POLITECNICO DI MILANO

The 2017 International Conference on Field-Programmable Technology

Melbourne, Australia, December 11 - 14, 2017

FPT'17
Melbourne

FPT Tutorial

BAMBU: An open-source framework for research in high-level synthesis

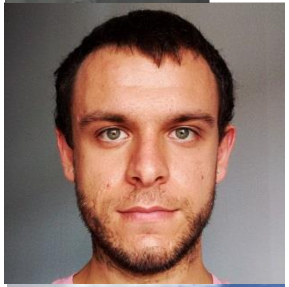
Fabrizio Ferrandi - Christian Pilato

Outline

- ❑ Presentation of bambu
- ❑ Target Customization and Tool Integration
- ❑ Compiler Based Optimizations, Tuning and Customization of Generated Accelerators
- ❑ Synthesis and optimization of memory accesses
- ❑ Debugging and Automated Bug Detection for Hardware Generated with bambu

Material prepared by

- ❑ Christian Pilato
  - Postdoc Researcher– USI Lugano
- ❑ Marco Lattuada
  - Postdoc Researcher – Politecnico di Milano
- ❑ Pietro Fezzardi
  - PhD Student – Politecnico di Milano
- ❑ Fabrizio Ferrandi
  - Associate Professor – Politecnico di Milano



Tutorial Material

- ❑ Slides and the VM with a pre-configured environment can be downloaded from:

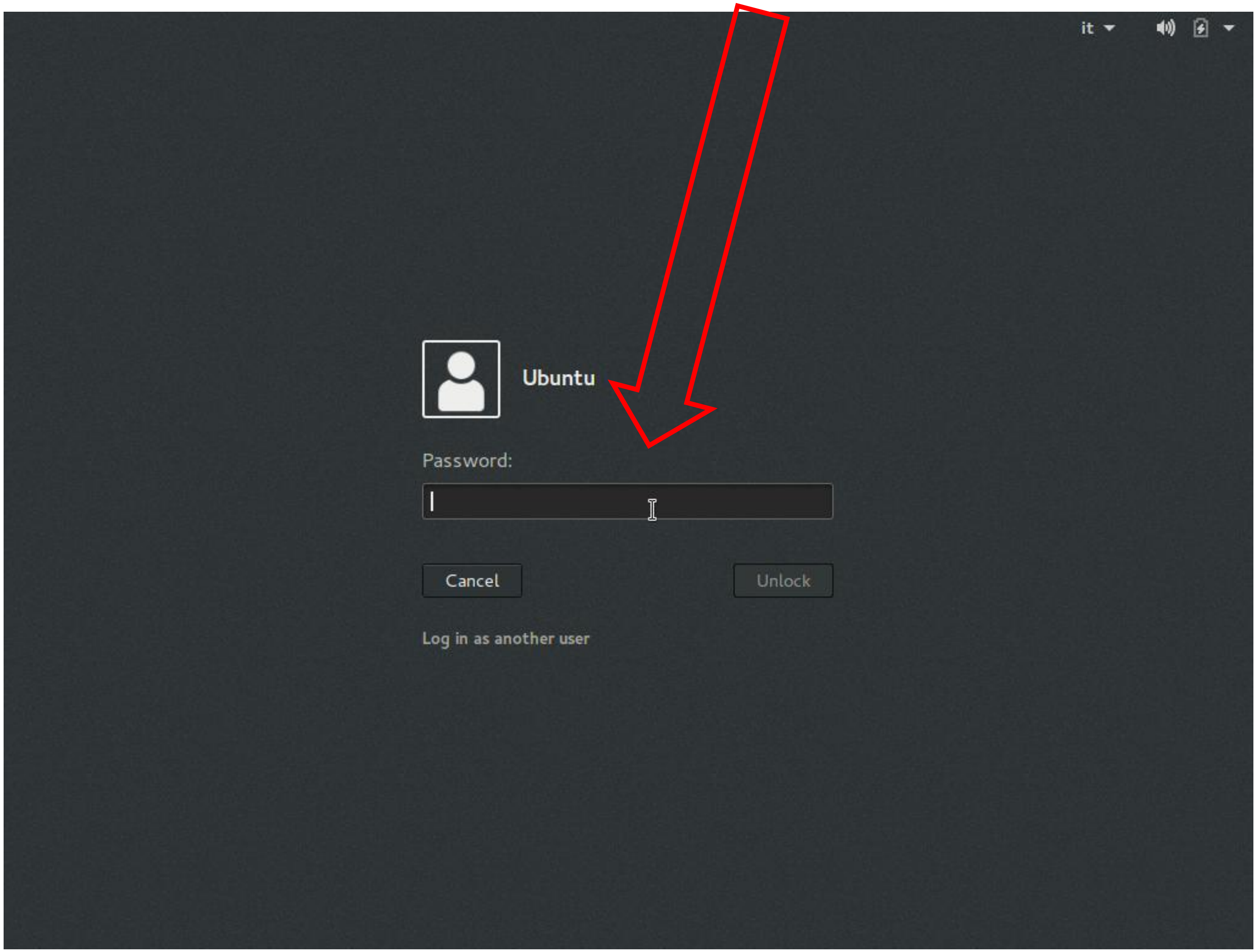
https://panda.dei.polimi.it/?page_id=767

