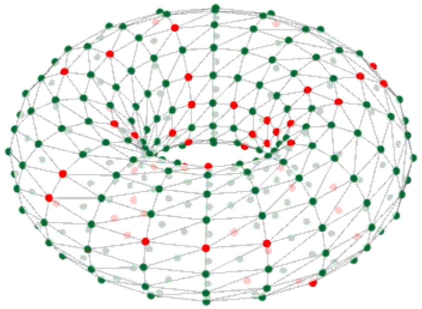
SpiNNaker System Software



Steve Temple
SpiNNaker Workshop – Manchester – Sep 2016

European Research Council
Established by the European Commission

Human Brain Project

Overview

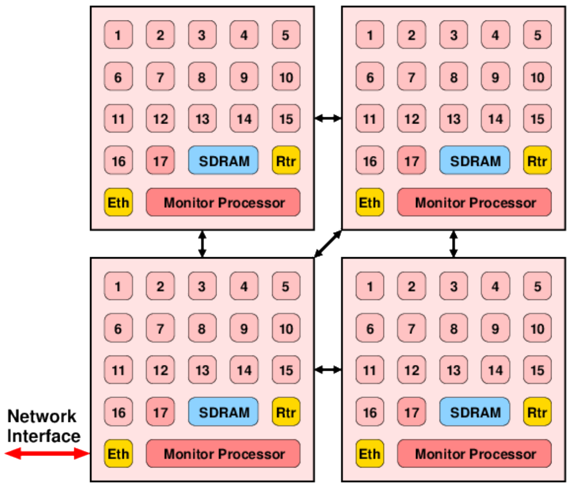
- SpiNNaker applications and their environment
- SC&MP, *ybug* and application loading
- SARK (SpiNNaker Application Runtime Kernel)
  - Application start-up
  - SARK function library
  - Examples
  - Documentation

Please interrupt if you have a question!

Building Applications

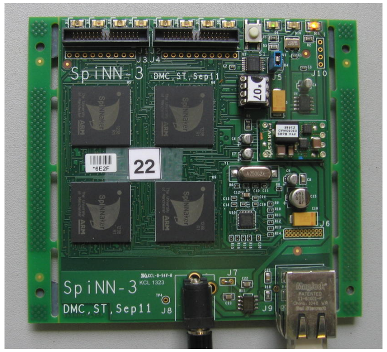
- Languages – mostly C with bits of assembler
- Toolchain choice
  - ARM tools – RVDS 4 and DS-5 (free for academics)
  - GCC – GNU ARM Embedded Toochain (free)
- Library support
  - Toolchain libraries – C library functions, maths, etc
  - SARK – low-level SpiNNaker support library
  - Spin1 API – event-based application library
- Linking – support libs + application code
  - Creates application to be loaded
  - Application file format is APLX

Execution Environment (1)



1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 SDRAM Rtr
Eth Monitor Processor

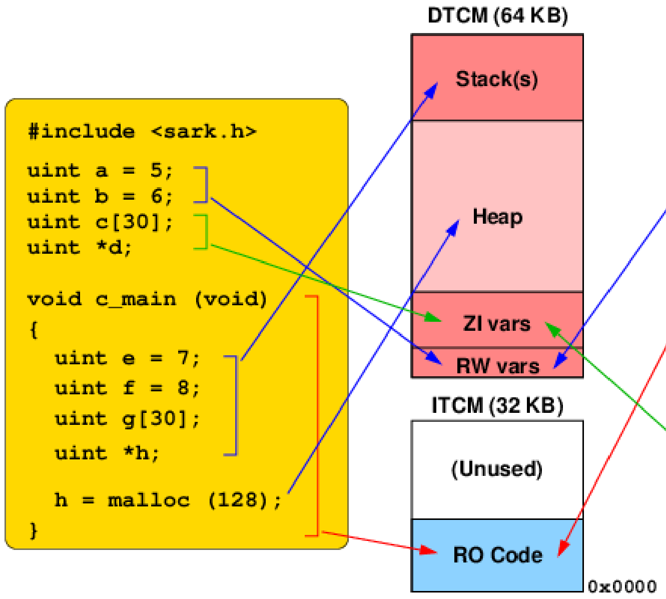
Network Interface



Execution Environment (2)

