ENTRA

318337

Whole-Systems ENergy TRAnsparency

Benchmark Suites

Deliverable number:	D5.1
Work package:	Establishing the Benchmarks (WP5)
Delivery date:	1 October 2013 (12 months)
Actual date:	13 November 2013
Nature:	Report
Dissemination level:	PU
Lead beneficiary:	XMOS Limited
Partners contributed:	Roskilde University, University of Bristol, IMDEA Software Institute, XMOS Limited

Project funded by the European Union under the Seventh Framework Programme, FP7-ICT-2011-8 FET Proactive call.

Short description:

This deliverable documents the benchmarks that we envisage to use throughout the ENTRA project. This document complements the actual source file of all benchmarks, and documents the meaning of the code, the intention of the code, and how this code may benefit from the ENTRA treatment.

The benchmarks roughly fall into three categories: code that has been written in different versions that demonstrate different power profiles that may be achieved semi automatically using program optimisations and transformations; code that can be analysed at source level to demonstrate transparency of throughout the system layers; and code that can be used as a regression suite.

Contents

1 Introduction

The purpose of this deliverable is to document the benchmarks that can be used to develop the modelling, analysis, and transformation tools that are developed in the remainder of the project. This document contains the description of the benchmark codes, with source code available in the project repository, but included in the appendices for completeness.

Benchmarks are self-contained sections of code that can be used to test algorithms and tools, yet that are small enough that the results can be interpreted by humans. *Benchmarks* (also known as *kernels*) are sections of code that have been extracted from real-life programs, or that are synthesised to look like code that could form the core of a program. Their use for evaluation is limited, as they are small and often concern specific aspects. This as opposed to *case studies* that are part of Workpackage 6. *Case studies* are real-life programs that have been selected as representative example of an application where energy requirements need to be analysed or optimised.

Benchmarks are sections of code that can be analysed in fine detail, because of their limited size. The benchmarks will enable us to make a head-start in understanding how to specify constraints and requirements. They also allow a detailed evaluation of which strategies do and don't work.

The programs listed in this deliverable serve three purposes as far as the project is concerned:

1. To provide a set of benchmarks that can be used to develop and test the modelling and analysis tools

2. To provide a set of benchmarks that can be used to test and develop the transformation and optimisation tools.

3. To provide a set of benchmarks that can be used as a regression suite.

In order to aid the first goal, we provide a set of benchmarks that have known properties that analysis tools could extra automatically.

In order to serve the second goal, we provide a set of benchmarks that have been written in different styles, providing a hand-optimised version that can be compared against a version that a software engineer may write.

The third goal is served by collating all sections of code, and then extend that set by using different compilation flags. For example, a program compiled with -O2 is as valid as a program compiled with -O0 for generating the regression suite.

Section 2 presents benchmarks that have been written in more than one style, and can therefore be used as a comparison for the optimisation and transformation tools. Section 3 presents

benchmarks that have been written in a single style; they can be used for modelling and analysis as is, can be part of a regression suite, and can be used as test cases for the optimisation and transformation tools.

2 Benchmarks with multiple versions

2.1 Equaliser using biquad filtering

The *equaliser* benchmark comprises a relatively small piece of code with many potential uses and changes. The purpose of an equaliser is to take a signal, and to attenuate/amplify different frequency bands. This will, for example in the case of an audio signal, correct for a speaker or microphone frequency response. Equalisers may have few bands (Bass, Mid, Treble) or many bands (for example half octaves), and they may be factory set or be user settable.

One implementation of an equaliser uses a cascade of *Biquad filters*. Each biquad filter attenuates or amplifies one specific frequency range; the signal is sequentially passed through all filters, eventually attenuating or amplifying all bands.

Typically, the designer will decide on the number of bands required, the filter settings for each biquad, and the precision of the coefficients and intermediate results. From an energy consumption perspective, the following parameters have a direct effect on the energy:

- The sample rate of the signal. Typical sample rates for audio signals are 44.1, 48, 96, or 192 KHz. A higher sample rate gives better fidelity, although there is no agreement between audio experts whether a 192 KHz signal is audibly different. The number of computations required per second is linear in the sample rate.

- The number of banks, typically between 3 and 30 for an audio equaliser. A higher number of banks enables the designer to create more precise frequency response curves. Computation is linear in the number of banks.

- The precision of the samples, coefficients, and intermediate results. Typical sample widths are 16 or 24 bits, typically coefficients are 32 bit numbers (representing a value of between -2 and 2), and typical intermediate results are 48-64 bit values. Using lower precision reduces fidelity.

  Power consumption will be better if there are fewer bits in the computation, and power consumption will be affected by the alignment of the numbers: right aligned signed values are prefixed with all ones or all zeros, left aligned signed numbers are postfixed with all zeroes which may be preferable.

- The cascade can be pipelined and spread over multiple threads. This will enable a lower clock frequency, and therefore voltage scaling; reducing power consumption.

- The cascade can be spread over multiple tiles, enabling further scaling of voltage and frequency, but at the expense of communication.

2.1.1 Biquad filters - a tutorial

A biquad filter comprises six coefficients, and uses the three most recent input samples and two most recent output samples to compute a new output sample:

$$y_0 = \frac{x_2 \times b_2 + x_1 \times b_1 + x_0 \times b_0 + y_2 \times a_2 + y_1 \times a_1}{a_0}$$

Where x_0 is the most recent input sample, x_1 is the previous input sample, y_1 is the most recent output sample, etc. Note that the division by a_0 can be distributed over the sum, reducing the effective number of coefficients to five: $a_1/a_0, a_2/a_0, b_0/a_0, b_1/a_0,$ and b_2/a_0.

Setting biquad coefficients is an art. A good set of formulas can be found in the *Audio Equaliser Cookbook* [BJ]. They have to be chosen carefully as the filter has a feedback loop built-in: each output sample is used in the next iteration of the filter. Choosing filter values without care might create an unstable filter or oscillator. Reducing precision too far may also introduce instabilities. For our benchmark, we have created two fixed sets of filter coefficients.

If coefficients must be changed at run-time, then this should also be done carefully as this may cause instability. Typically, the adjustments are made in small steps over a few hundred samples in order to avoid those instabilities. Restarting the filter with fresh coefficients is an option but this will produce an audible 'click' and is hence to be avoided.

2.1.2 Code

The code is listed in Appendix A. It comprises three directories, with two source files (the actual benchmark and a test harness with standardised input) and a Makefile. The standard equaliser settings are to use a seven bank filter, of which the low four banks are set to amplify, and the top three banks are set to attenuate:

- Frequencies below 1100 Hz are amplified by 20 dB
- Frequencies above 1100 Hz are attenuated by 20 dB

2.1.3 Versions

There is a sequential and two parallel versions; all use 24-bit coefficients and 24-bit sample values. The single-tile parallel version (7 threads) runs at 150 MHz, vs 450 MHz for the sequential version. The multi-tile parallel version runs at 75 MHz. All deliver the same filter performance.

Power consumption figures are:

- 202 mW for the sequential version; down to 177 mW with scaling to 0.95 V

- 166 mW for the parallel version; down to 93 mW with scaling to 0.75 V
- 163 mW for the multi-tile version; down to 82 mW with scaling to 0.70 V

These are all measured on an A16A device powered from 3V3. The figures includes losses incurred in the power supply, and are average measurements of the middle of the run.

2.2 Finite Impulse Response filter

The FIR filter (Finite Impulse Response) is different from the previous filter in that it simply computes the inner-product of two vectors: a vector of *input samples*, and a vector of *coefficients*. The inner-product is the output samples. Typically FIR filters have a large number of coefficients and can therefore be less efficient than Biquad filters. However, FIR filters do not have any feedback and are hence stable by design.

Typically, the designer will decide on the number of taps required, the coefficients, and the precision of the coefficients and intermediate results. Unlike a biquad filter, a single FIR filter can incorporate any number of low-pass, high-pass or pass-band filters. From a energy consumption perspective, the following parameters have a direct effect on the energy:

- The sample rate of the signal. Typical sample rates for an audio signal are are 44.1, 48, 96, or 192 KHz. As the number of coefficients of the FIR is linear in the sample rate of the frequency, the number of computations required second is typically $O(samplerate^2)$.
- The number of coefficients. The more coefficients, the higher the fidelity, and the lower the frequencies that can be filtered.
- The precision of the samples, coefficients, and intermediate results. Typical sample width is 16 or 24 bits, typically coefficients are 32 bit numbers (representing a value of between -1 and 1), and typical intermediate results are 48-64 bit values. Using lower precision reduces fidelity.
- The filter can be parallelised over multiple threads and tiles, enabling frequency and voltage scaling, reducing power consumption. Parallelisation over tiles enables further scaling at the expense of communication.

2.2.1 FIR filters - a tutorial

An FIR filter computes

$$y_0 = \sum_{i=0}^{N-1} x_i a_i$$

Where x_i are the past N input samples, N is the number of banks, and a_i are the coefficients of the filter. Typically, samples are represented by signed integers, and coefficients by signed fixed point numbers.

2.2.2 Code

The code is listed in Appendix B. It comprises two source files (the actual benchmark and a test harness with standardised input) and a Makefile. The standard equaliser settings uses a 121 tap filter that removes frequencies with a wavelength of less than 8 samples.

2.2.3 Versions

There is a sequential and two parallel versions; all use 24-bit coefficients and 24-bit sample values. The single-tile parallel versions (7 threads) run at 150 MHz, vs 450 MHz for the sequential version. All deliver similar filter performance.

Power consumption figures are:

- 204 mW for the sequential version; down to 180 mW with scaling to 0.95 V,
- 166 mW for the parallel version; down to 92 mW with scaling to 0.75 V
- 159 mW for the parallel version with reduced fidelity; down to 89 mW with scaling to 0.75 V

These are all measured on an A16A device powered from 3V3. The figures includes losses incurred in the power supply, and are average measurements of the middle of the run.

These numbers are similar to the biquad numbers; which is to be expected since the type of computation is very similar. The gain of the final version may seem small (3.3%), but if this increases battery life by 20 mins than that may be worthwhile.

2.3 I2S Master

The I2S benchmark comprises a piece of code that mostly performs I/O. It implements the I2S (pronounced I-squared-S) protocol that enables a stereo audio signal to be transferred between two digital chips in a digital format. There is a large number of versions of the I2S protocol, for example versions that align the data to the left or right in a word, and there are many similar protocols that are used in embedded computing, such as I8S, TDM, or SPI.

With the large number of variations, there is value in describing the protocol in a simple, software-style manner (enabling different versions to be made with little work) and then mechanically creating versions that have the right energy and performance characteristics. This as opposed to making many optimised versions that have seemingly little relationship with each other. It is worth to note here that XMOS chips have no built-in hardware for I2S or other protocols, and implements any such protocol in software.

2.3.1 Introduction to I2S

I2S comprises two control lines and any number of data lines:

- The central clock is the *bit clock*, normally abbreviated to BCLK, which governs when bits of the data signal are transported. 32 edges of the BCLK transport a single sample (which may contain up to 32 bits), and 64 edges transport a stereo sample. A typical BCLK value is 3.072 MHz.

- The second signal is LRCLK, which stands for Left-Right clock, but is commonly known as the word clock. Confusingly; it is not a clock in the sense that it does not clock any of the other I2S signals. The level of LRCLK governs whether the sample transported is the *left* or *right* sample: when LRCLK is high, the left sample is transferred; when it is low the right sample is transferred. The right sample always follows the left sample. There are 64 BCLK edges for each LRCLK. A typical value for the LRCLK is 48 KHz.

- Each data line is unidirectional. Data is driven on the data line during the rising edge of the clock, and is sampled by the receiving side on the falling edge of the clock. In the case of a typical audio CODEC (Coder/encoder) there are two data lines: one from the Audio CODEC (called the ADC line, for Audio-to-Digital Conversion) and one to the Audio CODEC (called the DAC line, for Digital-to-Audio Conversion).

The node where the LRCLK and BCLK are generated is typically known as the *master* node, the other node is known as the *slave* node. If either side of the communication is a digital to analog converter or analog to digital converter, there is typically a third control signal MCLK, the Master-clock, which is a high speed signal that the ADC/DAC can use to perform its conversion. This signal runs a power-of-two faster than BLCK, for example 24.576 MHz for 8 master-clocks per bit clock.

Compared to other protocols, I2S always carries two channels on each data-line, notionally a *left* and a *right* channel. I8S/TDM carry eight channels on a data line, and hence has $8 \times 32 =$

256 bit clocks per LRCLK. In order to support the same data rate, the bit clock is increased to 12.288 MHz.

I2S uses the LRCLK to synchronise the data stream: a word representing a sample of data always starts at the same location relative to the rising edge of the LRCLK. Typically at 0, 1, or 8 clocks offset depending on the setting. Other serial protocols such as SPI use a *Chip Select* signal (CS) in order to identify the beginning of a segment of data; for example, a byte may start on the first clock after the falling edge of Chip Select.

2.3.2 Code

This benchmark comprises an I2S master with two data lines, one receive line called the ADC, and one transmit line, called the DAC; the problem scales to serving multiple lines. We presume for the sake of this benchmark that the MCLK frequency is fixed at 25 MHz, and that the LRCLK and BCLK should divide the MCLK by 512 and 8 respectively (to generate a 48,828 Hz wordclock with 64 bits per word.

In real life, the master frequency would be 24.576 MHz leading to a normal 48 KHz audio wordclock, but by fixing it at 25 MHz exactly we can provide a masterclock without any external hardware. Having more ADC/DAC channels, different frequencies, varying frequencies, or heavy lifting DSP are extensions that are optional. We could chose to add those to make the problem more complex.

An I2S Master comprises the following tasks:

1. Generating an LRCLK signal
2. Generating an BCLK signal
3. Output words to the DAC signal
4. Input words from the ADC signal
5. transferring data to and from a secondary thread
6. Optionally applying some form of digital signal processing to the data

2.3.3 Versions

There are three implementations of this benchmark:

1. a multi-threaded implementation where the threads synchronise by means of channels

2. a multi-threaded implementation where the threads synchronise by means of par statements inside a while loop

3. a single-threaded implementation where all tasks have been interleaved.

The first two implementations are descriptions that clearly spell out what the program is supposed to do. The first can be compiled and ran, the second implementation compiles onto hardware, but is too inefficient to execute, as the compiler currently does not optimise thread creation in a while-loop.

It is often uneconomical to spend multiple threads on this task; so it has to be a single thread. However, by scheduling all I/O carefully throughout the code (statically), the clock frequency can be slowed down significantly, making significant power savings.

This is why the third implementation is typically used: it uses the fewest resources. However, its description is ambiguous in that the explicit synchronisations provided by the earlier two versions are lost, and hence the compiler is not aware of any dependencies.