VirtualMachine download&import

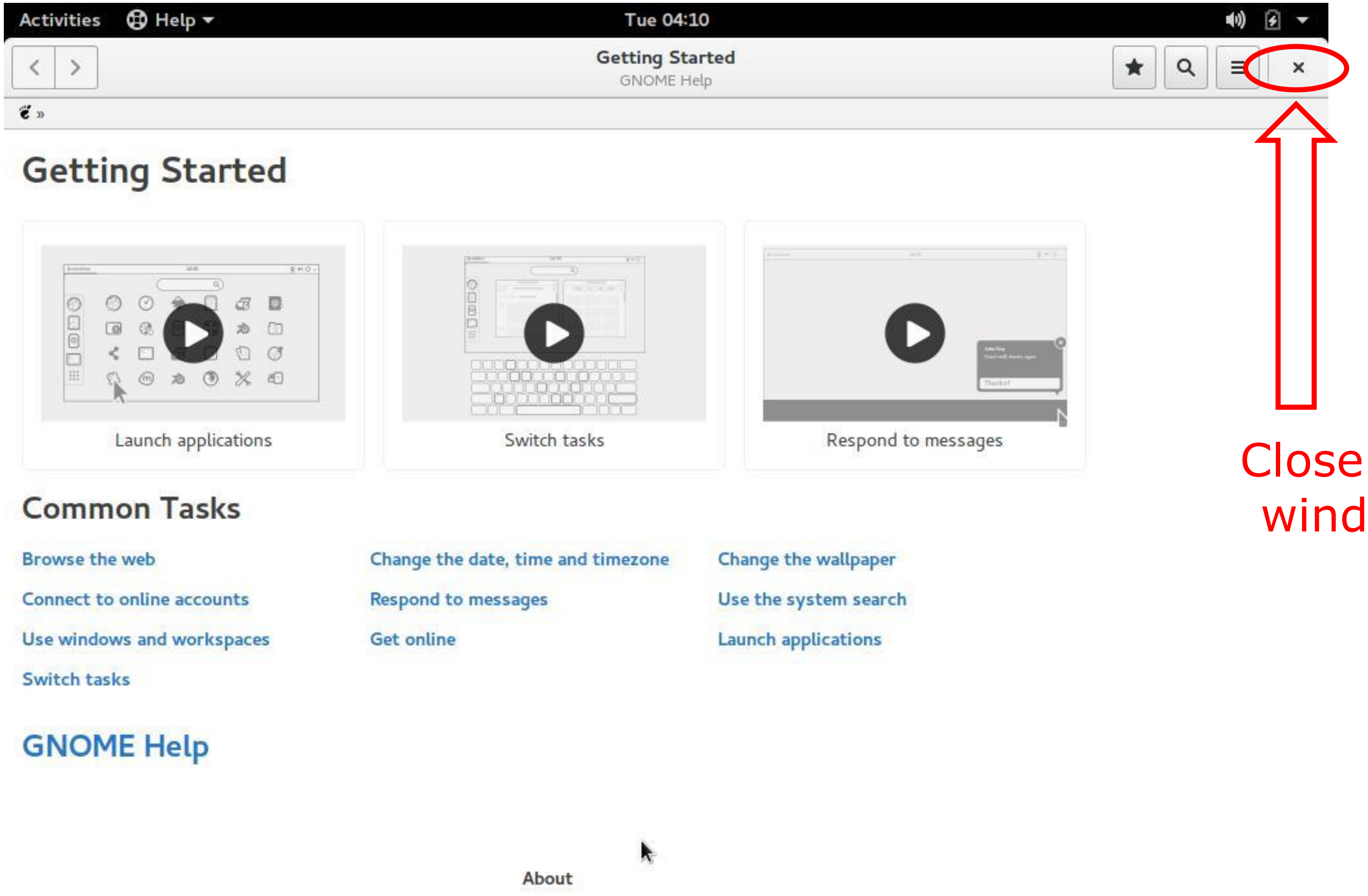
- There are two .ova files. One for 32bit operating systems and another one for 64bit operating systems.
- Once downloaded you can import the virtual machine by clicking on the VirtualBox Manager menu:
  - File
    - Import appliance…

