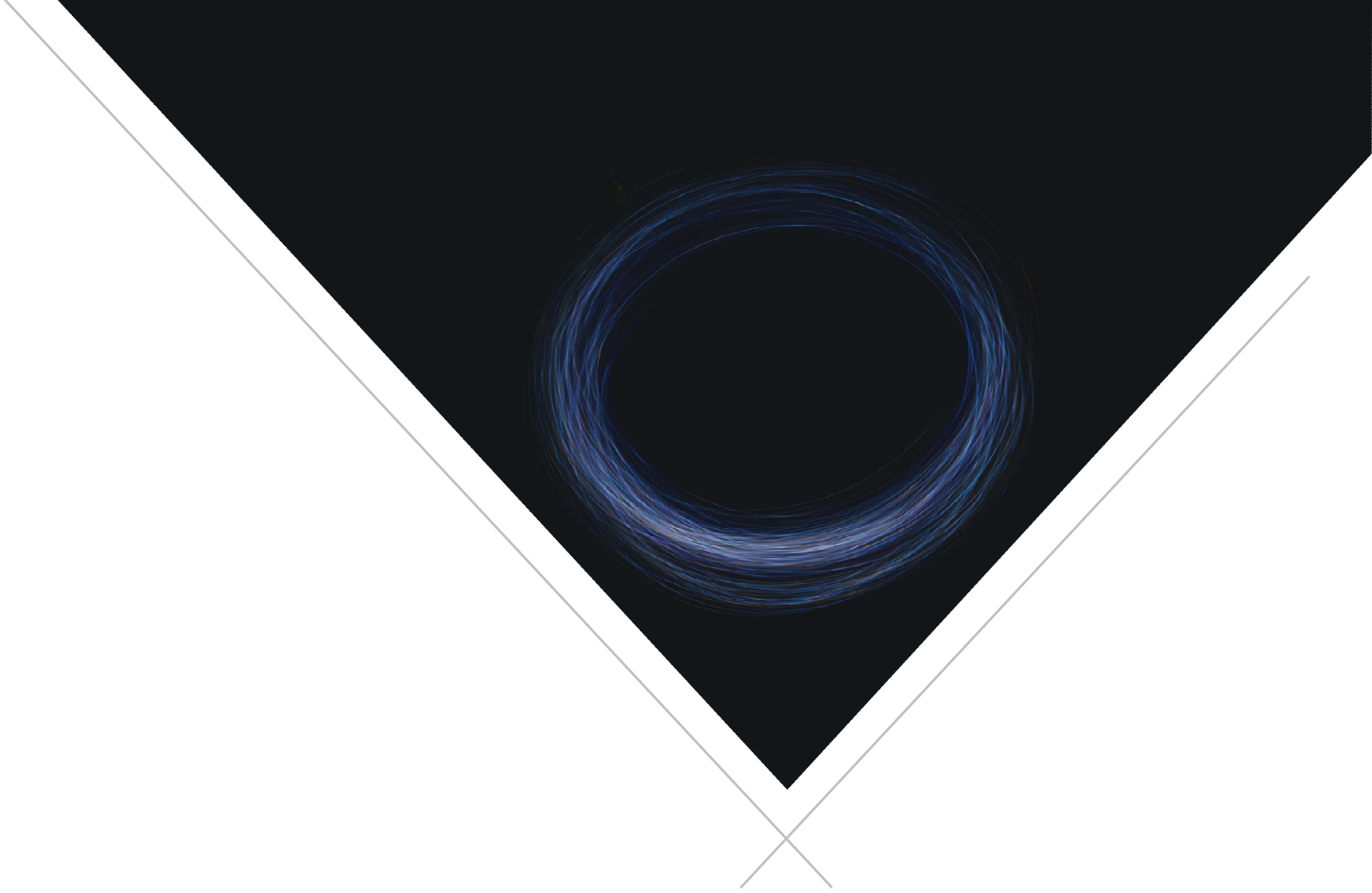


XTIMECOMPOSER TOOLS

A QUICK INTRODUCTION

20 AUGUST 2020

1. INTRODUCTION

XCORE is a family of cross-over processors for the IoT and AIoT, helping you to get to market fast, with products that stand out from the competition. With xcore multiple cores are available for the execution of real-time inferencing, decisioning at the edge, signal processing, control and communications - all wrapped up in a single chip. This means that Product Designers no longer need to rely on a costly applications processor or a microcontroller with additional components.