2.4 ADC to DAC benchmark

This benchmark code is an adaptation of a piece of code that an XMOS customer has developed for a battery operated device (reproduced with permission from the customer). The section of code that is of interest reads a value from an Analogue to Digital Converter (ADC), performs a small amount of processing on the data, and then creates a pulse-width-modulated analogue output. This code could, like in the previous example, be viewed as three independent processes that are merged together

2.5 Introduction

There are parts to this benchmark that all execute synchronously.

- A component that generates a regular clock pulse at a 768 kHz interval. This will request samples from the Analogue to Digital converter at a 768 KHz rate. In this particular task, two channels are sampled, and each channel is hence sampled at a 384 kHz rate.

- A component that picks up data that comes from the ADC. This component simply takes the data that must have arrived over the last period. This takes the form of inputting a word of data from a channel end, and inputting an END token from the same channel end.

- A component that processes the data and performs some very simple DSP on the data. The DSP performed takes two forms. Either a heterodyne mixer is applied (multiplying the signal with a sine-wave of a set frequency), or the data is stretched out.

Finally, an output signal is generated. The output is a pulse-width-modulated signal (PWM), that is the analogue output value is approximated by using a sequence of "ON" and "OFF" pulses, where the average over time equates to the analogue value. The "ON" and "OFF" pulses should be generated fast enough that they are inaudible and can be filtered out by an external LC filter, but slow enough that they will generate an efficient output signal without too much switching.

The trick is that we run all components off the same, virtual, 1,536 kHz clock:

- Every clock we invert the ADC sample pin, creating a 768 kHz ADC signal.
- Every four clocks, we input two ADC samples
- Every four clocks, we perform the DSP
- Every four clocks, we set the PWM signal high, and then low a fraction of the 2.6 us later.

The PWM signal is only generated with a 10 ns resolution, and hence there are only 2.6 us / 10 ns = 260 possible values, or 8 bits deep. In order to faithfully play a 16 bit signal, we must keep an accumulator with a 16-bit value, use the top 8 bits to generate the PWM value, and the bottom 8 bits as a error that we sum in during the next iteration.

2.5.1 Code

In contrast with the previous example, the synchronisation between the three sections of code is not governed by an external clock, but by means of data flow and I/O relative to the reference clock:

- Process 1 outputs a 768 kHz clock; an edge every 65 clock edges
- Process 2 inputs data on a 260 clock-tick interval (4 edges of process 1), and outputs the data to Process 3.
- Process 3 inputs data, processes it, and outputs data to Process 4
- Process 4 inputs data, and creates output signals on a 260 clock-tick interval.

The relative timing between processes 2 and 4 is not crucial (as long as it is not lagging behind hundreds of milliseconds); what is crucial is that the three processes all have the same throughput.

2.5.2 Versions

This benchmark comes in four variants:

- The first variant is described as four processes; this is one of the natural ways to formulate this benchmark. It would also be a power efficient way to execute the benchmark, but uneconomical.

- The second variant is described as a single process; this is also a reasonably natural way to describe it; but as presented this version needs a high clock frequency to execute.

- The third variant has (partially) scheduled code, where the code has been shuffled around to move code from time critical parts to non time critical parts. This needs a lower frequency and can execute at lower power.

The compiler is not timing aware, and will undo some of the scheduling performed in the third version, in order to reduce register pressure; this increases power consumption. We estimate that a properly scheduled version (that can be written in assembly but that is completely illegible) will require less than half the clock frequency.

Power consumption figures are:

- 163 mW for the sequential version at 350 MHz; down to 144 mW with scaling to 0.95 V,

- 117 mW for the parallel version at 75 MHz; down to 59 mW with scaling to 0.70 V

- 148 mW for the sequential version at 250 MHz; down to 105 mW with scaling to 0.85 V

We estimate that a properly scheduled version would run at 125 MHz requiring 70-120 mW depending on voltage scaling. This would be the most economical version.

These are all measured on an A16A device powered from 3V3. The figures includes losses incurred in the power supply, and are average measurements of the middle of the run.

2.6 MII send/receive

This benchmark represents a critical software component in constructing an Ethernet enabled XS1 based system. It has hard real-time requirements as well as energy saving opportunities by virtue of different implementation methods and the nature of the protocol it obeys. The intent of this benchmark is to enable characterisation of this component with several degrees of freedom, demonstrating the use of tools developed within the project for design-space exploration.

2.6.1 Introduction to the MII standard

The Media Independent Interface (MII) is used to bridge physical layer signals and digital hardware. MII is used in Ethernet networking, providing an interface between the various physical connection methods such as fiber and twisted copper pairs and the logic responsible for producing and consuming Ethernet packets. Several MII implementations exist to cater to different performance and pinout requirements.

This benchmark implements the digital side of MII for both transmit and receive. It provides several parameters that result in different implementations of the same protocol with different utilisation of the features available in the XS1 architecture. The components can be utilised in different ways as well as configured in various ways. It is intended therefore to be somewhat extensible, albeit with some additional development effort. It is not a production standard MII implementation, as it omits some specification aspects that are not key to or have a large influence on the areas that are being explored by the project.

The 4-bit MII standard is implemented, which operates at 25 MHz for 100 Mbps operation and 2.5 MHz for 10 Mbps mode. The clock signals are generated internally from the XS1 reference clock. Data valid and transmit enable signals are also used alongside the 4-bit data interfaces, as per the IEEE 802.3 standard [IEE95].

Key to the implementation of MII is the ability to produce and consume data from the interface at 4-bits per clock at 25 MHz. If the device is in a low power state upon the arrival of data, then it has approximately 600 nS from the start of an Ethernet preamble to the packet data arriving. Further, there is a minimum gap between Ethernet frames of 960 nS which can potentially be exploited. The benchmark is designed to allow exploration of these, as well as what benefits may arise if these times are made longer or shorter.

Possible component utilisation includes:

- Single or dual-ended usage, including:
  - Transmit and receive implemented on the same tile.
  - Separate transmit and receive onto different tiles.
  - Use of only one side, with the opposite side using a different source or sink than that provided in the benchmark, for example allowing external test generation.
- Loopback, digital or PHY-driven connection, including:
  - Transmit and receive pins connected in loopback within the simulator.
  - Pins physically connected in loopback on test hardware.

- Pins physically connected between multiple chips.
- Pins connected to other hardware for test generation.
- Interface to a PHY peripheral to use a real Ethernet link.

2.6.2 Code overview

The benchmark consists of a module that can be integrated into other applications, and an example application, built for the purpose of evaluating our tools. It provides two threads - one for MII transmit and one for MII receive.

The following compile-time module parameters can be specified, resulting in the generation of different implementations of MII transmit and receive, allowing the trade-off between energy saving techniques, protocol obedience, utilisation, deadlines and quality of service to be examined.

`EARLY_WAKE` The receiver thread will look for the `RXDV` signal being asserted, rather than waiting for the start of frame symbol. This gives more time to change operating state (i.e. increase clock speed) before incoming data must be consumed.

If Active Energy Conservation (AEC) mode is enabled, then fast mode is set in order to prevent the processor from reverting to a slower clock speed in the short potential idle periods between events.

`WAKE_BUSY` Working only in conjunction with `EARLY_WAKE`, this uses polling loops and no-ops to keep the processor awake, rather than fast mode.

`TX_SMALLBUFFER` The default implementation uses the maximum 32-bit buffer size for serialising the output in hardware. With this parameter set, an 8-bit buffer is used, increasing the software overhead but reducing buffer usage.

The following arguments are provided to the transmit function, with the intention of giving several degrees of freedom for the simulation of network traffic and data behaviour.

`txbuf`, `pkt_len` and `pkt_len_len` The `txbuf` is a 512 word buffer used for packet transmission. It is seeded with data before passing it to `mii_tx`. The buffer is then used to transmit packets of the lengths specified in `pkt_len`, cycling to the beginning of the array of lengths every `pkt_len_len` packets. This allows combinations of packets lengths and different traffic

patterns to be generated. If the packet lengths do not coincide with the size of `txbuf`, then the data in the transmitted packets will be unique for a longer period.

`duty_cycle_percent` The transmit thread will delay each transmission in order to achieve a percentage duty cycle roughly approximated by this parameter. A value of 100 will attempt to transmit constantly (save for the protocol enforced inter-frame gap). A value of 25 would result in approximately 25% utilisation.

`num_pkts` How many packets will be transmitted before the test ends. This is useful for time limiting test runs.

In addition to the above, the sample application code, that loops back the transmit and receive components, can have certain properties changed.

`AEC_DIV` If the standard operating frequency of the core is F as defined in the platform description (XN) file, then the Active Energy Conservation (AEC) divider will, when defined, set the clock speed of the core to $\frac{F}{1+\texttt{AEC\_DIV}}$ when all threads are idle or waiting on events and none are set to operate in fast mode.

3 Single version benchmarks

3.1 Image compression

This benchmark is one of the few benchmarks that is written in low level assembly language, rather than a high level language. It is included because it is (a) a good example of a piece of numerical code with the potential to trade-off fidelity against power consumption and (b) it is written in assembly code.

This benchmark could come in useful in two different ways

- If we choose to target (some) of our tools at assembly code, then this benchmark can be used as a target
- We can write a version of this code in a high level language and see if we can obtain code with the same power profile

The latter is difficult as it uses the CP and DP register of the architecture as part of the general purpose register set. As there are no function calls, this is completely legal as long as DP and CP are saved and restored. However, this is an optimisation that is currently not part of the compiler.

3.2 Small numerical benchmarks

This is a collection of small numerical benchmarks that are mostly self explanatory in nature. They have been selected to have a variety of user defined functions, arguments, and calling patterns. Most have a simple

Factorial This comprises a function that computes the factorial of a number in a simple sequential manner.

Fibonacci This comprises a function that computes the factorial of a number using a standard recursive function that is evaluated sequentially.

Power of two This computes a power of two using just additions.

Square This squares a number using just additions.

Power This comprises function that computes X to the power Y using no multiplication.

3.3 Polling loop

The polling loop demo is a program that shows particularly poor behaviour in that it has a polling loop that will consume as much power as is available. It is mainly included for analysis purposes as there is no simple transformation that removes the polling.

The benchmark comprises three processes, one consumer and two producers. The two producers output the values 1..9 over a channel end, asynchronous to each other; they use a timer to make sure that one producer runs slower than the other. The consuming process, called `consumer_polling`, will input samples on either channel-end; and print off any values that come in.

The correct version is also provided for comparison purposes.

3.4 WCET benchmarks

WCET analysis consists of computing the maximum time a program takes to run on a given hardware. Although mainly focussed on computing the worst case time, some WCET tools also infer values of input parameters corresponding to the worst case [WEE+08] while some other tools compute the worst case as a function of the program's input parameters [BEL11, vHHL+11].

Like WCET analysis, energy consumption analysis is a program resource analysis and one can expect that some of the tools and techniques developed for WCET can be adapted for energy consumption analysis. Here we take some of the WCET benchmark programs from [Gus] and consider them as energy consumption benchmark programs. These are simple programs which contain both simple and nested loops, use arrays and matrices, use bit operations, contain self or mutual recursion and so on. The main purpose of these programs in our project is to evaluate and compare different types of energy consumption analysis, simulation and measurement tools and methods.

Binary Search This searches for an element in the given array. The code is structured and contains a single loop. It is intended to check how energy varies when an element is in the array and when not. If an element is in the array, will its position in the array affects the energy consumption?

FibCall This computes the fibonacci of a given number. It contains a single loop and the execution of the loop depends on the parameter. It is intended to check the energy variation depending on the input parameter and the ability of the static analysis tool to express loop as a function of the program's input parameter.

Cover This is a single loop program whose loop contains many switches. It is intended to check how switching among different instructions affects energy consumption.

Exp-Int This computes an exponential integral function by series expansion. It contains nested loops and the inner loop runs only once (general WCET analysis gives overestimate, to check whether energy computing tools also do so). It is Intended to improve structural program analysis to get correct estimate of energy consumption.

Janne-Complex This is a nested loop program where the number of iterations of inner loop depends on the current iteration number of the outer loop. It Intended to improve structural program analysis (consider dependency) to get correct estimate of energy consumption.

Ndes This is a complex embedded code which involves bit manipulation, shifts, array and matrix calculation. It is intended to compute energy consumption when a program involves array, bit operations.

Recursion This is a simple example of recursive code which contains self recursion and mutual recursion . It is Intended to analyze methods which can handle recursion and how to compute energy in such case.

4 Summary

In this deliverable we have presented a number of benchmarks: small sections of code that can be used to develop and understand analysis and transformation tools.

Six benchmarks have been written in different styles (three versions each), in order to demonstrate the effects of parallellising code, sequentialising code, analysing the relationship between number of coefficients, precision of coefficients and power, frequency scaling, and voltage scaling. One of these benchmarks is extracted from actual code form an XMOS customer; all other have been extracted from real code that is in use on XMOS devices.

Additionally, 14 small benchmarks have been provided that are simpler in nature, and that can be used to understand specific parts of the code.

We do not expect all 29 benchmarks to be used exhaustively; they have been selected to enable a wide variety of problems to be researched, for example parallelisation of code to run it slow and wide, or analysing the relationship between input parameters and energy consumed.

References

[BEL11] Stefan Bygde, Andreas Ermedahl, and Björn Lisper. An efficient algorithm for parametric WCET calculation. *Journal of Systems Architecture - Embedded Systems Design*, 57(6):614–624, 2011.

[BJ] Robert Bristow-Johnson. Cookbook formulae for audio EQ biquad filter coefficients, at http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt.

[Gus] Jan Gustafsson. WCET benchmark programs ONLINE@http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[IEE95] IEEE standards for local and metropolitan area networks: Supplement to carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications media access control (MAC) parameters, physical layer, medium attachment units, and repeater for 100 mb/s operation, type 100BASE-T (clauses 21-30). *IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1993 Edition)*, pages 1–398, 1995.

