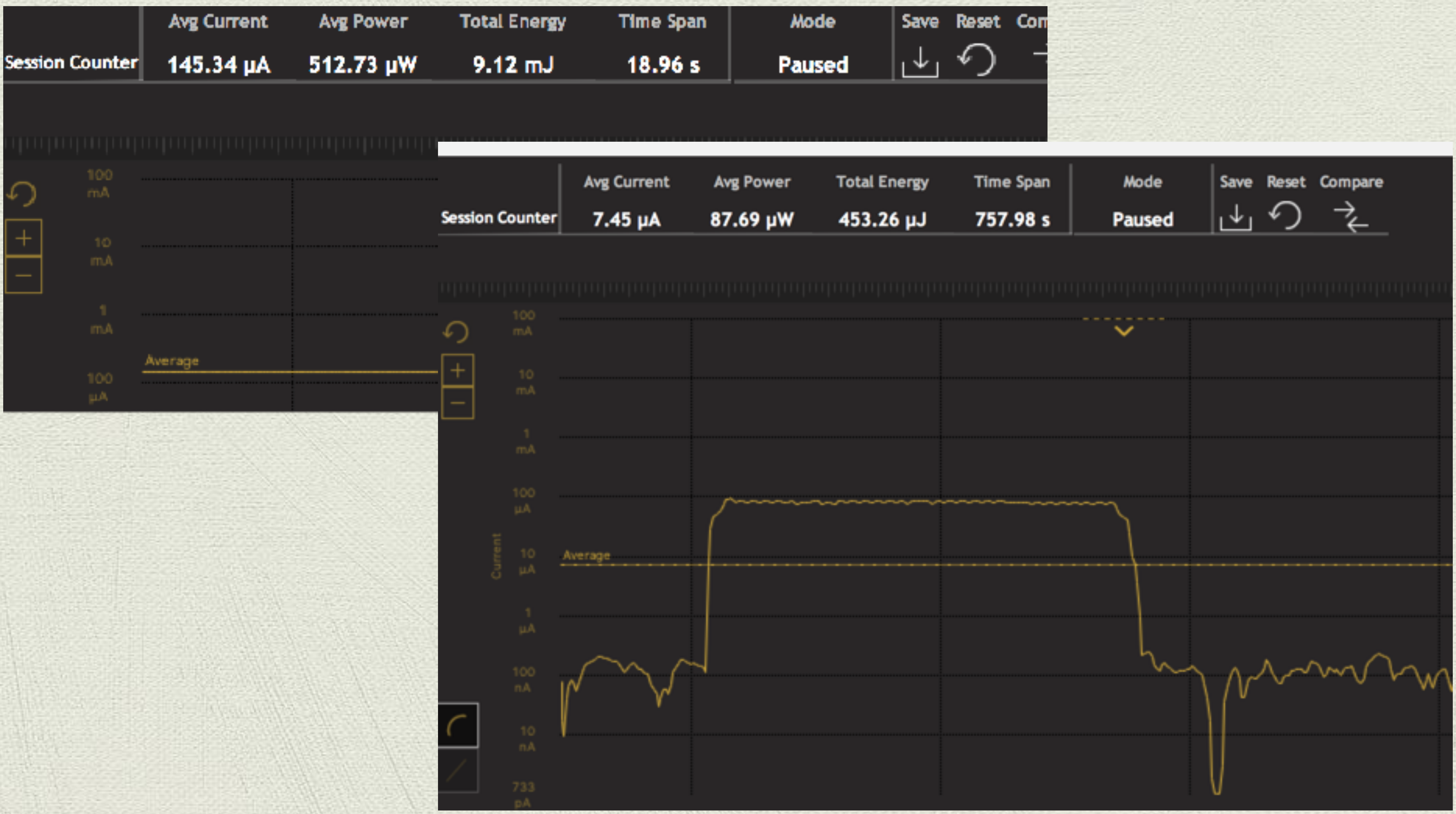
- Scheduler uses non-zero value to trigger task execution or event hander
- ISR can set the event bit to trigger task execution
- Scheduler calls event handler then the task

Bring up - Start small!