Login

Type as a password: password



Starting screen



Getting Started

Common Tasks

Browse the web
Connect to online accounts
Use windows and workspaces
Switch tasks
Change the date, time and timezone
Respond to messages
Get online
Change the wallpaper
Use the system search
Launch applications

GNOME Help

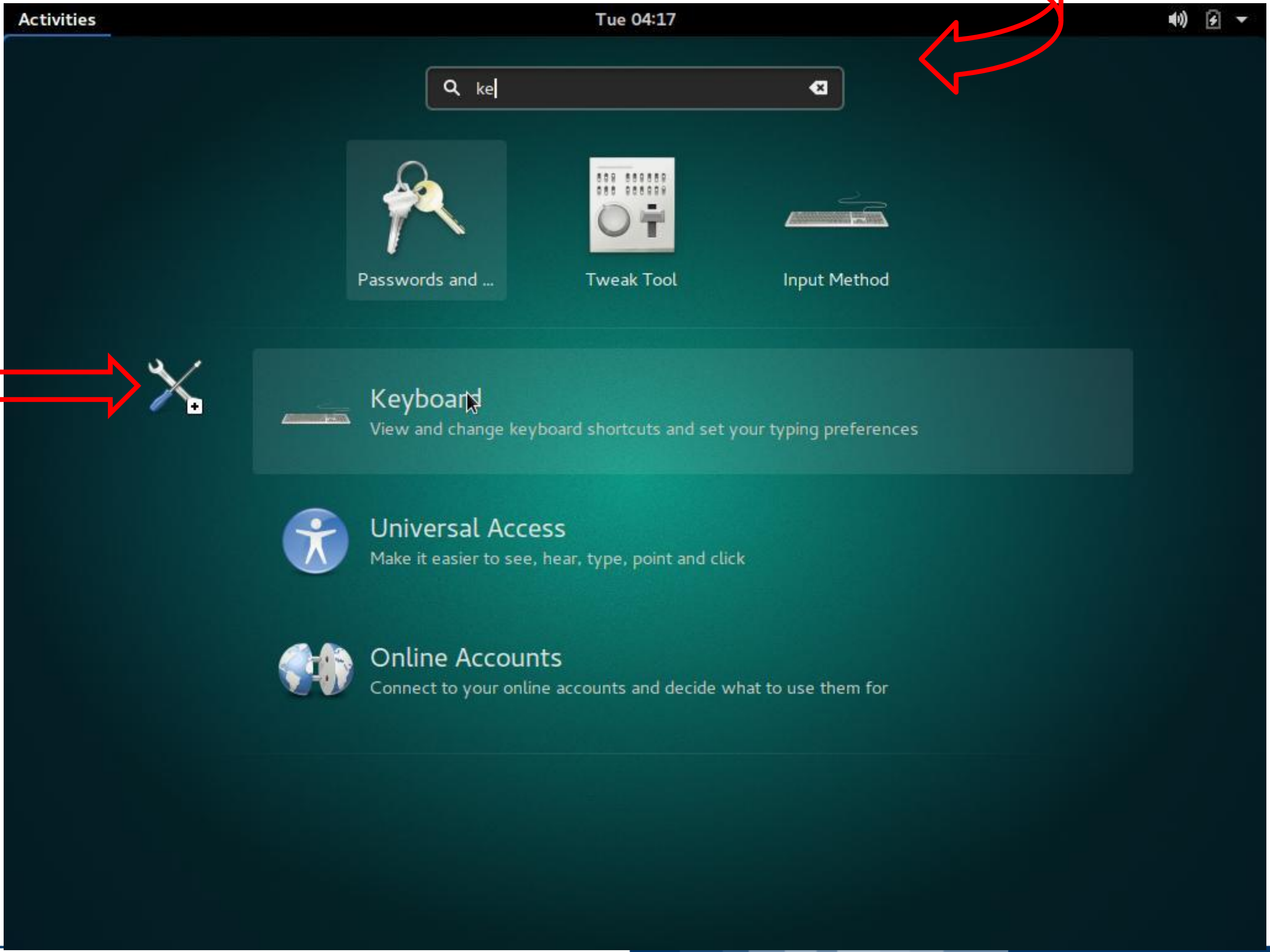
Close the window

Configure the keyboard

1) Click on activities

2) Search for “keyboard”

3) Select keyboard



Activities

Tue 04:17

ke

Passwords and ...