[vHHL+11] Reinhard von Hanxleden, Niklas Holsti, Björn Lisper, Erhard Ploedereder, Armelle Bonenfant, Hugues Cassé, Sven Bünte, Wolfgang Fellger, Sebastian Gepperth, Jan Gustafsson, Benedikt Huber, Nazrul Mohammad Islam, Daniel Kästner, Raimund Kirner, Laura Kovacs, Felix Krause, Marianne de Michiel, Mads Christian Olesen, Adrian Prantl, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Simon Wegener, Michael Zolda, and Jakob Zwirchmayr. WCET tool challenge 2011: Report. In Chris Healy, editor, *Proc. 11th International Workshop on Worst-Case Execution Time (WCET) Analysis (WCET 2011)*. OCG, July 2011.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

A Biquad

This benchmark is stored in `wps/wp5/biquad`, a sequential version is stored in the `seq` directory, a parallel version in the `par` and `partile` directories. A common set of input signals and coefficients is stored in `seq/samples.h`

The sequential version:

```
#include "main.h"
#include "biquad.h"
#include <xs1.h>
#include <stdio.h>

void initBiquads(biquadState &state) {
    for(int i = 0; i <= BANKS; i++) {
        state.b[i].xn1 = 0;
        state.b[i].xn2 = 0;
    }
}

int biquadCascade(biquadState &state, int xn) {
    unsigned int ynl;
    int ynh;

    for(int j=0; j<BANKS; j++) {
        ynl = (1<<(FRACTIONALBITS-1));
        ynh = 0;
        {ynh, ynl} = macs( biquads[j].b0, xn, ynh, ynl);
        {ynh, ynl} = macs( biquads[j].b1, state.b[j].xn1, ynh, ynl);
        {ynh, ynl} = macs( biquads[j].b2, state.b[j].xn2, ynh, ynl);
        {ynh, ynl} = macs( biquads[j].a1, state.b[j+1].xn1, ynh, ynl);
        {ynh, ynl} = macs( biquads[j].a2, state.b[j+1].xn2, ynh, ynl);
        if (sext(ynh,FRACTIONALBITS) == ynh) {
            ynh = (ynh << (32-FRACTIONALBITS)) | (ynl >> FRACTIONALBITS);
        } else if (ynh < 0) {
            ynh = 0x80000000;
        } else {
            ynh = 0x7fffffff;
        }
        state.b[j].xn2 = state.b[j].xn1;
        state.b[j].xn1 = xn;

        xn = ynh;
    }
    state.b[BANKS].xn2 = state.b[BANKS].xn1;
    state.b[BANKS].xn1 = ynh;
    return xn;
}
```

```
#include "main.h"
#include "biquad.h"
#include "voltage.h"
#include <stdio.h>
#include <xs1.h>
#include <platform.h>

#include "samples.h"


void save_power() {
    write_sswitch_reg_no_ack(0x8001, 7, 10);
    write_sswitch_reg(0x8003, 7, 10);
    write_pswitch_reg(0x8003, 6, 10);
    setps(0x010B, 0x10);
    set_voltage(950);
}
```

```
void main0(void) {
    biquadState bs;
    int start, end, o;
    timer tmr;
    initBiquads(bs);
    tmr :> start;
    for(int j = 0; j < ITERATIONS; j++) {
        for(int i = 0; i < SAMPLES; i++) {
            o = biquadCascade(bs, signal[i]);
        }
    }
    tmr :> end;
    printf("%9d    %9d\n", o, end - start);
}

int main() {
    par {
        on tile[0]: main0();
        on tile[1]: save_power();
    }
    return 0;
}
```

The parallel lower-power code version:

```
#include "main.h"
#include "biquad.h"
#include <xs1.h>
#include <stdio.h>

void initBiquads(biquadState &state) {
    for(int i = 0; i <= BANKS; i++) {
        state.b[i].xn1 = 0;
        state.b[i].xn2 = 0;
    }
}

int biquadCascade(const struct coeff &biquad, biquadState &state, int xn) {
    unsigned int ynl;
    int ynh;

    ynl = (1<<(FRACTIONALBITS-1));
    ynh = 0;
    {ynh, ynl} = macs( biquad.b0, xn, ynh, ynl);
    {ynh, ynl} = macs( biquad.b1, state.b[0].xn1, ynh, ynl);
    {ynh, ynl} = macs( biquad.b2, state.b[0].xn2, ynh, ynl);
    {ynh, ynl} = macs( biquad.a1, state.b[1].xn1, ynh, ynl);
    {ynh, ynl} = macs( biquad.a2, state.b[1].xn2, ynh, ynl);
    if (sext(ynh,FRACTIONALBITS) == ynh) {
        ynh = (ynh << (32-FRACTIONALBITS)) | (ynl >> FRACTIONALBITS);
    } else if (ynh < 0) {
        ynh = 0x80000000;
    } else {
        ynh = 0x7fffffff;
    }
    state.b[0].xn2 = state.b[0].xn1;
    state.b[0].xn1 = xn;

    state.b[BANKS].xn2 = state.b[BANKS].xn1;
    state.b[BANKS].xn1 = ynh;
    return ynh;
}
```

```
#include "main.h"
#include "biquad.h"
#include "voltage.h"
#include <stdio.h>
#include <platform.h>
```

```
#include "../seq/samples.h"

void one_biquad(int n, chanend ?inp, chanend? outp) {
    biquadState bs;
    int start, end, o;
    timer tmr;
    initBiquads(bs);
    tmr :> start;
    for(int j = 0; j < ITERATIONS; j++) {
        for(int i = 0; i < SAMPLES; i++) {
            int in_signal;
            if (isnull(inp)) {
                in_signal = signal[i];
            } else {
            inp :> in_signal;
            }
            o = biquadCascade(biquads[n], bs, in_signal);
            if (!isnull(outp)) {
                outp <: o;
            }
        }
    }
    tmr :> end;
    if (isnull(outp)) {
        printf("%9d    %9d\n", o, end - start);
    }
}

void save_power() {
    set_voltage(750);
    write_sswitch_reg_no_ack(0x8001, 7, 10);
    write_sswitch_reg(0x8003, 7, 10);
    write_pswitch_reg(0x8003, 6, 10);
    setps(0x010B, 0x10);
}

int main(void) {
    chan a, b, c, d, e, f;
    par {
        on tile[0]: one_biquad(0, null, a);
        on tile[0]: one_biquad(1, a, b);
        on tile[0]: one_biquad(2, b, c);
        on tile[0]: one_biquad(3, c, d);
        on tile[0]: one_biquad(4, d, e);
        on tile[0]: one_biquad(5, e, f);
        on tile[0]: one_biquad(6, f, null);
        on tile[1]: save_power();
    }
    return 0;
}
```

The common input settings:

```
/* Standardised set of coefficients for testing */

struct coeff biquads[7] = {
    {16967096, -33375710, 16417658, 33379780, -16603468},
    {16994608, -33494641, 16511515, 33494641, -16728907},
    {17211375, -33412099, 16246577, 33412099, -16680736},
    {17643057, -33179262, 15718966, 33179262, -16584807},
    {15215588, -29426736, 14868559, 29426736, -13306931},
    {13946172, -24906194, 13317051, 24906194, -10486008},
    {3252441, -883755, 600579, 22199243, -8391292},
};

/* Standardised set of input samples for testing */

#define SAMPLES 192

int signal[SAMPLES] = {
```

```
    0,          19969040,   25107969,   15136355,
    4342263,    7068910,    23197578,   38155414,
    37447598,   21184205,   4072434,    874754,
    11863283,   22801011,   19451147,   2086462,
    -14529495,  -15688013,  -1277089,   14210800,
    16205546,   4591563,    -6284418,   -2130463,
    16777215,   35613052,   39551787,   28318129,
    16205546,   17562902,   32277342,   45782069,
    43588485,   25813028,   7169373,    2426546,
    11863283,   21249219,   16354209,   -2542361,
    -20670382,  -23314668,  -10356853,  3716808,
    4342263,    -8590210,   -20728237,  -17774475,
    0,          17774475,   20728237,   8590210,
    -4342263,   -3716808,   10356853,   23314668,
    20670382,   2542361,    -16354209,  -21249219,
    -11863283,  -2426546,   -7169373,   -25813028,
    -43588485,  -45782069,  -32277342,  -17562902,
    -16205546,  -28318129,  -39551787,  -35613053,
    -16777216,  2130463,    6284418,    -4591563,
    -16205546,  -14210800,  1277089,    15688013,
    14529495,   -2086462,   -19451147,  -22801011,
    -11863283,  -874754,    -4072434,   -21184204,
    -37447598,  -38155414,  -23197578,  -7068910,
    -4342263,   -15136355,  -25107969,  -19969040,
    0,          19969040,   25107969,   15136355,
    4342263,    7068910,    23197578,   38155414,
    37447598,   21184205,   4072434,    874754,
    11863283,   22801011,   19451147,   2086462,
    -14529495,  -15688013,  -1277089,   14210800,
    16205546,   4591563,    -6284418,   -2130463,
    16777215,   35613052,   39551787,   28318129,
    16205546,   17562902,   32277342,   45782069,
    43588485,   25813028,   7169373,    2426546,
    11863283,   21249219,   16354209,   -2542361,
    -20670382,  -23314668,  -10356853,  3716808,
    4342263,    -8590210,   -20728237,  -17774475,
    0,          17774475,   20728237,   8590210,
    -4342262,   -3716808,   10356853,   23314668,
    20670382,   2542361,    -16354209,  -21249219,
    -11863283,  -2426546,   -7169373,   -25813028,
    -43588485,  -45782069,  -32277342,  -17562902,
    -16205546,  -28318129,  -39551787,  -35613053,
    -16777216,  2130463,    6284418,    -4591563,
    -16205546,  -14210800,  1277089,    15688013,
    14529495,   -2086461,   -19451147,  -22801011,
    -11863283,  -874754,    -4072434,   -21184204,
    -37447598,  -38155414,  -23197578,  -7068910,
    -4342262,   -15136355,  -25107969,  -19969040
};

#define ITERATIONS 10000
```

The version that runs on multiple tiles is not included: it just forks three of the tasks on a `tile[1]` rather than `tile[0]`.

B FIR

This benchmark is stored in `wps/wp5/fir/seq`, with a parallel version in the `par` directory. The low fidelity version is created using a compile time flag on the `par` version.

The code comprises two files. The code of the benchmark, and the code of the calling function. The standard input filter and input signal are stored in a separate file.

The sequential version:

```
#include "fir.h"
#include <xs1.h>

int fir(int xn, int coeffs[], int state[], int ELEMENTS) {
    unsigned int ynl;
    int ynh;
    ynl = (1<<23);
    ynh = 0;
    for(int j=ELEMENTS-1; j!=0; j--) {
        state[j] = state[j-1];
        {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
    }
    state[0] = xn;
    {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);

    if (sext(ynh,24) == ynh) {
        ynh = (ynh << 8) | (((unsigned) ynl) >> 24);
    } else if (ynh < 0) {
        ynh = 0x80000000;
    } else {
        ynh = 0x7fffffff;
    }
    return ynh;
}
```

```
#include "fir.h"
#include "voltage.h"
#include <xs1.h>
#include <platform.h>
#include <stdio.h>

#include "samples.h"

void main0(void) {
    int state[TAPS];
    int o, start, end;
    timer tmr;
    for(int i = 0; i < TAPS; i++) {
        state[i] = 0;
    }
#define LOWPREC
#ifdef LOWPREC
    for(int i = 0; i < SAMPLES; i++) {
        signal[i] &= 0xFFFF0000;
    }
    for(int i = 0; i < TAPS; i++) {
        coeffs[i] &= 0xFFFF0000;
    }
#endif
tmr :> start;
    for(int j = 0; j < ITERATIONS; j++) {
    for(int i = 0; i < SAMPLES; i++) {
        o = fir(signal[i], coeffs, state, TAPS);
    }
    }
tmr :> end;
```

```
    printf("%9d  %9d\n", o, end - start);
}

void save_power() {
//    set_voltage(950);
    write_sswitch_reg_no_ack(0x8001, 7, 10);
    write_sswitch_reg(0x8003, 7, 10);
    write_pswitch_reg(0x8003, 6, 10);
    setps(0x010B, 0x10);
}

int main(void) {
    par {
        on tile[0]: main0();
        on tile[1]: save_power();
    }
    return 0;
}
```

The parallel version with optional low-precision setting:

```
#include "fir.h"
#include <xs1.h>

{int,int,int} fir(int xn, const int coeffs[], int state[], int ELEMENTS, int ynh, unsigned int ynl) {
    int o = state[ELEMENTS-1];
//    ynl = (1<<23);
    for(int j=ELEMENTS-1; j!=0; j--) {
        state[j] = state[j-1];
        {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
    }
    state[0] = xn;
    {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);

    return {o, ynh, ynl};
}
```

```
#include "fir.h"
#include "voltage.h"
#include <xs1.h>
#include <platform.h>
#include <stdio.h>

#include "../seq/samples.h"

void main0(int n, streaming chanend ?inp, streaming chanend ?outp) {
    int state[TAPS];
    int o, start, end;
    int sh, sl, ynh, input_signal;
    unsigned ynl;
    timer tmr;
    for(int i = 0; i < TAPS; i++) {
        state[i] = 0;
    }
    tmr :> start;
    for(int j = 0; j < ITERATIONS; j++) {
    for(int i = 0; i < SAMPLES; i++) {
        if (isnull(inp)) {
            input_signal = signal[i];// & 0xfff00000;
            sh = 0;
            sl = 1<<23;
        } else {
            inp :> input_signal;
            inp :> sh;
            inp :> sl;
        }

        {o,ynh,ynl} = fir(input_signal, &coeffs[18*n], state, 18, sh, sl);
```

```
        if (isnull(outp)) {
            if (sext(ynh,24) == ynh) {
                ynh = (ynh << 8) | (((unsigned) ynl) >> 24);
            } else if (ynh < 0) {
                ynh = 0x80000000;
            } else {
                ynh = 0x7fffffff;
            }
            o = ynh;
        } else {
            outp <: o;
            outp <: ynh;
            outp <: ynl;
        }

    }
    }
    tmr :> end;
    if (isnull(outp)) {
        printf("%9d  %9d\n", o, end - start);
    }
}

void save_power() {
    //set_voltage(750);
    write_sswitch_reg_no_ack(0x8001, 7, 10);
    write_sswitch_reg(0x8003, 7, 10);
    write_pswitch_reg(0x8003, 6, 10);
    setps(0x010B, 0x10);
}

int main(void) {
    streaming chan a, b, c, d, e, f;
    par {
        on tile[0]: main0(0, null, a);
        on tile[0]: main0(1, a, b);
        on tile[0]: main0(2, b, c);
        on tile[0]: main0(3, c, d);
        on tile[0]: main0(4, d, e);
        on tile[0]: main0(5, e, f);
        on tile[0]: main0(6, f, null);
        on tile[1]: save_power();
    }
    return 0;
}
```

The common input settings:

```
#define TAPS 121

#define ITERATIONS 300

int coeffs[TAPS+5] = {
    0,       -6320,   -6580,    0,  7437,      8045,
    0,       -9647,   -10652,   0,  13100,     14555,
    0,       -17960,  -19923,   0,  24407,     26944,
    0,       -32652,  -35843,   0,  42951,     46893,
    0,       -55623,  -60445,   0,  71096,     76971,
    0,       -89958,  -97136,   0,  113068,    121922,
    0,       -141732, -152852,  0,  178058,    192428,
    0,       -225664, -245061,  0,  291317,    319290,
    0,       -389242, -433986,  0,  555119,    640511,
    0,       -910478, -1144598, 0,  2306614,   4621966,
    5592405, 4621966, 2306614,  0, -1144598,  -910478,
    0,        640511,  555119,  0, -433986,   -389242,
    0,        319290,  291317,  0, -245061,   -225664,
    0,        192428,  178058,  0, -152852,   -141732,
    0,        121922,  113068,  0, -97136,    -89958,
    0,        76971,   71096,   0, -60445,    -55623,
    0,        46893,   42951,   0, -35843,    -32652,
    0,        26944,   24407,   0, -19923,    -17960,
```

```
    0,        14555,   13100,   0, -10652,    -9647,
    0,        8045,    7437,    0, -6580,     -6320,
    0, 0, 0, 0, 0, 0 };

#define SAMPLES 192

int signal[SAMPLES] = {
    0,         18871758,  22918103,  11863283,
    0,         1676050,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863283,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863283, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863283,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863283, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863283,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863283, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863283,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863283, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863282,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863282, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863282,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863282, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187232,
    0,         10187232,  6140887,   -11863282,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863282, -22918103, -18871758,
    0,         18871758,  22918103,  11863283,
    0,         1676051,   16777215,  30735041,
    29058990,  11863283,  -6140887,  -10187231,
    0,         10187232,  6140887,   -11863282,
    -29058990, -30735041, -16777216, -1676050,
    0,         -11863282, -22918103, -18871758,
};
```

C I2S

This benchmark is stored in `wps/wp5/i2s/seq`, `wps/wp5/i2s/par`, and `wps/wp5/i2s/par2`.