- Exercise the low level driver

```
const char message[] = "Boot!! ";
void SayHello() {
    const char *p = message;
    while(*p) {
        // LEUART_Tx(LEUART0,*p++); // Direct
        // console_leuart_putchar(*p++,0); // function
        Putchar(0,*p++); // System call.
      }
    }
```

More sleeping!



Pitfalls

- Make sure your programmer is loading the right image.
- Build/steal a stitcher so you don't have to build both parts
- Remember to erase all of forth's memory before you launch it.
  - Forth should probably do this itself.
- MPE is full-featured. It can be slimmed down.
- Watch memory allocation on small systems. Tasks need space.

Demo - Stitching

Forth
24k Flash/3k Ram

Application Development
Control Interface
Debugging/Bringup

Launcher - 8k/1k

Clock Init
UART Init

Stitching with a script

```
LAUNCHSIZE=$(( 2 * 4096 ))
FIRSTBINARY=supervisor/exe/supervisor.bin
SECONDBINARY=forth/TINY.img

cd supervisor; make; cd ..

set -- $( ls -l $FIRSTBINARY ); LEN=$5
PAD=$(( $LAUNCHSIZE - $LEN ))
set -- $( ls -l $SECONDBINARY ); LEN2=$5
TOT=$(( $LEN + $PAD + $LEN2))

echo "$FIRSTBINARY($LEN) + $PAD + $SECONDBINARY($LEN2) = $TOT"
{
    cat $FIRSTBINARY;
    dd if=/dev/zero bs=1 count=$PAD;
    cat $SECONDBINARY;
} > packaged.bin
```

https://github.com/rbsexton/gecko/blob/master/tiny/basic/build.sh

Stitching with MPE

```
\ *P The Flash memory starts at $0000:0000.
\ We own the whole thing, but we have to start at a 1k boundary.
\ to leave room for the launcher.   Its possible to include
\ it from here.

$0000:0000 $0000:1FFF cdata section Sup   \ Supervisor goes here.
 data-file supervisor.bin    $2000 swap  - allot \ CRITICAL!!!!

$0000:2000 $0000:7FFF cdata section Tiny   \ code
$2000:0400 $2000:06FF idata section PROGd \  IDATA - New words live here.
$2000:0700 $2000:0FFF udata section PROGu \  UDATA
```

https://github.com/rbsexton/gecko/blob/master/tiny/basic/forth/tiny.ctl

Stitching with SREC

```
# Generate a combined firmware.bin
# by producing a checksummed NXP binary, padding it out,
# and appending the forth image.

# srec_cat command file to generate a binary
# with a NXP Cortex vector checksum at 0x1C
# Usage: srec_cat @filename
# input file
launcher/exe/launcher.hex -Intel
-crop 0x0 0x1C # just keep code area for CRC calculation below
-Checksum_Negative_Little_Endian 0x001C 4 4

# insert the remainder of the file.
launcher/exe/launcher.hex -Intel -crop 0x20

forth/11UXX.img -binary -offset 0x2000

-Output firmware.bin -binary
```

https://github.com/rbsexton/nxp-cortex/blob/master/11u35/basic-i2c/packageit.srec

Stitching w/ LD for gdb

```
MEMORY {
 FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 65536
 FLASH2 (rx) : ORIGIN = 0x00010000, LENGTH = 3 * 65535
 RAM (rwx)  : ORIGIN = 0x20000000, LENGTH = 8192
 }
.forth :  { KEEP(*(.forth)) } > FLASH2
```

```
forth.o: ../usbforth/LEOPARD.img
   arm-none-eabi-objcopy -O elf32-littlearm \
      -B arm --rename-section\
      .data=.forth -I binary ../usbforth/LEOPARD.img\
      forth.o
```

https://github.com/rbsexton/gecko/blob/master/leopard/usbsupervisor/Makefile

https://github.com/rbsexton/gecko/blob/master/leopard/usbsupervisor/efm32lg-package.ld

Loading the binary

- Vendor Tools / Openocd
  - Load a binary and tell it where
  - Use intel .hex format - it specifies a memory address
- gdb - use the one that got built with gcc.
  - Not so good with binary files.   You can use objcopy to make them into .elfs

```
# Generate a .o file for use with gdb & the Black Magic probe.
arm-none-eabi-objcopy -O elf32-littlearm \
   -B arm --rename-section .data=.text\
   -I binary packaged.bin packaged.elf
```