- One application per core
- Executable code (instructions) in ITCM (32 KB)
- Data (variables, stacks, heap) in DTCM (64 KB)
- Bulk and/or shared data in SDRAM (128 MB)
- Code/data access from ITCM/DTCM is fast (5 ns)
- Data access to SDRAM is slow (> 100 ns) and subject to contention
- DMA controller in each core can move bulk data between I/DTCM and SDRAM faster (~ 15 ns/word) without requiring CPU

Mapping Program to Memory



```
#include <sark.h>

uint a = 5;
uint b = 6;
uint c[30];
uint *d;

void c_main (void)
{
  uint e = 7;
  uint f = 8;
  uint g[30];
  uint *h;

  h = malloc (128);
}
```

SC&MP

- “SpiNNaker Control & Monitor Program”
- Loaded onto all Monitor Processors during bootstrap
- Communicates with host computer using SCP (SpiNNaker Command Protocol) over SDP
- Supervises operation of a single chip
- Allows program loading to Application Cores
- Acts as router for SDP packets between any pair of cores or with external Internet endpoints
- Flashes the LED!

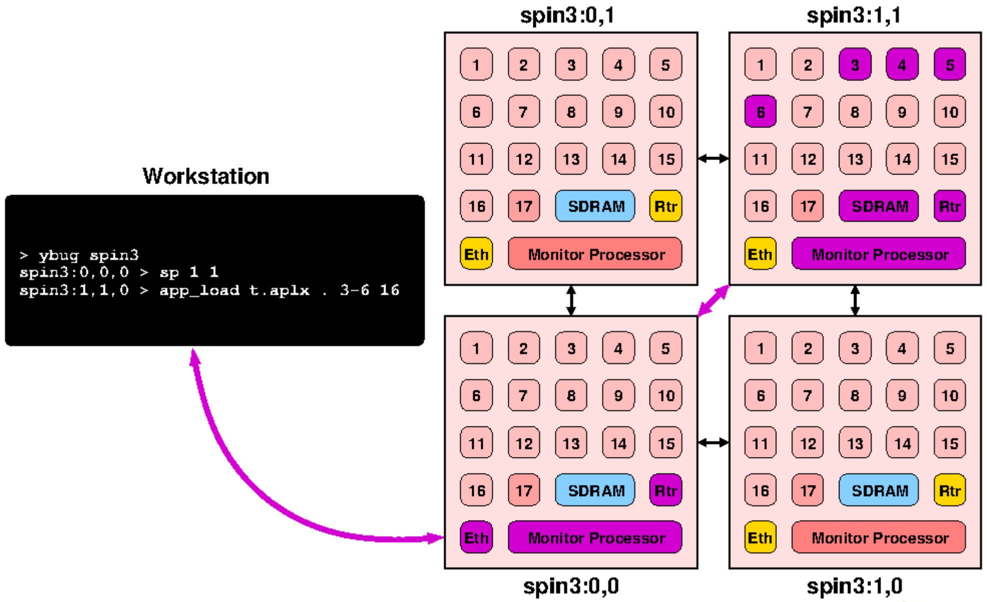
SC&MP, SCP and *ybug*

- SC&MP provides command interface via SCP
  - `Ver` – give S/W version, etc
  - `Read (addr, length)` – read SpiNNaker memory
  - `Write (addr, length, data)` – write SpiNNaker memory
  - `Reset (core_mask)` – reset Application Cores
- Host (workstation) embeds SCP/SDP in UDP/IP to talk to SpiNNaker Monitor Processor on the Root Chip
- *ybug* is a simple command-line tool which runs on a workstation and provides an interface to SC&MP for application loading and debug

Application Loading (1)

- *ybug* sends the application APLX to the relevant SpiNNaker chips.
- The APLX image is copied to a known place in shared memory
- *ybug* requests that the relevant Application cores are reset.
- The reset code is an *APLX loader* which loads the image according to instructions in the APLX header
- This usually results in the application being copied into ITCM and entered at address zero (the ARM reset vector)

Application Loading (2)



```
> ybug spin3
spin3:0,0,0 > sp 1 1
spin3:1,1,0 > app_load t.aplx . 3-6 16
```

SARK

- SpiNNaker Application Runtime Kernel
- Three main functions
  1) Application start-up
  2) Library of useful functions
  3) Communication via SDP with Monitor Processor (and hence rest of system)
- SARK is automatically linked with applications when they are built
- Occupies around 3 KB in the image