Each directory contains two files. The code of the benchmark, and the code of the calling function. The calling function is identical for all three and listed only once.

```
#include "i2s.h"

#include <xs1.h>
#include <stdio.h>

buffered out port:32 LRCLK = XS1_PORT_1A;
buffered out port:32 BCLK = XS1_PORT_1B;
buffered out port:32 dac = XS1_PORT_1C;
buffered in  port:32 adc = XS1_PORT_1D;
port MCLK = XS1_PORT_1E;

clock mclk = XS1_CLKBLK_1;
clock bclk = XS1_CLKBLK_2;

void i2s_setup() {
    configure_clock_ref(mclk, 2);
    configure_out_port_no_ready(BCLK, mclk, 0);
    start_clock(mclk);
    configure_clock_src(bclk, BCLK);
    configure_out_port_no_ready(LRCLK, bclk, 0);
    configure_out_port_no_ready(dac, bclk, 0);
    configure_in_port_no_ready(adc, bclk);
    start_clock(bclk);
    BCLK <: 0xAAAAAAAA;
    sync(BCLK);
    clearbuf(dac);
    clearbuf(adc);
    clearbuf(LRCLK);
    LRCLK <: ~0; dac <: 0;  // preload.
    BCLK <: 0xFFFF0000;
    sync(BCLK);
    LRCLK <: 0; dac <: 0;  // preload.
}

static void generateBCLK16() {
#pragma loop unroll
    for(int i = 0; i < 4; i++) { BCLK <: 0xF0F0F0F0; }
}

void i2s_loop(chanend adc_data, chanend dac_data) {
    unsigned l, r;
    unsigned dl, dr;
    while(1) {
        generateBCLK16();
        adc_data <: l;
        adc_data <: r;
        dac_data :> dl;
        dac_data :> dr;
        generateBCLK16();
        LRCLK <: ~0;
        adc :> l;
        dac <: dl;
        generateBCLK16();
        generateBCLK16();
        LRCLK <: 0;
        adc :> r;
        dac <: dr;
    }
}
```

```
#include "i2s.h"

#include <xs1.h>

buffered out port:32 LRCLK = XS1_PORT_1A;
buffered out port:32 BCLK = XS1_PORT_1B;
buffered out port:32 dac = XS1_PORT_1C;
buffered in  port:32 adc = XS1_PORT_1D;
port MCLK = XS1_PORT_1E;

clock mclk = XS1_CLKBLK_1;
clock bclk = XS1_CLKBLK_2;

void i2s_setup() {
    configure_clock_ref(mclk, 2);
    configure_out_port_no_ready(BCLK, mclk, 0);
    start_clock(mclk);
    configure_clock_src(bclk, BCLK);
    configure_out_port_no_ready(LRCLK, bclk, 0);
    configure_out_port_no_ready(dac, bclk, 0);
    configure_in_port_no_ready(adc, bclk);
    start_clock(bclk);
}

void generateLRCLK() {
    LRCLK <: 0; LRCLK <: ~0;
}

static void generateBCLK() {
    for(int i = 0; i < 16; i++) { BCLK <: 0xF0F0F0F0; }
}

static void outputWords(unsigned l, unsigned r) {
    dac <: l; dac <: r;
}

static void inputWords(unsigned &l, unsigned &r) {
    adc :> r; adc :> l;
}

static void bufferAdc(chanend c, unsigned l, unsigned r) {
    c <: l; c <: r;
}

static void bufferDac(chanend c, unsigned &l, unsigned &r) {
    c :> l; c :> r;
}

void i2s_loop(chanend adc_data, chanend dac_data) {
    unsigned dacL, dacR, newDacL, newDacR;
    unsigned dacLDsp, dacRDsp, newDacLDsp, newDacRDsp;
    unsigned adcL, adcR, newAdcL, newAdcR;
    while(1) {
        par {
            generateLRCLK();
            generateBCLK();
            outputWords(dacLDsp, dacRDsp);
            inputWords(newAdcL, newAdcR);
            bufferAdc(adc_data, adcL, adcR);
            bufferDac(dac_data, dacL, dacR);
        }
        par {
            dacL = newDacL;
            dacR = newDacR;
            dacLDsp = newDacLDsp;
            dacRDsp = newDacRDsp;
            adcL = newAdcL;
            adcR = newAdcR;
        }
    }
}
```

```
#include "i2s.h"

#include <xs1.h>
#include <stdio.h>

buffered out port:32 LRCLK = XS1_PORT_1A;
buffered out port:32 BCLK = XS1_PORT_1B;
buffered out port:32 dac = XS1_PORT_1C;
buffered in  port:32 adc = XS1_PORT_1D;
port MCLK = XS1_PORT_1E;

clock mclk = XS1_CLKBLK_1;
clock bclk = XS1_CLKBLK_2;

void i2s_setup() {
    configure_clock_ref(mclk, 2);
    configure_out_port_no_ready(BCLK, mclk, 0);
    start_clock(mclk);
    configure_clock_src(bclk, BCLK);
    configure_out_port_no_ready(LRCLK, bclk, 0);
    configure_out_port_no_ready(dac, bclk, 0);
    configure_in_port_no_ready(adc, bclk);
    start_clock(bclk);
    BCLK <: 0xAAAAAAAA;
    sync(BCLK);
    clearbuf(dac);
    clearbuf(adc);
    clearbuf(LRCLK);
    LRCLK <: ~0; dac <: 0;  // preload.
    BCLK <: 0xFFFF0000;
    sync(BCLK);
    LRCLK <: 0; dac <: 0;  // preload.
}

void generateLRCLK(streaming chanend ton) {
    while(1) {
        LRCLK <: ~0;
        ton <: 0;
        LRCLK <: 0;
    }
}

static void generateBCLK(streaming chanend from) {
    while(1) {
#pragma loop unroll
        for(int i = 0; i < 16; i++) { BCLK <: 0xF0F0F0F0; }
        from :> int _;
    }
}

static void outputWords(chanend from, streaming chanend si, streaming chanend so) {
    unsigned l, r;
    while(1) {
        from :> l;
        si :> int _;
        so <: 0;
        from :> r;
        dac <: l; dac <: r;
    }
}

static void inputWords(chanend ton, streaming chanend si, streaming chanend so) {
    unsigned l, r;
    while(1) {
        adc :> l;
        si :> int _;
        so <: 0;
        adc :> r;
        ton <: l;
        ton <: r;
    }
}
```

```
void i2s_loop(chanend adc_data, chanend dac_data) {
    streaming chan todac, toadc, toblck;
    par {
        generateLRCLK(todac);
        outputWords(dac_data, todac, toadc);
        inputWords(adc_data, toadc, toblck);
        generateBCLK(toblck);
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "i2s.h"

void produce(chanend c) {
    while(1) {
        c <: 1;
        c <: 2;
    }
}

void consume(chanend c) {
    timer t;
    unsigned int last_time;
    unsigned int now;
    int first = 1;
    for(int i = 0; i < 10; i++) {
        c :> int _;
        t :> now;
        c :> int _;
        if (!first) {
            unsigned diff = (now - last_time);
            if (diff > 2050 || diff < 2046) {
                printf("Diff %d\n", diff);
            }
        }
        first = 0;
        last_time = now;
    }
    exit(0);
}


int main(void) {
    chan adc, dac;
    i2s_setup();
    par {
        produce(dac);
        consume(adc);
        i2s_loop(adc, dac);
    }
    return 0;
}
```

D ADC to DAC

This benchmark is stored in `wps/wp5/adc/par`, `wps/wp5/adc/seq` and `wps/wp5/adc/seq_schedul`