The xTIMEcomposer tools help our customers and partners get to market fast with xcore based designs. The tools help our users to develop, debug and understand the performance of their designs, even before hardware exists. The tools then support the deployment and debug of those designs onto target hardware.

The xcore is the only processing resource needed in many applications - designs for xcore frequently combine AI, DSP, control and IO processing in a highly integrated and cost-effective solution. xTIMEcomposer is the only embedded processor tool suite that enables co-development of all four of these classes of processing in a single environment, thereby enabling significant time to market benefits.

xTIMEcomposer is designed to be very familiar and easy to use for any embedded programmer. Indeed, in many cases the same open-source tools are used as are frequently used for platforms like the ARM Cortex M-series.

2. THE XTIMECOMPOSER TOOLS

The xTIMEcomposer tools include:

- TOOLCHAIN: stack usage aware LLVM-based C/C++ compiler1, assembler and linker
- DEVELOPMENT TOOLS: debugger, hardware debug adaptor with high-bandwidth SW debug probe capability.
- SIMULATION: SoC-level cycle accurate simulation
- DEPLOYMENT: generator and writer of flash images for secure boot of user applications

3. PROGRAMMING IN C

C developers new to the xcore can leverage their existing skills using the powerful xcore platform and the open-source LLVM compiler supplied with xTIMEcomposer.

The LLVM Clang C/C++ compiler is directly suitable for generation and rigorous optimisation of the xcore instruction set. System libraries containing APIs and macros that allow complete use of the xcore hardware in the C language are provided.

For instance, here is a trivial "Hello world" example:

```
#include <stdio.h>

int main(void) {
  printf("Hello world\n");
}
```

Forking and joining a pair of truly parallel tasks each onto their own xcore core is equally simple:

```
#include <stdio.h>
#include <xcore/parallel.h>

DECLARE_JOB(print_int_sum, (int, int));
void print_int_sum(int a, int b)
{
 int result = a + b;
 printf("Int sum is: %d\n", result);
}

DECLARE_JOB(print_float_sum, (float, float));
void print_float_sum(float a, float b)
{
 float result = a + b;
 printf("Float sum is: %.1f\n", result);
}

int main(void)
{
 PAR_JOBS(
  PJOB(print_int_sum, (10, 5)),
  PJOB(print_float_sum, (10.0, 5.0))
 );
}
```

Creating event handlers with lightning-fast response times is similarly straightforward. The following "lap timer" example counts timer events until the button is pressed at which point it prints and resets the counter. A guard pauses counting whilst the button is down:

```
#include <platform.h>
#include <stdio.h>
#include <xcore/hwtimer.h>
#include <xcore/port.h>
#include <xcore/select.h>

static const long unsigned timer_interval = 100000;

void lap_timer(port_t button_port, hwtimer_t timer)
```

```
10. {
11.   unsigned counter = 0;
12.   unsigned long port_value = port_peek(button_port),
13.                 button_mask = 0x1;
14.
15.   port_set_trigger_in_not_equal(button_port, port_value);
16.   hwtimer_set_trigger_time(timer, hwtimer_get_time(timer) + timer_interval);
17.
18.   SELECT_RES_ORDERED(
19.     CASE_THEN(button_port, on_button_change),
20.     CASE_GUARD_THEN(timer, port_value & button_mask, on_timer_tick))
21.   {
22.   on_timer_tick:
23.     hwtimer_change_trigger_time(timer, hwtimer_get_time(timer) + timer_interval);
24.     counter += 1;
25.     continue;
26.
27.   on_button_change:
28.     port_value = port_in(button_port);
29.     port_set_trigger_value(button_port, port_value);
30.
31.     if (~port_value & button_mask)
32.     {
33.       printf("Counter value: %u\n", counter);
34.       counter = 0;
35.     }
36.
37.     continue;
38.   }
39. }
40.
41. int main(void)
42. {
43.   hwtimer_t timer = hwtimer_alloc();
44.   port_t button_port = XS1_PORT_4D;
45.   port_enable(button_port);
46.
47.   lap_timer(button_port, timer);
48.
49.   port_disable(button_port);
50.   hwtimer_free(timer);
51. }
```