Demo - Threads

Resources

- https://github.com/rbsexton/cm3lib
  - Assembly files for launching forth
  - lockless ringbuffers
- https://github.com/rbsexton/cm3forthtools
  - Atomic operatons for Forth
  - AAPCS wrappers
  - Improved scheduler

Questions?

http://www.kudra.com/forth

Advanced Techniques

Lock-Avoidance

- Cortex-M3 and up - LDREX/STREX

```
code BICEX! \ addr mask --
   ldr r0, [ psp ], # 4 \ Address
L$1:
    ldrex r1, [ r0 ]
    bic r1, r1, tos
    strex r2, r1, [ r0 ]
    cmp r2, # 0
    b .ne L$1
    ldr tos, [ psp ], # 4
    next,
end-code
```

- Cortex-M0 - Irq Disable

https://github.com/rbsexton/cm3forthtools/blob/master/CortexM3Atomic.fth

https://github.com/rbsexton/cm3forthtools/blob/master/CortexM0Atomic.fth

Run-time linking 1/2

```
static volatile uint32_t tick_cnt;

// There must be a matching forth structure for this.
typedef struct {
        volatile uint32_t *ticks;

} tSharedData;
tSharedData theshareddata
        // This section is pinned to the
        // beginning of SRAM.  linker must KEEP() it.
        __attribute__ ((section(".shareddata"))) =
        { &tick_cnt };
```

The beginning of SRAM is a reliably-known location thats similar across Cortex-M devices.

Run-time linking 2/2

```
\ Access to the interconnect things - It’s got to match the C side.
$10000000 equ ICROOT
struct /INTER   \ -- size
        int inter.ticks
end-struct
: ticks icroot inter.ticks @ ;
---------------------------------------
dasm ticks
TICKS
( 0000:6894 0248 .H )        ldr r0, [ PC, # $08 ] ( @$68A0=$10000000 )
( 0000:6896 0568 .h )        ldr r5, [ r0, # $00 ]
( 0000:6898 361F 6. )        sub .s r6, r6, # $04
( 0000:689A 3760 7` )        str r7, [ r6, # $00 ]
( 0000:689C 2F46 /F )        mov r7, r5
( 0000:689E 7047 pG )        bx LR
12 bytes, 6 instructions.
```

Simple, but must be hand-maintained. Scripting/Automation is required for this to scale

Run-time linking 3/2

```
typedef struct {
    // This union is a bit crazy, but its the simplest way of
    // getting the compiler to shut up.
    union {
        void (*fp) (void);
        int*  ip;
        unsigned int    ui;
        } p;   ///< Pointer to the object of interest (4)
    int16_t size;  ///< Size in bytes (6)
    int16_t count; ///< How many (8)
    int8_t kind;   ///< Is this a variable or a constant? (9)
    uint8_t strlen;   ///< Length of the string (10)
    const char name[DYNLINKNAMEMLEN]; ///< Null-Terminated C string.
    } runtimelink_t;

const runtimelink_t dynamiclinks[] __attribute__((aligned( sizeof(runtimelink_t) ))) = {
 { { .ui  = sizeof(runtimelink_t)  },                 0, 0, 'C', FORTHNAME("RECORDLEN") },
 { { .ulp = &g_ulSystemTimeMS      }, sizeof(uint32_t), 1, 'V', FORTHNAME("SYSTIMEMS") },
 { { .fp = (void (*) (void)) &getMSFunction}, sizeof(uint32_t), 1, 'C', FORTHNAME("TEST-FN") },
 { { .ui = 0 } ,0,0,0,FORTHNAME("") }
 };
```

https://github.com/rbsexton/sockpuppet/blob/master/sapi/sapi-dylink.h

https://github.com/rbsexton/sockpuppet/blob/master/sapi/sapi-dylink.c

https://github.com/rbsexton/sockpuppet/blob/master/forth/dylink.fth

Getting to User / Thread

- Why? MPU can only usefully trap user faults
- NVIC Uses the link register to trigger the change
- Build a fake stack so it looks like a system startup
  - reset the stack pointer
  - install the program counter into the fake stack
  - initalize the status register

R0
R1
R2
R3
R12
LR
PC
xPSR

https://github.com/rbsexton/sockpuppet/blob/master/sapi/pendsv-launcher.c