Each directory contains three files. The code of the benchmark, the supporting `sin` function (computing sin(x) where 0 ¡= x ¡ 384 represents a number between 0 and 2π) and the main calling function. The latter two are identical for all versions and listed only once.

```
#include "adc.h"
#include "sine_384.h"
#include <xs1.h>
#include <stdio.h>
#include <platform.h>

on tile[0]: buffered out port:32 adc_sample = XS1_PORT_32A;
on tile[0]: buffered out port:1 pwm_plus   = XS1_PORT_1J;
on tile[0]: buffered out port:1 pwm_minus  = XS1_PORT_1K;

#define PWM_CYCLE        (260)
#define ADC_HALF_CYCLE   (PWM_CYCLE/4)


int adc_init(chanend c) {
    int t;
    unsigned int data[1];
    /* Disable adc */
    data[0] = 0x0;
    write_periph_32(adc_tile, 2, 0x20, 1, data);

    /* Enable adc on channel */
    asm("add %0,%1,0":"=r"(data[0]):"r"(c));
    data[0]--;
    write_periph_32(adc_tile, 2, 0x0, 1, data);
    write_periph_32(adc_tile, 2, 0x4, 1, data);

    /* General ADC control (enabled, 2 samples per packet, 16 bits per sample) */
    data[0] = 0x10201;
    write_periph_32(adc_tile, 2, 0x20, 1, data);
    adc_sample <: 0 @ t;
    for(int i = 0; i < 8; i++) {
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: 0;
    }
    return t;
}

static void adc_clock(int t) {
    while(1) {
        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint1"
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: 0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint0"
        adc_sample @ t <: 0;
    }
}

static void adc_process(chanend todsp, chanend adc_half) {
    int sample, signal;

    while(1) {
        sample = inuint(adc_half);
```

```
        chkct(adc_half, 1);
        signal = (sample >> 16) & 0xffff;
        todsp <: signal;
    }
}

static void dsp_process(chanend fromin, chanend toout) {
    int sineIndex = 0, sine, heterodyne_value;
    int mixing_frequency_khz = 50, signal = 32768;
    while(1) {
#pragma xta endpoint "endpoint2"
        fromin :> signal;
        sineIndex += mixing_frequency_khz;
        if (sineIndex >= 384) {
            sineIndex -= 384;
        }
        sine = sine_384(sineIndex);
        heterodyne_value = ((signal - 32768) * (long long) sine) >> 32;
        toout <: heterodyne_value;
    }
}

static void pwm_process(int t, chanend fromdsp) {
    int error = 0, output, output_value;
    pwm_minus <: 0;
    pwm_plus <: 0;
    while(1) {
        fromdsp :> output_value;
        t += PWM_CYCLE;
        error += output_value + (128 << 8);
        output = error >> 8;
        error -= output << 8;
        pwm_plus @ t <: 1;
        pwm_minus @ t <: 0;
        pwm_plus @ (t + output) <: 0;
        pwm_minus @ (t + output) <: 1;
//        printf("Out: %d\n", output);
    }
}


void adc_task(chanend adc_half) {
    chan intodsp, dsptoout;
    int t = adc_init(adc_half);
    par {
        adc_clock(t);
        adc_process(intodsp, adc_half);
        dsp_process(intodsp, dsptoout);
        pwm_process(t, dsptoout);
    }
}

//#pragma xta command "help config"
#pragma xta command "config freq 0 500"
#pragma xta command "config tasks tile[0] 4"

#pragma xta command "analyze endpoints endpoint0 endpoint1"
#pragma xta command "set required - 650.0 ns"

#pragma xta command "analyze endpoints endpoint2 endpoint2"
#pragma xta command "set required - 2600.0 ns"
```

```
#include "adc.h"
#include "sine_384.h"
#include <xs1.h>
#include <stdio.h>
#include <platform.h>

on tile[0]: buffered out port:32 adc_sample = XS1_PORT_32A;
on tile[0]: buffered out port:1 pwm_plus   = XS1_PORT_1J;
on tile[0]: buffered out port:1 pwm_minus  = XS1_PORT_1K;
```

```
#define PWM_CYCLE        (260)
#define ADC_HALF_CYCLE   (PWM_CYCLE/4)

int adc_init(chanend c) {
    int t;
    int r;
    unsigned int data[1];
    /* Disable adc */
    data[0] = 0x0;
    r = write_periph_32(adc_tile, 2, 0x20, 1, data);

    /* Enable adc on channel */
    asm("add %0,%1,0":"=r"(data[0]):"r"(c));
    data[0]--;
    write_periph_32(adc_tile, 2, 0x0, 1, data);
    r = write_periph_32(adc_tile, 2, 0x4, 1, data);

    /* General ADC control (enabled, 2 samples per packet, 16 bits per sample) */
    data[0] = 0x10201;
    r = write_periph_32(adc_tile, 2, 0x20, 1, data);
    adc_sample <: 0 @ t;
    for(int i = 0; i < 8; i++) {
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: 0;
    }
    return t;
}


void adc_task(chanend adc_half) {
    int t;
    int sineIndex = 0;
    int sample;
    int mixing_frequency_khz = 50, signal = 32768;
    int error = 0;
    int sine, output_value, output = 20;

    pwm_minus <: 0;
    t = adc_init(adc_half);
    while(1) {
        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint1"
        adc_sample @ t <: ~0;
        pwm_plus @ t <: 1;
        pwm_minus @ t <: 0;
        pwm_plus @ (t + output) <: 0;
        pwm_minus @ (t + output) <: 1;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: 0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint0"
        adc_sample @ t <: 0;

        sample = inuint(adc_half);
        chkct(adc_half, 1);
        signal = (sample >> 16) & 0xffff;

        sineIndex += mixing_frequency_khz;
        if (sineIndex >= 384) {
            sineIndex -= 384;
        }
        sine = sine_384(sineIndex);
        output_value = ((signal - 32768) * (long long) sine) >> 32;

        error += output_value + (128 << 8);
        output = error >> 8;
        error -= output << 8;
    }
```

```
}

//#pragma xta command "help config"
#pragma xta command "config freq 0 500"
#pragma xta command "config tasks tile[0] 4"

//#pragma xta command "analyze endpoints endpoint0 endpoint1"
//#pragma xta command "set required - 650.0 ns"
```

```
#include "adc.h"
#include "sine_384.h"
#include <xs1.h>
#include <platform.h>

on tile[0]: buffered out port:32 adc_sample = XS1_PORT_32A;
on tile[0]: buffered out port:1 pwm_plus   = XS1_PORT_1J;
on tile[0]: buffered out port:1 pwm_minus  = XS1_PORT_1K;

#define PWM_CYCLE        (260)
#define ADC_HALF_CYCLE   (PWM_CYCLE/4)

int adc_init(chanend c) {
    int t;
    unsigned int data[1];
    /* Disable adc */
    data[0] = 0x0;
    write_periph_32(adc_tile, 2, 0x20, 1, data);

    /* Enable adc on channel */
    asm("add %0,%1,0":"=r"(data[0]):"r"(c));
    data[0]--;
    write_periph_32(adc_tile, 2, 0x0, 1, data);
    write_periph_32(adc_tile, 2, 0x4, 1, data);

    /* General ADC control (enabled, 2 samples per packet, 16 bits per sample) */
    data[0] = 0x10201;
    write_periph_32(adc_tile, 2, 0x20, 1, data);
    adc_sample <: 0 @ t;
    for(int i = 0; i < 8; i++) {
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        t += ADC_HALF_CYCLE;
        adc_sample @ t <: 0;
    }
    return t;
}


void adc_task(chanend adc_half) {
    int t;
    int sineIndex = 0;
    int sample;
    int mixing_frequency_khz = 50, signal = 32768;
    int error = 0;
    int sine, output_value, output = 20;

    pwm_minus <: 0;
    t = adc_init(adc_half);
    while(1) {

        t += ADC_HALF_CYCLE;
        adc_sample @ t <: ~0;
        pwm_plus @ t <: 1;
#pragma xta endpoint "endpoint1"
        pwm_minus @ t <: 0;
        pwm_plus @ (t + output) <: 0;
        pwm_minus @ (t + output) <: 1;


        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint2"
```

```
        adc_sample @ t <: 0;

        sample = inuint(adc_half);
        chkct(adc_half, 1);
        signal = (sample >> 16) & 0xffff;

        sineIndex += mixing_frequency_khz;
        if (sineIndex >= 384) {
            sineIndex -= 384;
        }

        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint3"
        adc_sample @ t <: ~0;

        sine = sine_384(sineIndex);
        output_value = ((signal - 32768) * (long long) sine) >> 32;


        t += ADC_HALF_CYCLE;
#pragma xta endpoint "endpoint0"
        adc_sample @ t <: 0;

        error += output_value + (128 << 8);
        output = error >> 8;
        error -= output << 8;

    }
}

//#pragma xta command "help config"
#pragma xta command "config freq 0 500"
#pragma xta command "config tasks tile[0] 4"

#pragma xta command "analyze endpoints endpoint0 endpoint1"
#pragma xta command "set required - 650.0 ns"

#pragma xta command "analyze endpoints endpoint1 endpoint2"
#pragma xta command "set required - 650.0 ns"

#pragma xta command "analyze endpoints endpoint2 endpoint3"
#pragma xta command "set required - 650.0 ns"

#pragma xta command "analyze endpoints endpoint3 endpoint0"
#pragma xta command "set required - 650.0 ns"
```

```
#include <xs1.h>
#include <platform.h>
#include "voltage.h"
#include <stdio.h>
#include <stdlib.h>
#include "adc.h"


void save_power() {
    timer tmr;
    unsigned t;

    write_sswitch_reg_no_ack(0x8001, 7, 10);
    write_sswitch_reg(0x8003, 7, 10);
    write_pswitch_reg(0x8003, 6, 10);
    setps(0x010B, 0x10);
    set_voltage(700);

    tmr :> t;
    tmr when timerafter(t+5000000) :> void;
    exit(0);
}

int main(void) {
    chan adc;
```

```
    par {
        on tile[0]: adc_task(adc);
        on tile[1]: save_power();
    }
    return 0;
}
```

```
int sine_lookup[97] = {
    0,
    35136551,
    70263695,
    105372028,
    140452150,
    175494670,
    210490206,
    245429388,
    280302863,
    315101294,
    349815365,
    384435782,
    418953276,
    453358606,
    487642561,
    521795963,
    555809667,
    589674567,
    623381597,
    656921733,
    690285995,
    723465451,
    756451217,
    789234464,
    821806413,
    854158345,
    886281597,
    918167571,
    949807729,
    981193601,
    1012316784,
    1043168944,
    1073741823,
    1104027236,
    1134017074,
    1163703308,
    1193077990,
    1222133257,
    1250861329,
    1279254515,
    1307305214,
    1335005915,
    1362349204,
    1389327758,
    1415934356,
    1442161874,
    1468003290,
    1493451686,
    1518500249,
    1543142273,
    1567371161,
    1591180425,
    1614563692,
    1637514701,
    1660027308,
    1682095485,
    1703713325,
    1724875039,
    1745574963,
    1765807554,
    1785567396,
    1804849198,
```

```
    1823647798,
    1841958164,
    1859775393,
    1877094715,
    1893911493,
    1910221226,
    1926019546,
    1941302224,
    1956065169,
    1970304428,
    1984016188,
    1997196780,
    2009842673,
    2021950483,
    2033516968,
    2044539032,
    2055013723,
    2064938237,
    2074309917,
    2083126254,
    2091384888,
    2099083607,
    2106220351,
    2112793209,
    2118800422,
    2124240380,
    2129111627,
    2133412860,
    2137142927,
    2140300829,
    2142885720,
    2144896909,
    2146333858,
    2147196181,
    2147483648
};

int sine_384(int x) {
    int sign;
    if (x >= 192) {
        sign = -1;
        x -= 192;
    } else {
        sign = 1;
    }
    if (x >= 96) {
        x = 192 - x;
    }
    return sine_lookup[x] * sign;
}
```

E MII

This benchmark is stored in `wps/wp5/sc_mii_sim`. The first file is the calling function; the latter two files are the MII implementation and its header file.

```
/**
 * mii_sim - Ethernet MII interface simulator example
 *
 * It's not a useful implementation of MII, but it's port-enabled and
 * has various options for trying out different approaches
 *
 * Author: Steve Kerrison <steve.kerrison@bristol.ac.uk>
 * Date: 30th October 2013
**/
#include <platform.h>
#include <stdio.h>
#include "mii_sim.h"

struct txports tx = {
  XS1_PORT_4A,
  XS1_PORT_1A,
  XS1_CLKBLK_1
};
struct rxports rx = {
  XS1_PORT_4B,
  XS1_PORT_1B,
  XS1_CLKBLK_2
};

#ifdef AEC_DIV
#define XCORE_CTRL0_CLOCK_MASK 0x30
#define XCORE_CTRL0_ENABLE_AEC 0x30
  void enableAEC(unsigned standbyClockDivider)
  {
    unsigned xcore_ctrl0_data;
    // Set standby divider
    write_pswitch_reg(get_local_tile_id(), XS1_PSWITCH_PLL_CLK_DIVIDER_NUM,
      standbyClockDivider);
    // Modify the clock control bits
    xcore_ctrl0_data = getps(XS1_PS_XCORE_CTRL0);
    xcore_ctrl0_data &= 0xffffffff - XCORE_CTRL0_CLOCK_MASK;
    xcore_ctrl0_data += XCORE_CTRL0_ENABLE_AEC;
    setps(XS1_PS_XCORE_CTRL0, xcore_ctrl0_data);
  }
#endif

int main(void) {
  unsigned txbuf[512];
  unsigned pkt_len[1] = {10},
    pkt_len_len = 1,
    duty_cycle_percent = 10, num_pkts = 10, tv1, tv2, words;
  timer t;
  for (int i = 0; i < 512; i +=1 ) {
    txbuf[i] = 0xcafe0000 + i;
  }
#ifdef AEC_DIV
  enableAEC(AEC_DIV);
#endif
  t :> tv1;
  par {
    mii_rx(rx,num_pkts);
    words = mii_tx(tx,txbuf,pkt_len,pkt_len_len,duty_cycle_percent,num_pkts);
  }
  t :> tv2;
  printf("TX'd %u packets (%u words) in %u refclks at %u%% duty cycle\n"
    "Bitrate: %.2f mbps\n",
    num_pkts,words,tv2-tv1,duty_cycle_percent,
    (words * 32 * (float)PLATFORM_REFERENCE_HZ/(tv2-tv1))/1000000);
  return 0;
```

```
}
```

```
/**
 * mii_sim - Ethernet MII interface simulator
 *
 * It's not a useful implementation of MII, but it's port-enabled and
 * has various options for trying out different approaches
 *
 * Author: Steve Kerrison <steve.kerrison@bristol.ac.uk>
 * Date: 21st October 2013
**/

#include <platform.h>
#include <stdlib.h>
#include <print.h>
#include "mii_sim.h"

unsigned mii_tx(struct txports &tx, unsigned buf[512],
  unsigned pkt_len[], size_t pkt_len_len, unsigned duty_cycle_percent,
  unsigned num_pkts) {
  size_t p = 0, bp = 0;
  timer t;
  unsigned txstart, txend, npk = 0, words = 0;
  // REFCLK / (2*param2) (Therefore 2 gives 25MHz)
  configure_clock_ref(tx.clk,MII_REF_DIV);
  configure_out_port_strobed_master(tx.txd,tx.txen,tx.clk,0);
  start_clock(tx.clk);
  while(npk++ < num_pkts) {
    size_t i = 0, lim = pkt_len[p];
    words += pkt_len[p];
#ifdef DEBUG_TX
    printstrln("TX:");
    for (int x = 0; x < lim; x += 1) {
      printhex(buf[(bp+x) & 0x1ff]);
      printchar(' ');
    }
    printchar('\n');
#endif
    t :> txstart;
#ifdef TX_SMALLBUFFER
    for (int j = 0; j < 7; j += 1) {
      tx.txd <: 0x55;
    }
    tx.txd <: 0xd5;
#else
    tx.txd <: 0x55555555;
    tx.txd <: 0xd5555555;
#endif
    do {
#ifdef TX_SMALLBUFFER
      unsigned word = buf[bp];
      tx.txd <: word;
      word >>= 8;
      tx.txd <: word;
      word >>= 8;
      tx.txd <: word;
      word >>= 8;
      tx.txd <: word;
#else
      tx.txd <: buf[bp];
#endif
      bp = (bp + 1) & 0x1ff;
    } while (i++ < lim);
    p = (p + 1) % pkt_len_len;
    t :> txend;
    /* Enforce some idle period based on duty cycle */
    {
      unsigned pkttime = txend - txstart,
        idletime = (pkttime * (100-duty_cycle_percent)) / duty_cycle_percent;
      /* Don't use a timer if it looks like it'll wrap! */
      idletime = idletime < TX_MIN_DELAY ? TX_MIN_DELAY : idletime;
```

```
      t when timerafter(txend + idletime) :> void;
    }
  }
  return words;
}

void mii_rx(struct rxports &rx, unsigned num_pkts) {
  unsigned buf[512];
  size_t bp = 0, npk = 0;
  rx.rxdv when pinseq(0) :> void;
  // REFCLK / (2*param2) (Therefore 2 gives 25MHz)
  configure_clock_ref(rx.clk,MII_REF_DIV);
  configure_in_port_strobed_slave(rx.rxd,rx.rxdv,rx.clk);
  start_clock(rx.clk);
  while(npk++ < num_pkts) {
    unsigned loop = 1, pkt_len = 0;
#ifdef EARLY_WAKE
    /*
     * If EARLY_WAKE is configured, we wake up as soon as RXDV is high,
     * then spin for the SOF nibble to avoid going back to sleep (because
     * disabling any automatic power saving would probably take too long to do).
     */
    rx.rxdv when pinseq(1) :> void;
#ifdef WAKE_BUSY
    /* With WAKE_BUSY we keep the processor active with a loop */
    while (rx.rxd != 0xd);
#else
    set_core_fast_mode_on();
    rx.rxd when pinseq(0xd) :> void;
#endif
#else
    /* Otherwise we just wait for the SOF nibble the usual way */
    rx.rxd when pinseq(0xd) :> void;
#endif
    while(loop) {
      select {
        case rx.rxd :> buf[bp]:
          bp = (bp + 1) & 0x1ff;
          pkt_len += 1;
          break;
        case rx.rxdv when pinseq(0) :> void:
          {
            unsigned bits = endin(rx.rxd);
            if (bits) {
              buf[bp] = partin(rx.rxd,bits);
              pkt_len += 1;
            }
            loop = 0;
          }
          break;
#if defined(EARLY_WAKE) && defined(WAKE_BUSY)
        /* If EARLY_WAKE is configured make sure we have an active idle loop */
        default:
          __asm__ __volatile__("nop"::);
          break;
#endif
      }
    }
#if defined(EARLY_WAKE) && !defined(WAKE_BUSY)
    set_core_fast_mode_off();
#endif
#ifdef DEBUG_RX
    {
      size_t tbp = (bp - pkt_len) & 0x1ff;
      printstrln("RX:");
      for (int x = 0; x < pkt_len; x += 1) {
        printhex(buf[(tbp+x) & 0x1ff]);
        printchar(' ');
      }
      printchar('\n');
    }
#endif
  }
```

```
  return;
}
```

```
/**
 * mii_sim - Ethernet MII interface simulator
 *
 * It's not a useful implementation of MII, but it's port-enabled and
 * has various options for trying out different approaches
 *
 * Author: Steve Kerrison <steve.kerrison@bristol.ac.uk>
 * Date: 21st October 2013
**/
#ifndef MII_SIM_H
#define MII_SIM_H

#include <platform.h>
#include <stdlib.h>

/* Minimum delay between packets */
#ifndef TX_MIN_DELAY
#define TX_MIN_DELAY 60
#endif

/* 100Mbit is default, but 10Mbit can be specified with this macro */
#ifdef ETH_10
#define MII_REF_DIV 20
#else
#define MII_REF_DIV 2
#endif


struct rxports {
#ifdef RX_SMALLBUFFER
  buffered in port:8 rxd;
#else
  buffered in port:32 rxd;
#endif
  in port rxdv;
  clock clk;
};

struct txports {
#ifdef TX_SMALLBUFFER
  buffered out port:8 txd;
#else
  buffered out port:32 txd;
#endif
  out port txen;
  clock clk;
};

unsigned mii_tx(struct txports &tx, unsigned buf[512],
  unsigned pkt_len[], size_t pkt_len_len, unsigned duty_cycle_percent,
  unsigned num_pkts);
void mii_rx(struct rxports &rx, unsigned num_pkts);

#endif //MII_SIM_H
```