4. AVOIDING STACK OVERFLOW

Ensuring against stack overflow in a complex parallel program is notoriously difficult. When it does occur, it can be incredibly time-consuming and costly to debug. If these issues remain resident in production firmware, expensive product recalls may be required. For this reason, the xTIMEcomposer toolchain performs callgraph analysis, stack size calculation and constraint checking. Compilation and linking of the parallel program above, for instance, generates the following guidance:

```
$ xcc -target=XCORE-AI-EXPLORER -report -g main.c -o main.xe
Constraint check for tile[0]:
  Memory available:       524288,   used:      22664 .  OKAY
    (Stack: 1644, Code: 19728, Data: 1292)
Constraints checks PASSED.
```

Even the stack usage of indirect function calls can be managed with the xTIMEcomposer toolchain using a concept known as “function pointer groups” – a function call via a function pointer is assumed to require the greatest stack of the members of the associated function pointer group.

5. PLATFORM-LEVEL SUPPORT

The xTIMEcomposer tools have an inbuilt understanding of the xcore processor itself, the other on-chip resources and the typical configurations in which it is likely to be used.
This allows developers to focus on the application, not the time-consuming (but still essential) boilerplate functionality around it. For instance, without recompilation, you can take your already proven application and deploy it in a secure flash-booted multi-chip network using the xTIMEcomposer’s xflash tool.

6. SEAMLESS SIMULATION

The xTIMEcomposer tools include the xsim simulator. xsim simulates one or more xcore.ai chips in a network with optional external flash and LPDDR devices (xcore.ai only). For deployments of the xcore alongside user-supplied external devices, an API is provided to allow the simulation to incorporate the entire system.

The xsim simulator is straightforward to invoke and invaluable when wanting to understand detailed processing, IO or memory interactions. Detailed logs of every instruction are produced with cycle-accurate results in most cases allowing a level of insight into your running program not available by running on the hardware directly.

7. SQUASHING BUGS

The xTIMEcomposer xgdb tool is a multi-core extension of the popular gdb debugger. Familiar usage is enhanced with some additional commands. In the example below, we debug the parallel application above on an xsim simulated target, first setting a breakpoint in one of the parallel functions:

```
$ xgdb main.xe
(gdb) connect -s
0x00080000 in _start ()
(gdb) break print_float_sum
Breakpoint 1 at 0x80128: file main.c, line 14.
```

```
(gdb) run
[Switching to tile[0] core[1]]

Breakpoint 1, print_float_sum (a=10, b=5) at main.c:14
14       float result = a + b;
(gdb) info threads
  3  tile[1] core[0]  0x00080006 in _start ()
* 2  tile[0] core[1]  print_float_sum (a=10, b=5) at main.c:14
  1  tile[0] core[0]  0x0008010e in print_int_sum (a=10, b=5) at main.c:8
(gdb) where
#0  print_float_sum (a=10, b=5) at main.c:14
#1  0x000801e8 in __xcore_ugs_shim_print_float_sum (__xcore_pargs_=0x84e08) at main.c:11
#2  0x00081e90 in memmove ()
Backtrace stopped: frame did not save the PC
(gdb) thread 1
[Switching to thread 1 (tile[0] core[0])]#0  0x0008010e in print_int_sum (a=10, b=5) at
main.c:8
8        printf("Int sum is: %d\n", result);
(gdb) where
#0  0x0008010e in print_int_sum (a=10, b=5) at main.c:8
#1  0x000801c0 in main () at main.c:20
```