Application Start-Up

- Start-up code at start of ITCM is SARK
  - Configures stacks for 4 ARM execution modes
  - Initialises Heap and SDP message buffers in DTCM
  - Initialises shared-memory data structure (VCPU)
  - Calls a function to do pre-application set-up
  - Calls the function **`c_main`**, the application entry point
  - Calls a function to do post-application clean-up
  - Goes to sleep!
- Some applications will never terminate
- SARK provides SDP communications with the application

SARK Library (1)

- CPU control
  - Interrupt disabling and enabling
  - Entering low power (sleep) mode
- Memory manipulation
  - Memory copy and fill (small footprint)
  - SDP message copying
- Pseudo-Random number generation (32-bit)
- SDP messaging
  - Message allocation in DTCM and shared memory
  - SDP message transmission

SARK Library (2)

- Text output via “printf”
  - Text sent to a host system using SDP packets
  - Text buffered in SDRAM
- Hardware locks and semaphores
- Memory management
  - `malloc/free` for DTCM heap
  - `malloc/free` for shared memories (eg SDRAM) with locking
  - `malloc/free` for router MC routing table
- Environment queries
  - What is my core ID, chipID, etc

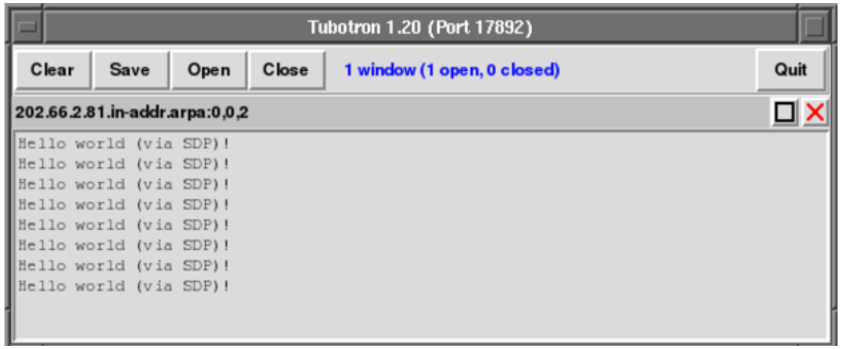
SARK Library (3)

- Hardware interfaces
  - LED control
  - Router control – setting MC and P2P table entries
  - VIC control – allocating interrupt handlers to specific hardware interrupts
- Timer management
  - Routines to schedule/cancel events at some time in the future
- Event management
  - Routines to associate events with interrupts
  - Management of priority event queues

SARK – Example 1

```
#include <sark.h>

void c_main (void)
{
  io_printf (IO_STD, "Hello world (via SDP)!\n");
  io_printf (IO_BUF, "Hello world (via SDRAM)!\n");
}
```


Tubotron 1.20 (Port 17892)
Clear
Save
Open
Close
1 window (1 open, 0 closed)
Quit
202.66.2.81.in-addr.arpa:0,0,2

```
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
Hello world (via SDP)!
```

SARK - Example 2

```
#include <sark.h>

INT_HANDLER timer_int_han (void)
{
  tc[T1_INT_CLR] = (uint) tc;      // Clear interrupt in timer
  sark_led_set (LED_FLIP (1));     // Flip a LED
  vic[VIC_VADDR] = (uint) vic;     // Tell VIC we're done
}

void timer_setup (uint period)
{
  tc[T1_CONTROL] = 0xe2;                 // Set up count-down mode
  tc[T1_LOAD] = sark.cpu_clk * period;  // Load time (us)
  sark_vic_set (SLOT_0, TIMER1_INT, 1, timer_int_han);
}

void c_main ()
{
  io_printf (IO_STD, "Timer interrupt example\n");
  timer_setup (500000);       // (0.5 secs)
  cpu_sleep ();               // Send core to sleep
}
```

Documentation & Help

- SARK – notes in SpiNNaker Tools - `docs/sark.pdf`
- *ybug* – user guide in SpiNNaker Tools – `docs/ybug.pdf`
- `“spinnaker.h”` - describes the SpiNNaker hardware – memory maps, peripheral registers...
- `“sark.h”` describes all SARK data structures and functions. Commented in Doxygen style.
- All source code is provided...
- If desperate, talk to us!