F Image compression

This benchmark is stored in `wps/wp5/dct_transform`. It comprises one file that is of interest (`fdctint.S`) and three auxiliary files that are used as a test harness and as a test input image.

```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative
   ↪ of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

    // Forward DCT for JPEG - gets a patch of 8x8 (ints) and
       ↪  a quantisation table (8x8 ints).
    // Makes liberal use of CP and DP. To be optimised
       ↪ further.
    // Based on reference algorithm.

    // TODO: remove SHR prior to MACCU
    // TODO: rounding after phase 1 and phase 2.
    // TODO: remove abuse of stackpointer for loop iteration
       ↪  - there must be a better way.


    .globl   doDCT.nstackwords
    .linkset doDCT.nstackwords, 0

    .globl doDCT

    .align 4
doDCT:
    out   res[r0], r1
    in    r0, res[r0]
    retsp 0


    .globl   endDCT.nstackwords
    .linkset endDCT.nstackwords, 0

    .globl endDCT

    .align 4
endDCT:
    outct res[r0], 5
    chkct res[r0], 5
    retsp 0

#define FIX_0_298631336  (  2446)       /* FIX(0.298631336)
   ↪ */
#define FIX_0_390180644  (  3196)       /* FIX(0.390180644)
   ↪ */
#define FIX_0_541196100  (  4433)       /* FIX(0.541196100)
   ↪ */
#define FIX_0_765366865  (  6270)       /* FIX(0.765366865)
   ↪ */
#define FIX_0_899976223  (  7373)       /* FIX(0.899976223)
   ↪ */
#define FIX_1_175875602  (  9633)       /* FIX(1.175875602)
   ↪ */
#define FIX_1_501321110  (  12299)      /* FIX(1.501321110)
   ↪ */
#define FIX_1_847759065  (  15137)      /* FIX(1.847759065)
   ↪ */
#define FIX_1_961570560  (  16069)      /* FIX(1.961570560)
   ↪ */
#define FIX_2_053119869  (  16819)      /* FIX(2.053119869)
   ↪ */
```

```
#define FIX_2_562915447  (  20995)      /* FIX(2.562915447)
   ↪ */
#define FIX_3_072711026  (  25172)      /* FIX(3.072711026)
   ↪ */

#define N 14

    .globl   forwardDCT.nstackwords
    .linkset forwardDCT.nstackwords, N

    .globl forwardDCT

    .align 4
forwardDCT:
    entsp N
    stw r4, sp[0]
    stw r5, sp[1]
    stw r6, sp[2]
    stw r7, sp[3]
    stw r8, sp[4]
    stw r9, sp[5]
    stw r10, sp[6]
    stw r11, sp[7]
    stw dp, sp[8]
    stw cp, sp[9]
    stw r0, sp[10]
    stw r1, sp[11]

blockLoop:
    ldw cp, sp[11]
    testct r1, res[r0]
    bt r1, noMoreBlocks
    in r1, res[r0]
    set dp, r1
    stw r1, sp[12]
    ldc r11, 0                          // Just misalign
       ↪ nextRow - improves performance.

nextRow:
    ldw r0, dp[0]
    ldw r11, dp[7]
    sub r7, r0, r11
    add r0, r0, r11

    ldw r1, dp[1]
    ldw r11, dp[6]
    sub r6, r1, r11
    add r1, r1, r11

    ldw r2, dp[2]
    ldw r11, dp[5]
    sub r5, r2, r11
    add r2, r2, r11

    ldw r3, dp[3]
    ldw r11, dp[4]
    sub r4, r3, r11
    add r3, r3, r11

    add r10, r0, r3
    sub  r9, r0, r3
    add r11, r1, r2           // r0 & r3 dead
```

```
    sub  r8, r1, r2           // r1 & r2 dead

    add r0, r10, r11
    ldc r1, 1024               // Bias term - take 128
       ↪ offset in all pixels out.
    sub r0, r0, r1
    shl r0, r0, 2
    stw r0, dp[0]

    sub r0, r10, r11
    shl r0, r0, 2
    stw r0, dp[4]            // r10, r11 dead.

    ldc r2, 11

    add r0, r8, r9
    ldc r1,  FIX_0_541196100
    mul r0, r0, r1
    ldc r1,  FIX_0_765366865
    mul r9, r1, r9
    add r9, r9, r0
    ashr r9, r9, r2
    stw r9, dp[2]
    ldc r1,  FIX_1_847759065
    mul r8, r1, r8
    sub r8, r0, r8
    ashr r8, r8, r2
    stw r8, dp[6]            // r8, r9 dead

    add r1, r4, r7
    add r2, r5, r6
    add r3, r4, r6
    add r0, r5, r7
    add r11, r3, r0
    ldc r10, FIX_1_175875602
    mul r11, r11, r10

    ldc r10, FIX_0_298631336
    mul r4, r4, r10
    ldc r10, FIX_2_053119869
    mul r5, r5, r10
    ldc r10, FIX_3_072711026
    mul r6, r6, r10
    ldc r10, FIX_1_501321110
    mul r7, r7, r10

    ldc r10, FIX_0_899976223
    mul r1, r1, r10
    ldc r10, FIX_2_562915447
    mul r2, r2, r10
    ldc r10, FIX_1_961570560
    mul r3, r3, r10
    ldc r10, FIX_0_390180644
    mul r0, r0, r10

    sub r3, r11, r3
    sub r0, r11, r0

    ldc r10, 11

    sub r4, r4, r1
    add r4, r4, r3
    ashr r4, r4, r10
    stw r4, dp[7]

    sub r5, r5, r2
    add r5, r5, r0
    ashr r5, r5, r10
    stw r5, dp[5]

    sub r6, r6, r2
    add r6, r6, r3
    ashr r6, r6, r10
    stw r6, dp[3]

    sub r7, r7, r1
    add r7, r7, r0
    ashr r7, r7, r10
    stw r7, dp[1]

    ldaw dp, dp[8]

    ldaw r11, cp[1]
    set  cp, r11
    ldw  r10, cp[7*8]
    bt   r10, nextRow

    extdp 64


// End of ROW pass

    ldw cp, sp[11]

//    extsp 8

   // .align 4
nextColumn:
    ldw r0, dp[0*8]
    ldw r11, dp[7*8]
    sub r7, r0, r11
    add r0, r0, r11

    ldw r1, dp[1*8]
    ldw r11, dp[6*8]
    sub r6, r1, r11
    add r1, r1, r11

    ldw r2, dp[2*8]
    ldw r11, dp[5*8]
    sub r5, r2, r11
    add r2, r2, r11

    ldw r3, dp[3*8]
    ldw r11, dp[4*8]
    sub r4, r3, r11
    add r3, r3, r11

    add r10, r0, r3
    sub  r9, r0, r3
    add r11, r1, r2           // r0 & r3 dead
    sub  r8, r1, r2           // r1 & r2 dead

    add r0, r10, r11
    ashr r0, r0, 2
    ldw r1, cp[0*8]
    ldc r2, 0
    ldc r3, 0x80
    shl r3, r3, 24
    maccs r2, r3, r1, r0
    stw r2, dp[0*8]

    sub r0, r10, r11
    ashr r0, r0, 2
    ldw r1, cp[4*8]
    ldc r2, 0
    ldc r3, 0x80
    shl r3, r3, 24
    maccs r2, r3, r1, r0
    stw r2, dp[4*8]            // r10, r11 dead.

    ldc r10, 15

    add r0, r8, r9
    ldc r1,  FIX_0_541196100
    mul r0, r0, r1
```

```
    ldc r1,  FIX_0_765366865
    mul r9, r1, r9
    add r9, r9, r0
    ashr r9, r9, r10
    ldw r1, cp[2*8]
    ldc r2, 0
    ldc r3, 0x80
    shl r3, r3, 24
    maccs r2, r3, r9, r1
    stw r2, dp[2*8]
    ldc r1,  FIX_1_847759065
    mul r8, r1, r8
    sub r8, r0, r8
    ashr r8, r8, r10
    ldw r1, cp[6*8]
    ldc r2, 0
    ldc r3, 0x80
    shl r3, r3, 24
    maccs r2, r3, r8, r1
    stw r2, dp[6*8]            // r8, r9 dead

    add r1, r4, r7
    add r2, r5, r6
    add r3, r4, r6
    add r0, r5, r7
    add r11, r3, r0
    ldc r10, FIX_1_175875602
    mul r11, r11, r10

    ldc r10, FIX_0_298631336
    mul r4, r4, r10
    ldc r10, FIX_2_053119869
    mul r5, r5, r10
    ldc r10, FIX_3_072711026
    mul r6, r6, r10
    ldc r10, FIX_1_501321110
    mul r7, r7, r10

    ldc r10, FIX_0_899976223
    mul r1, r1, r10
    ldc r10, FIX_2_562915447
    mul r2, r2, r10
    ldc r10, FIX_1_961570560
    mul r3, r3, r10
    ldc r10, FIX_0_390180644
    mul r0, r0, r10

    sub r3, r11, r3
    sub r0, r11, r0

    ldc r10, 15

    sub r4, r4, r1
    add r4, r4, r3
    ashr r4, r4, r10
    ldw r8, cp[7*8]
    ldc r9, 0
    ldc r11, 0x80
    shl r11, r11, 24
    maccs r9, r11, r4, r8
    stw r9, dp[7*8]

    sub r5, r5, r2
    add r5, r5, r0
    ashr r5, r5, r10
    ldw r8, cp[5*8]
    ldc r9, 0
    ldc r11, 0x80
    shl r11, r11, 24
    maccs r9, r11, r5, r8
    stw r9, dp[5*8]

    sub r6, r6, r2
    add r6, r6, r3
    ashr r6, r6, r10
    ldw r8, cp[3*8]
    ldc r9, 0
    ldc r11, 0x80
    shl r11, r11, 24
    maccs r9, r11, r6, r8
    stw r9, dp[3*8]

    sub r7, r7, r1
    add r7, r7, r0
    ashr r7, r7, r10
    ldw r8, cp[1*8]
    ldc r9, 0
    ldc r11, 0x80
    shl r11, r11, 24
    maccs r9, r11, r7, r8
    stw r9, dp[1*8]

    ldaw dp, dp[1]
    ldaw r11, cp[1]
    set  cp, r11
    ldw  r11, cp[7*8]
    bt   r11, nextColumn

    ldw r0, sp[10]
    ldw r1, sp[12]
    out res[r0], r1

    bu blockLoop
noMoreBlocks:
    inct r1, res[r0]
    outct res[r0], r1

    ldw r4, sp[0]
    ldw r5, sp[1]
    ldw r6, sp[2]
    ldw r7, sp[3]
    ldw r8, sp[4]
    ldw r9, sp[5]
    ldw r10, sp[6]
    ldw r11, sp[7]
        ldw dp, sp[8]
        ldw cp, sp[9]

    retsp N
```

G Small numerical benchmarks

These benchmarks are stored in `wps/wp5/short_numerical`, in five subdirectories with self explanatory names.

Each directory contain two files. The code of the benchmark, and the code of the calling function. The calling function defines standard parameter settings that enable us to make comparisons.

```
/*
 * factorial.xc
 *
 *  Created on: May 17, 2013
 *      Author: umer
 * Description: Computes factorial of a number. Exhibit polynomial time complexity.
 */

int fact(int i) {
  if(i<=0)  return 1;
  return i*fact(i-1);
}
```

```
/*
 * fibonacci.xc
 *
 *  Created on: May 17, 2013
 *      Author: umer
 * Description: Computes nth fibonacci number, exhibiting exponential time complexity.
 */

#include "fibonacci.h"


int fibonacci(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return ( fibonacci(n-1) + fibonacci(n-2) );
}
```

```
/*
 * poweroftwo.xc
 *
 *  Created on: Jun 13, 2013
 *      Author: umer
 * Description: Computes power of two.
 */
#include "poweroftwo.h"

int powerof2(int n)
{
        if(n==0)
                return 1;
        return powerof2(n-1)+powerof2(n-1);
}
```

```
/*
 * sqr.xc
```

```
 *
 *  Created on: Jun 13, 2013
 *      Author: umer
 * Description: Computes square of a number.
 */
#include "sqr.h"

int sqr_aux(int n)
{
  if (n == 0)
    return 0;
  return 1 + sqr_aux(n-1);
}

int sqr(int n)
{
  if (n == 0)
    return 0;
  return sqr_aux(n) + sqr(n-1);
}
```

```
/*
 * power.xc
 *
 *  Created on: Jun 13, 2013
 *      Author: umer
 * Description: Calculates power with base and exponent as input. Exhibit multiple input
 *              argument functions and multiple user defined functions.
 */
#include "power.h"

int power_aux(int y)
{
  if (y == 0)
    return 0;
  return 1 + power_aux(y-1);
}

int power(int x, int y)
{
  if (x == 0)
    return 0;
  return power_aux(y) + power(x-1,y);
}
```

H Polling loop

This benchmark is stored in `wps/wp5/polling_loop`. It comprises one file that contains all three tasks; a second file `correct.xc` shows what the code should have looked like if it had been coded correctly.

```
#include <stdio.h>
#include <xs1.h>

void consumer_polling(chanend a, chanend b) {
    while(1) {
        select {
        case a :> int x:
            printf("X is %d\n", x);
            break;
        default:
            break;
        }
        select {
        case b :> int y:
            printf("        Y is %d\n", y);
            break;
        default:
            break;
        }
    }
}

void produce(chanend x, int delay) {
    timer tmr;
    int t;
    tmr :> t;
    for(int i = 1; i < 10; i++) {
        t += delay;
        tmr when timerafter(t) :> void;
        x <: i;
    }
}

int main() {
    chan a, b;
    par {
        consumer_polling(a, b);
        produce(a, 10000);
        produce(b, 16000);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <xs1.h>

void consumer_non_polling(chanend a, chanend b) {
    while(1) {
        select {
        case a :> int x:
            printf("X is %d\n", x);
            break;
        case b :> int y:
            printf("        Y is %d\n", y);
            break;
        }
    }
}

void produce(chanend x, int delay) {
```

```
    timer tmr;
    int t;
    tmr :> t;
    for(int i = 1; i < 10; i++) {
        t += delay;
        tmr when timerafter(t) :> void;
        x <: i;
    }
}

int main() {
    chan a, b;
    par {
        consumer_non_polling(a, b);
        produce(a, 10000);
        produce(b, 16000);
    }
    return 0;
}
```

I WCET Benchmarks

These benchmarks are stored in `wps/wp5/wcet_benchmarks`, in seven subdirectories with self explanatory names.