Tweak Tool

Input Method

Keyboard

View and change keyboard shortcuts and set your typing preferences

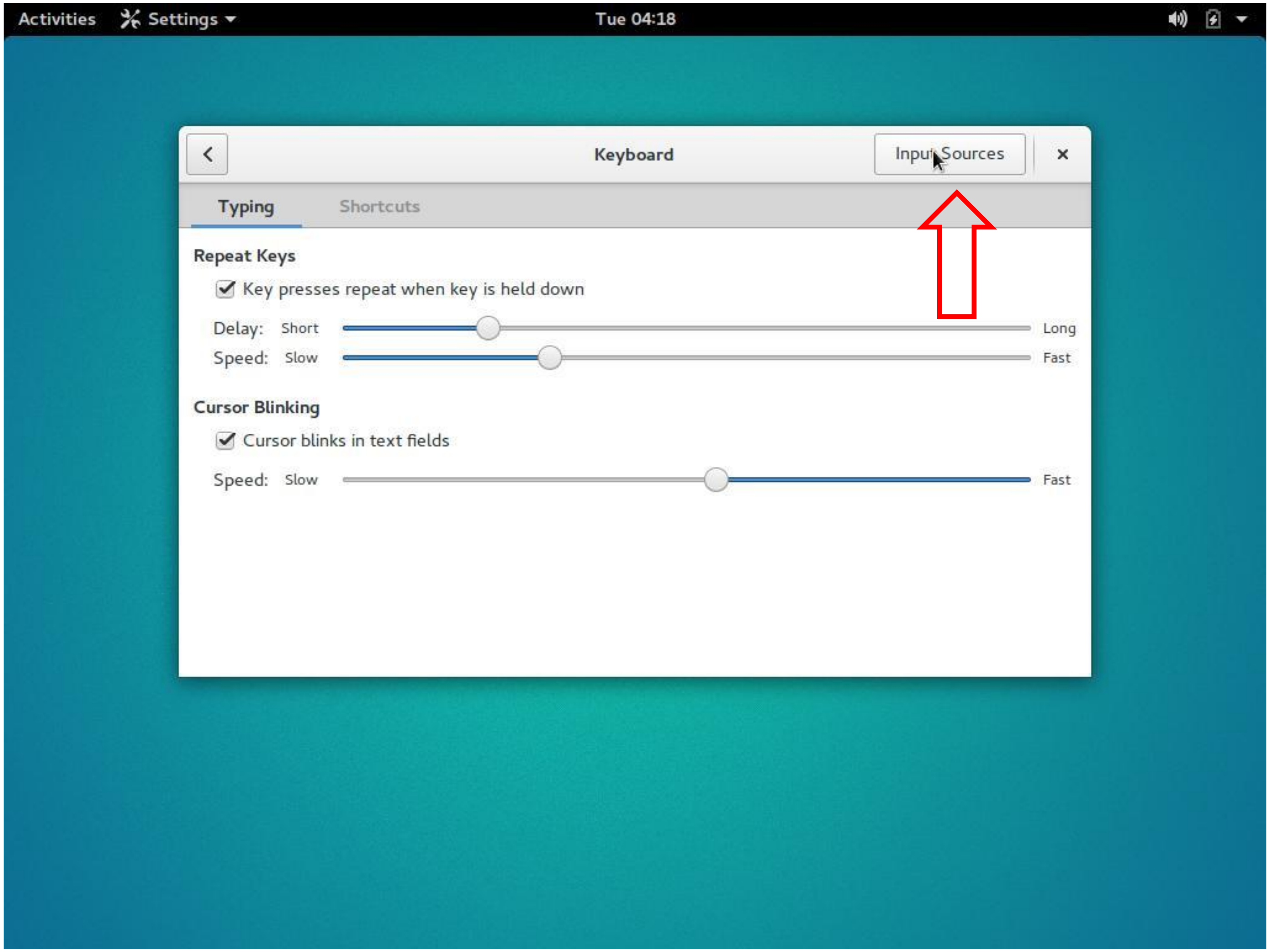
Universal Access

Make it easier to see, hear, type, point and click

Online Accounts

Connect to your online accounts and decide what to use them for