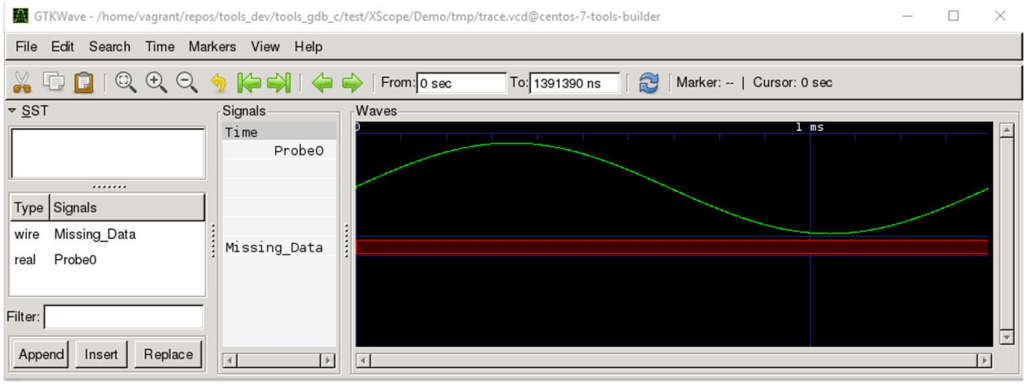
The ability to stop and debug a running real-time application is not always useful or even relevant, as it immediately breaks real-time behaviour. For that reason, the xscope high bandwidth debugging interface is provided as an inbuilt capability of the xtag hardware debugging interface.

Simple code annotations with negligible real-time overhead allow for detailed off and online tracing and analysis. The example overleaf uses xscope_float() to instrument a dummy piece of code:

```
1.  #include <math.h>
2.  #include <platform.h>
3.  #include <xscope.h>
4.
5.  #define NUM_SAMPLES 1000
6.  float samples[NUM_SAMPLES];
7.
8.  typedef enum {
9.   ch_continuous_e=0,
10. } xscope_channels_t;
11.
12. void xscope_user_init(void)
13. {
14.  xscope_register(1,
15.        XSCOPE_CONTINUOUS, "Data0",   XSCOPE_FLOAT, "Data");
16. }
17.
18.
19. void prepare_samples()
20. {
21.  float pi  = 2 * asinf(1);
22.  float step = (2*pi)/NUM_SAMPLES;
23.
24.  for (unsigned i=0; i<NUM_SAMPLES; i++) {
25.   samples[i] = sinf(i*step);
26.  }
27. }
28.
29. void output_samples()
30. {
31.  for (unsigned i=0; i<NUM_SAMPLES; i++) {
```

```
32.    xscope_float(ch_continuous_e, samples[i]);
33.   }
34.  }
35.
36.  int main()
37.  {
38.   xscope_mode_lossless();
39.
40.   prepare_samples();
41.
42.   output_samples();
43.
44.   return 0;
45.  }
```

When the example is built and run with the appropriate command line options, a file in the IEEE VCD format is produced. GTKWave or a similar 3rd-party VCD viewer can be used to interrogate the output:



8. SUMMARY

xTIMEcomposer is a full suite of development tools available for all three generations of the xcore processor. These tools allow users a unique opportunity to combine AI, DSP, control and IO processing together to expedite their time to market and deliver the most integrated and economical solutions. The capabilities of xcore, and the ease-of-use of the xTIMEcomposer tools represent a distinct advantage for those developing a diverse range of solutions for the IoT and AIoT.

Visit xmos.com for more information.