Each directory contains a file with the code of the benchmark together with the code of the calling function. The calling function defines standard parameter settings that enable us to make comparisons. Unlike other programs these programs are written in C.

```
/**************************************************************************/
/*                                                                        */
/*   SNU-RT Benchmark Suite for Worst Case Timing Analysis                */
/*   =====================================================                */
/*                              Collected and Modified by S.-S. Lim        */
/*                                        sslim@archi.snu.ac.kr           */
/*                                     Real-Time Research Group           */
/*                                    Seoul National University           */
/*                                                                        */
/*                                                                        */
/*        &lt; Features &gt; - restrictions for our experimental environment   */
/*                                                                        */
/*          1. Completely structured.                                     */
/*               - There are no unconditional jumps.                      */
/*               - There are no exit from loop bodies.                    */
/*                 (There are no 'break' or 'return' in loop bodies)      */
/*          2. No 'switch' statements.                                    */
/*          3. No 'do..while' statements.                                 */
/*          4. Expressions are restricted.                                */
/*               - There are no multiple expressions joined by 'or',      */
/*                'and' operations.                                       */
/*          5. No library calls.                                          */
/*               - All the functions needed are implemented in the        */
/*                 source file.                                           */
/*                                                                        */
/*                                                                        */
/**************************************************************************/
/*                                                                        */
/*  FILE: bs.c                                                            */
/*  SOURCE : Public Domain Code                                           */
/*                                                                        */
/*  DESCRIPTION :                                                         */
/*                                                                        */
/*     Binary search for the array of 15 integer elements.                */
/*                                                                        */
/*  REMARK :                                                              */
/*                                                                        */
/*  EXECUTION TIME :                                                      */
/*                                                                        */
/*                                                                        */
/**************************************************************************/



struct DATA {
  int  key;
  int  value;
}  ;

#ifdef DEBUG
        int cnt1;
#endif

struct DATA data[15] = { {1, 100},
             {5,200},
             {6, 300},
             {7, 700},
```

```
            {8, 900},
            {9, 250},
            {10, 400},
            {11, 600},
            {12, 800},
            {13, 1500},
            {14, 1200},
            {15, 110},
            {16, 140},
            {17, 133},
            {18, 10} };

main()
{
        binary_search(8);
}

binary_search(x)
{
  int fvalue, mid, up, low ;

  low = 0;
  up = 14;
  fvalue = -1 /* all data are positive */ ;
  while (low &lt;= up) {
    mid = (low + up) &gt;&gt; 1;
    if ( data[mid].key == x ) {  /*  found  */
      up = low - 1;
      fvalue = data[mid].value;
#ifdef DEBUG
        printf("FOUND!!\n");
#endif
    }
    else  /* not found */
      if ( data[mid].key &gt; x )       {
        up = mid - 1;
#ifdef DEBUG
        printf("MID-1\n");
#endif
      }
      else   {
                low = mid + 1;
#ifdef DEBUG
        printf("MID+1\n");
#endif
      }
#ifdef DEBUG
        cnt1++;
#endif
  }
#ifdef DEBUG
        printf("Loop Count : %d\n", cnt1);
#endif
  return fvalue;
}
```

```
/* Changes:
 * JG 2005/12/21: Inserted function prototypes
                  Indented program.
 */

int             swi120(int c);
int             swi50(int c);
int             swi10(int c);

int
swi120(int c)
{
        int             i;
```

```
	for (i = 0; i < 120; i++) {
		switch (i) {
		case 0:
			c++;
			break;
		case 1:
			c++;
			break;
		case 2:
			c++;
			break;
		case 3:
			c++;
			break;
		case 4:
			c++;
			break;
		case 5:
			c++;
			break;
		case 6:
			c++;
			break;
		case 7:
			c++;
			break;
		case 8:
			c++;
			break;
		case 9:
			c++;
			break;
		case 10:
			c++;
			break;
		case 11:
			c++;
			break;
		case 12:
			c++;
			break;
		case 13:
			c++;
			break;
		case 14:
			c++;
			break;
		case 15:
			c++;
			break;
		case 16:
			c++;
			break;
		case 17:
			c++;
			break;
		case 18:
			c++;
			break;
		case 19:
			c++;
			break;
		case 20:
			c++;
			break;
		case 21:
			c++;
			break;
		case 22:
			c++;
			break;
		case 23:
			c++;
```

```
        break;
    case 24:
        c++;
        break;
    case 25:
        c++;
        break;
    case 26:
        c++;
        break;
    case 27:
        c++;
        break;
    case 28:
        c++;
        break;
    case 29:
        c++;
        break;
    case 30:
        c++;
        break;
    case 31:
        c++;
        break;
    case 32:
        c++;
        break;
    case 33:
        c++;
        break;
    case 34:
        c++;
        break;
    case 35:
        c++;
        break;
    case 36:
        c++;
        break;
    case 37:
        c++;
        break;
    case 38:
        c++;
        break;
    case 39:
        c++;
        break;
    case 40:
        c++;
        break;
    case 41:
        c++;
        break;
    case 42:
        c++;
        break;
    case 43:
        c++;
        break;
    case 44:
        c++;
        break;
    case 45:
        c++;
        break;
    case 46:
        c++;
        break;
    case 47:
        c++;
        break;
```

```
                case 48:
                        c++;
                        break;
                case 49:
                        c++;
                        break;
                case 50:
                        c++;
                        break;
                case 51:
                        c++;
                        break;
                case 52:
                        c++;
                        break;
                case 53:
                        c++;
                        break;
                case 54:
                        c++;
                        break;
                case 55:
                        c++;
                        break;
                case 56:
                        c++;
                        break;
                case 57:
                        c++;
                        break;
                case 58:
                        c++;
                        break;
                case 59:
                        c++;
                        break;
                case 60:
                        c++;
                        break;
                case 61:
                        c++;
                        break;
                case 62:
                        c++;
                        break;
                case 63:
                        c++;
                        break;
                case 64:
                        c++;
                        break;
                case 65:
                        c++;
                        break;
                case 66:
                        c++;
                        break;
                case 67:
                        c++;
                        break;
                case 68:
                        c++;
                        break;
                case 69:
                        c++;
                        break;
                case 70:
                        c++;
                        break;
                case 71:
                        c++;
                        break;
                case 72:
```

```
                c++;
                break;
        case 73:
                c++;
                break;
        case 74:
                c++;
                break;
        case 75:
                c++;
                break;
        case 76:
                c++;
                break;
        case 77:
                c++;
                break;
        case 78:
                c++;
                break;
        case 79:
                c++;
                break;
        case 80:
                c++;
                break;
        case 81:
                c++;
                break;
        case 82:
                c++;
                break;
        case 83:
                c++;
                break;
        case 84:
                c++;
                break;
        case 85:
                c++;
                break;
        case 86:
                c++;
                break;
        case 87:
                c++;
                break;
        case 88:
                c++;
                break;
        case 89:
                c++;
                break;
        case 90:
                c++;
                break;
        case 91:
                c++;
                break;
        case 92:
                c++;
                break;
        case 93:
                c++;
                break;
        case 94:
                c++;
                break;
        case 95:
                c++;
                break;
        case 96:
                c++;
```

```
        break;
    case 97:
        c++;
        break;
    case 98:
        c++;
        break;
    case 99:
        c++;
        break;
    case 100:
        c++;
        break;
    case 101:
        c++;
        break;
    case 102:
        c++;
        break;
    case 103:
        c++;
        break;
    case 104:
        c++;
        break;
    case 105:
        c++;
        break;
    case 106:
        c++;
        break;
    case 107:
        c++;
        break;
    case 108:
        c++;
        break;
    case 109:
        c++;
        break;
    case 110:
        c++;
        break;
    case 111:
        c++;
        break;
    case 112:
        c++;
        break;
    case 113:
        c++;
        break;
    case 114:
        c++;
        break;
    case 115:
        c++;
        break;
    case 116:
        c++;
        break;
    case 117:
        c++;
        break;
    case 118:
        c++;
        break;
    case 119:
        c++;
        break;
    default:
        c--;
        break;
```

```
                }
        }
        return c;
}


int
swi50(int c)
{
        int             i;
        for (i = 0; i < 50; i++) {
                switch (i) {
                case 0:
                        c++;
                        break;
                case 1:
                        c++;
                        break;
                case 2:
                        c++;
                        break;
                case 3:
                        c++;
                        break;
                case 4:
                        c++;
                        break;
                case 5:
                        c++;
                        break;
                case 6:
                        c++;
                        break;
                case 7:
                        c++;
                        break;
                case 8:
                        c++;
                        break;
                case 9:
                        c++;
                        break;
                case 10:
                        c++;
                        break;
                case 11:
                        c++;
                        break;
                case 12:
                        c++;
                        break;
                case 13:
                        c++;
                        break;
                case 14:
                        c++;
                        break;
                case 15:
                        c++;
                        break;
                case 16:
                        c++;
                        break;
                case 17:
                        c++;
                        break;
                case 18:
                        c++;
                        break;
                case 19:
                        c++;
                        break;
                case 20:
```

```
                c++;
                break;
        case 21:
                c++;
                break;
        case 22:
                c++;
                break;
        case 23:
                c++;
                break;
        case 24:
                c++;
                break;
        case 25:
                c++;
                break;
        case 26:
                c++;
                break;
        case 27:
                c++;
                break;
        case 28:
                c++;
                break;
        case 29:
                c++;
                break;
        case 30:
                c++;
                break;
        case 31:
                c++;
                break;
        case 32:
                c++;
                break;
        case 33:
                c++;
                break;
        case 34:
                c++;
                break;
        case 35:
                c++;
                break;
        case 36:
                c++;
                break;
        case 37:
                c++;
                break;
        case 38:
                c++;
                break;
        case 39:
                c++;
                break;
        case 40:
                c++;
                break;
        case 41:
                c++;
                break;
        case 42:
                c++;
                break;
        case 43:
                c++;
                break;
        case 44:
                c++;
```

```
                        break;
                case 45:
                        c++;
                        break;
                case 46:
                        c++;
                        break;
                case 47:
                        c++;
                        break;
                case 48:
                        c++;
                        break;
                case 49:
                        c++;
                        break;
                case 50:
                        c++;
                        break;
                case 51:
                        c++;
                        break;
                case 52:
                        c++;
                        break;
                case 53:
                        c++;
                        break;
                case 54:
                        c++;
                        break;
                case 55:
                        c++;
                        break;
                case 56:
                        c++;
                        break;
                case 57:
                        c++;
                        break;
                case 58:
                        c++;
                        break;
                case 59:
                        c++;
                        break;
                default:
                        c--;
                        break;
                }
        }
        return c;
}


int
swi10(int c)
{
        int             i;
        for (i = 0; i < 10; i++) {
                switch (i) {
                case 0:
                        c++;
                        break;
                case 1:
                        c++;
                        break;
                case 2:
                        c++;
                        break;
                case 3:
                        c++;
                        break;
```

```
                case 4:
                        c++;
                        break;
                case 5:
                        c++;
                        break;
                case 6:
                        c++;
                        break;
                case 7:
                        c++;
                        break;
                case 8:
                        c++;
                        break;
                case 9:
                        c++;
                        break;
                default:
                        c--;
                        break;
                }
        }
        return c;
}

int
main()
{
        volatile int    cnt = 0;

        cnt = swi10(cnt);
        cnt = swi50(cnt);
        cnt = swi120(cnt);

        /* printf("cnt: %d\n", cnt); */

        return cnt;

}
```

```
/**************************************************************************
 * FROM:
 *   http://sron9907.sron.nl/manual/numrecip/c/expint.c
 *
 * FEATURE:
 *   One loop depends on a loop-invariant value to determine
 *   if it run or not.
 *
 **************************************************************************/

 /*
  * Changes: JG 2005/12/23: Changed type of main to int, added prototypes.
                            Indented program.
  */

long int        foo(long int x);
long int        expint(int n, long int x);

int
main(void)
{
        expint(50, 1);
        /* with  expint(50,21) as argument, runs the short path */
        /* in expint.   expint(50,1)  gives the longest execution time */
        return 0;
}

long int
```

```
foo(long int x)
{
        return x * x + (8 * x) << (4 - x);
}


/* Function with same flow, different data types,
   nonsensical calculations */
long int
expint(int n, long int x)
{
        int             i, ii, nm1;
        long int        a, b, c, d, del, fact, h, psi, ans;

        nm1 = n - 1;            /* arg=50 --> 49 */

        if (x > 1) {            /* take this leg? */
                b = x + n;
                c = 2e6;
                d = 3e7;
                h = d;

                for (i = 1; i <= 100; i++) {    /* MAXIT is 100 */
                        a = -i * (nm1 + i);
                        b += 2;
                        d = 10 * (a * d + b);
                        c = b + a / c;
                        del = c * d;
                        h *= del;
                        if (del < 10000) {
                                ans = h * -x;
                                return ans;
                        }
                }
        } else {                /* or this leg? */
                /* For the current argument, will always take */
                /* '2' path here: */
                ans = nm1 != 0 ? 2 : 1000;
                fact = 1;
                for (i = 1; i <= 100; i++) {    /* MAXIT */
                        fact *= -x / i;
                        if (i != nm1)   /* depends on parameter n */
                                del = -fact / (i - nm1);
                        else {  /* this fat piece only runs ONCE *//* runs on
                                 * iter 49 */
                                psi = 0x00FF;
                                for (ii = 1; ii <= nm1; ii++)   /* */
                                        psi += ii + nm1;
                                del = psi + fact * foo(x);
                        }
                        ans += del;
                        /* conditional leave removed */
                }

        }
        return ans;
}
```

```
/* $Id: fibcall.c,v 1.2 2005/04/04 11:34:58 csg Exp $ */

/*************************************************************************/
/*                                                                       */
/*   SNU-RT Benchmark Suite for Worst Case Timing Analysis               */
/*   =====================================================               */
/*                                Collected and Modified by S.-S. Lim     */
/*                                        sslim@archi.snu.ac.kr          */
/*                                     Real-Time Research Group          */
/*                                    Seoul National University          */
/*                                                                       */
/*                                                                       */
/*        < Features > - restrictions for our experimental environment   */
```

```
/*                                                                      */
/*          1. Completely structured.                                   */
/*               - There are no unconditional jumps.                    */
/*               - There are no exit from loop bodies.                  */
/*                 (There are no 'break' or 'return' in loop bodies)    */
/*          2. No 'switch' statements.                                  */
/*          3. No 'do..while' statements.                               */
/*          4. Expressions are restricted.                              */
/*               - There are no multiple expressions joined by 'or',    */
/*                'and' operations.                                     */
/*          5. No library calls.                                        */
/*               - All the functions needed are implemented in the      */
/*                 source file.                                         */
/*                                                                      */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/*  FILE: fibcall.c                                                     */
/*  SOURCE : Public Domain Code                                         */
/*                                                                      */
/*  DESCRIPTION :                                                       */
/*                                                                      */
/*     Summing the Fibonacci series.                                    */
/*                                                                      */
/*  REMARK :                                                            */
/*                                                                      */
/*  EXECUTION TIME :                                                    */
/*                                                                      */
/*                                                                      */
/************************************************************************/



int fib(int n)
{
  int  i, Fnew, Fold, temp,ans;

    Fnew = 1;  Fold = 0;
    for ( i = 2;
          i <= 30 && i <= n;          /* apsim_loop 1 0 */
          i++ )
    {
      temp = Fnew;
      Fnew = Fnew + Fold;
      Fold = temp;
    }
    ans = Fnew;
  return ans;
}

int main()
{
  int a;

  a = 30;
  fib(a);
  return a;
}
```

```
/* MDH WCET BENCHMARK SUITE. File version $Id: janne_complex.c,v 1.2 2005/11/11 10:30:51 ael01 Exp $ */

/*--------------------------------------------------------------------
 *  WCET Benchmark created by Andreas Ermedahl, Uppsala university,
 *  May 2000.
 *
 *  The purpose of this benchmark is to have two loop where the inner
 *  loops max number of iterations depends on the outer loops current
 *  iterations.
 *
 * The example appeard for the first time in:
 *
```

```
 *  @InProceedings{Ermedahl:Annotations,
 * author =       "A. Ermedahl and J. Gustafsson",
 * title =        "Deriving Annotations for Tight Calculation of Execution Time",
 * year =         1997,
 * month =        aug,
 * booktitle =    EUROPAR97,
 * publisher =    "Springer Verlag",
 * pages =        "1298-1307"
 * }
 *
 * The result should be the following:
 *   outer loop:       1   2   3   4   5   6   7   8   9   10   11
 *   inner loop max:   5   9   8   7   4   2   1   1   1   1    1
 *
 *-------------------------------------------------------------------------*/

 /*
  * Changes: JG 2005/12/25: Inserted prototypes.
                            Indented program.
  */

int             complex(int a, int b);

int
complex(int a, int b)
{
        while (a < 30) {
                while (b < a) {
                        if (b > 5)
                                b = b * 3;
                        else
                                b = b + 2;
                        if (b >= 10 && b <= 12)
                                a = a + 10;
                        else
                                a = a + 1;
                }
                a = a + 2;
                b = b - 10;
        }
        return 1;
}

int
main()
{
        /* a = [1..30] b = [1..30] */
        int             a = 1, b = 1, answer = 0;
        /*
         * if(answer) {a = 1; b = 1;} else {a = 30; b = 30;}
         */
        answer = complex(a, b);
        return answer;
}
```

```
/* MDH WCET BENCHMARK SUITE. */

/* 2012/10/03, Jan Gustafsson <jan.gustafsson@mdh.se>
 * Changes:
 *  - init of "is" fixed (added a lot of brackets)
 *  - warning: array subscript is of type 'char': fixed in three places
 */


/* #include <math.h> -- no include files in Uppsala tests, plz */

/* All output disabled for wcsim */

/* A read from this address will result in an known value of 1 */
/* #define KNOWN_VALUE (int)(*((char *)0x80200001)) Changed JG/Ebbe */
#define KNOWN_VALUE 1
```

```
/* A read from this address will result in an unknown value */
#define UNKNOWN_VALUE (int)(*((char *)0x80200003))


#define WORSTCASE 1

typedef struct IMMENSE { unsigned long l, r; } immense;
typedef struct GREAT { unsigned long l, c, r; } great;

unsigned long bit[33];

static immense icd;
static char ipc1[57]={0,57,49,41,33,25,17,9,1,58,50,
   42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,
   52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,
   30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4};
static char ipc2[49]={0,14,17,11,24,1,5,3,28,15,6,21,
   10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,
   37,47,55,30,40,51,45,33,48,44,49,39,56,34,
   53,46,42,50,36,29,32};

void des(immense inp, immense key, int * newkey, int isw, immense * out);
unsigned long getbit(immense source, int bitno, int nbits);
void ks(/*immense key, */int n, great * kn);
void cyfun(unsigned long ir, great k, unsigned long * iout);

void des(immense inp, immense key, int * newkey, int isw, immense * out) {

   static char ip[65] =
      {0,58,50,42,34,26,18,10,2,60,52,44,36,
      28,20,12,4,62,54,46,38,30,22,14,6,64,56,48,40,
      32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,
      27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,
      31,23,15,7};
   static char ipm[65]=
      {0,40,8,48,16,56,24,64,32,39,7,47,15,
      55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
      53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
      51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
      49,17,57,25};
   static great kns[17];
#ifdef WORSTCASE
   static int initflag=1;
#else
   static int initflag=0;
#endif
   int ii,i,j,k;
   unsigned long ic,shifter,getbit();
   immense itmp;
   great pg;

   if (initflag) {
      initflag=0;
      bit[1]=shifter=1L;
      for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
      }
   if (*newkey) {
      *newkey=0;
      icd.r=icd.l=0L;
      for (j=28,k=56;j>=1;j--,k--) {
         icd.r = (icd.r <<= 1) | getbit(key,ipc1[j],32);
         icd.l = (icd.l <<= 1) | getbit(key,ipc1[k],32);
         }

      for(i=1;i<=16;i++) {pg = kns[i]; ks(/* key,*/ i, &pg); kns[i] = pg;}
      }
   itmp.r=itmp.l=0L;
   for (j=32,k=64;j>=1;j--,k--) {
      itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
      itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
      }
   for (i=1;i<=16;i++) {
```

```
      ii = (isw == 1 ? 17-i : i);
      cyfun(itmp.l, kns[ii], &ic);
      ic ^= itmp.r;
      itmp.r=itmp.l;
      itmp.l=ic;
      }
   ic=itmp.r;
   itmp.r=itmp.l;
   itmp.l=ic;
   (*out).r=(*out).l=0L;
   for (j=32,k=64; j >= 1; j--, k--) {
      (*out).r = ((*out).r <<= 1) | getbit(itmp,ipm[j],32);
      (*out).l = ((*out).l <<= 1) | getbit(itmp,ipm[k],32);
      }
}
unsigned long getbit(immense source, int bitno, int nbits) {
   if (bitno <= nbits)
      return bit[bitno] & source.r ? 1L : 0L;
   else
      return bit[bitno-nbits] & source.l ? 1L : 0L;
}

void ks(/*immense key, */int n, great * kn) {
   int i,j,k,l;

   if (n == 1 || n == 2 || n == 9 || n == 16) {
      icd.r = (icd.r | ((icd.r & 1L) << 28)) >> 1;
      icd.l = (icd.l | ((icd.l & 1L) << 28)) >> 1;
      }
   else
      for (i=1;i<=2;i++) {
         icd.r = (icd.r | ((icd.r & 1L) << 28)) >> 1;
         icd.l = (icd.l | ((icd.l & 1L) << 28)) >> 1;
         }
   (*kn).r=(*kn).c=(*kn).l=0;
   for (j=16,k=32,l=48; j>=1; j--,k--,l--) {
      (*kn).r=((*kn).r <<= 1) | (unsigned short)
          getbit(icd,ipc2[j],28);
      (*kn).c=((*kn).c <<= 1) | (unsigned short)
          getbit(icd,ipc2[k],28);
      (*kn).l=((*kn).l <<= 1) | (unsigned short)
          getbit(icd,ipc2[l],28);
      }
}

void cyfun(unsigned long ir, great k, unsigned long * iout) {
   static int iet[49]={0,32,1,2,3,4,5,4,5,6,7,8,9,8,9,
      10,11,12,13,12,13,14,15,16,17,16,17,18,19,
      20,21,20,21,22,23,24,25,24,25,26,27,28,29,
      28,29,30,31,32,1};
   static int ipp[33]={0,16,7,20,21,29,12,28,17,1,15,
      23,26,5,18,31,10,2,8,24,14,32,27,3,9,19,13,
      30,6,22,11,4,25};
   static char is[16][4][9]={
     {{0,14,15,10,7,2,12,4,13},{0,0,3,13,13,14,10,13,1},
      {0,4,0,13,10,4,9,1,7},{0,15,13,1,3,11,4,6, 2}},
     {{0,4,1,0,13,12,1,11,2},{0,15,13,7,8,11,15,0,15},
      {0,1,14,6,6,2,14,4,11},{0,12,8,10,15,8,3,11,1}},
     {{0,13,8,9,14,4,10,2,8},{0,7,4,0,11,2,4,11,13},
      {0,14,7,4,9,1,15,11,4},{0,8,10,13,0,12,2,13,14}},
     {{0,1,14,14,3,1,15,14,4},{0,4,7,9,5,12,2,7,8},
      {0,8,11,9,0,11,5,13,1},{0,2,1,0,6,7,12,8,7}},
     {{0,2,6,6,0,7,9,15,6},{0,14,15,3,6,4,7,4,10},
      {0,13,10,8,12,10,2,12,9},{0,4,3,6,10,1,9,1,4}},
     {{0,15,11,3,6,10,2,0,15},{0,2,2,4,15,7,12,9,3},
      {0,6,4,15,11,13,8,3,12},{0,9,15,9,1,14,5,4,10}},
     {{0,11,3,15,9,11,6,8,11},{0,13,8,6,0,13,9,1,7},
       {0,2,13,3,7,7,12,7,14},{0,1,4,8,13,2,15,10,8}},
     {{0,8,4,5,10,6,8,13,1},{0,1,14,10,3,1,5,10,4},
      {0,11,1,0,13,8,3,14,2},{0,7,2,7,8,13,10,7,13}},
     {{0,3,9,1,1,8,0,3,10},{0,10,12,2,4,5,6,14,12},
      {0,15,5,11,15,15,7,10,0},{0,5,11,4,9,6,11,9,15}},
     {{0,10,7,13,2,5,13,12,9},{0,6,0,8,7,0,1,3,5},
```

```
      {0,12,8,1,1,9,0,15,6},{0,11,6,15,4,15,14,5,12}},
     {{0,6,2,12,8,3,3,9,3},{0,12,1,5,2,15,13,5,6},
      {0,9,12,2,3,12,4,6,10},{0,3,7,14,5,0,1,0,9}},
     {{0,12,13,7,5,15,4,7,14},{0,11,10,14,12,10,14,12,11},
      {0,7,6,12,14,5,10,8,13},{0,14,12,3,11,9,7,15,0}},
     {{0,5,12,11,11,13,14,5,5},{0,9,6,12,1,3,0,2,0},
      {0,3,9,5,5,6,1,0,15},{0,10,0,11,12,10,6,14,3}},
     {{0,9,0,4,12,0,7,10,0},{0,5,9,11,10,9,11,15,14},
      {0,10,3,10,2,3,13,5,3},{0,0,5,5,7,4,0,2,5}},
     {{0,0,5,2,4,14,5,6,12},{0,3,11,15,14,8,3,8,9},
      {0,5,2,14,8,0,11,9,5},{0,6,14,2,2,5,8,3,6}},
     {{0,7,10,8,15,9,11,1,7},{0,8,5,1,9,6,8,6,2},
       {0,0,15,7,4,14,6,2,8},{0,13,9,12,14,3,13,12,11}}};
   static char ibin[16]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15};
   great ie;
   unsigned long itmp,ietmp1,ietmp2;
   char iec[9];
   int jj,irow,icol,iss,j,l,m;
   unsigned long *p;

   p = bit;
   ie.r=ie.c=ie.l=0;
   for (j=16,l=32,m=48;j>=1;j--,l--,m--) {
      ie.r = (ie.r <<=1) | (p[iet[j]] & ir ? 1 : 0);
      ie.c = (ie.c <<=1) | (p[iet[l]] & ir ? 1 : 0);
      ie.l = (ie.l <<=1) | (p[iet[m]] & ir ? 1 : 0);
      }
   ie.r ^= k.r;
   ie.c ^= k.c;
   ie.l ^= k.l;
   ietmp1=((unsigned long) ie.c << 16)+(unsigned long) ie.r;
   ietmp2=((unsigned long) ie.l << 8)+((unsigned long) ie.c >> 8);
   for (j=1,m=5;j<=4;j++,m++) {
      iec[j]=ietmp1 & 0x3fL;
      iec[m]=ietmp2 & 0x3fL;
      ietmp1 >>= 6;
      ietmp2 >>= 6;
      }
   itmp=0L;
   for (jj=8;jj>=1;jj--) {
      j =iec[jj];
      irow=((j & 0x1) << 1)+((j & 0x20) >> 5);
      icol=((j & 0x2) << 2)+(j & 0x4)
          +((j & 0x8) >> 2)+((j & 0x10) >> 4);
      iss=is[icol][irow][jj];
      itmp = (itmp <<= 4) | ibin[iss];
      }
   *iout=0L;
   p = bit;
   for (j=32;j>=1;j--)
      *iout = (*iout <<= 1) | (p[ipp[j]] & itmp ? 1 : 0);
}
#ifdef WORSTCASE
   int value = 1;
#else
   int value = 0;
#endif

int main(void)
{
   immense inp, key, out;
   int newkey, isw;

   inp.l = KNOWN_VALUE * 35;
   inp.r = KNOWN_VALUE * 26;
   key.l = KNOWN_VALUE * 2;
   key.r = KNOWN_VALUE * 16;

   newkey = value;
   isw = value;

   des(inp, key, &newkey, isw, &out);
/*   printf("%u %u\n", out.l, out.r);*/
```

```
  return 0;
}
```

```
/* $Id: recursion.c,v 1.2 2005/04/04 11:34:58 csg Exp $ */

/* Generate an example of recursive code, to see  *
 * how it can be modeled in the scope graph.      */

/* self-recursion  */
int fib(int i)
{
  if(i==0)
    return 1;
  if(i==1)
    return 1;
  return fib(i-1) + fib(i-2);
}

/* mutual recursion */
int anka(int);

int kalle(int i)
{
  if(i<=0)
    return 0;
  else
    return anka(i-1);
}

int anka(int i)
{
  if(i<=0)
    return 1;
  else
    return kalle(i-1);
}

extern volatile int In;

void main(void)
{
  In = fib(10);
}
```