Select the input sources



Repeat Keys

☑ Key presses repeat when key is held down

Delay: Short — Long

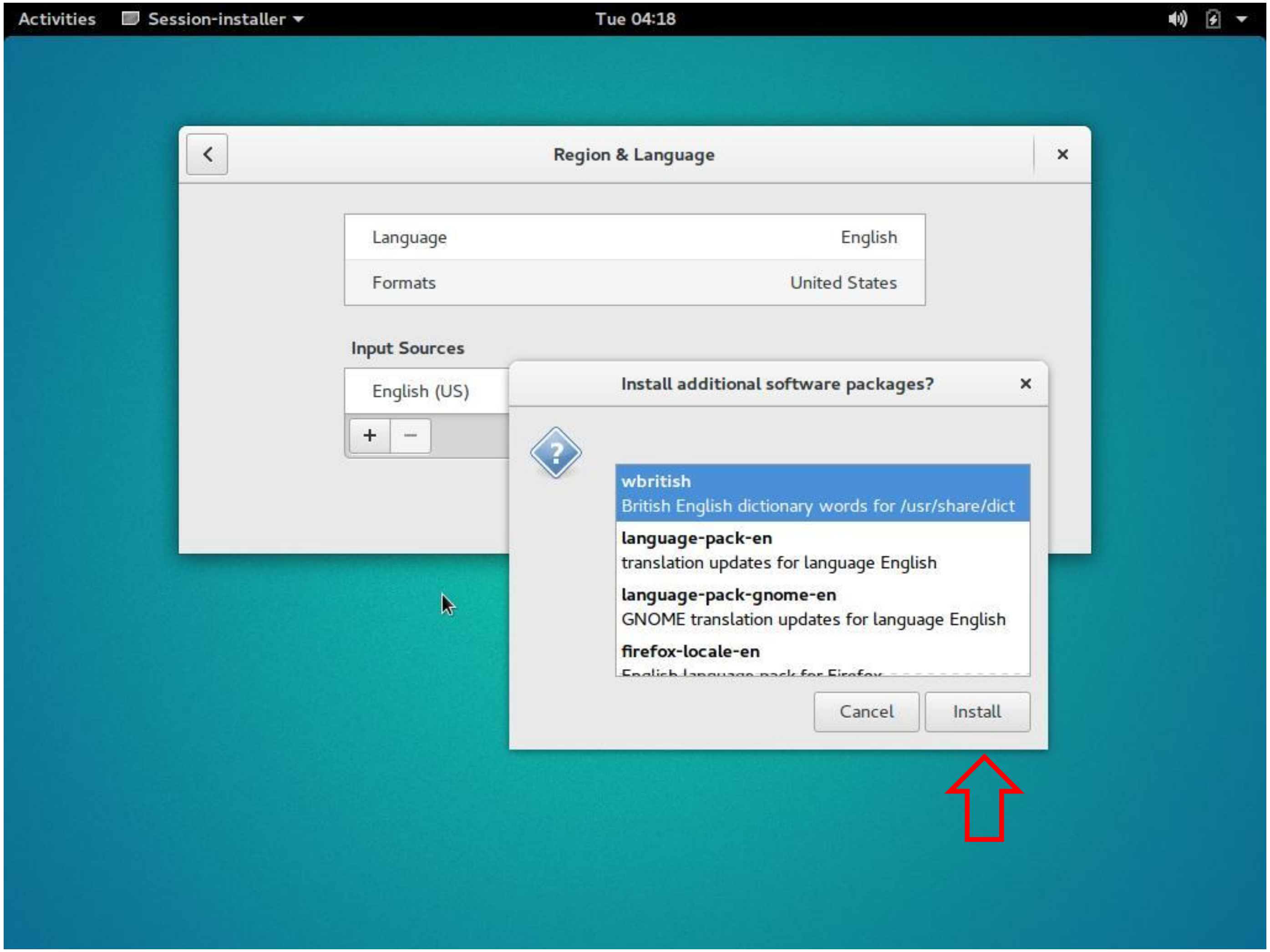
Speed: Slow — Fast

Cursor Blinking

☑ Cursor blinks in text fields

Speed: Slow — Fast

Install additional packages



Language English

Formats United States

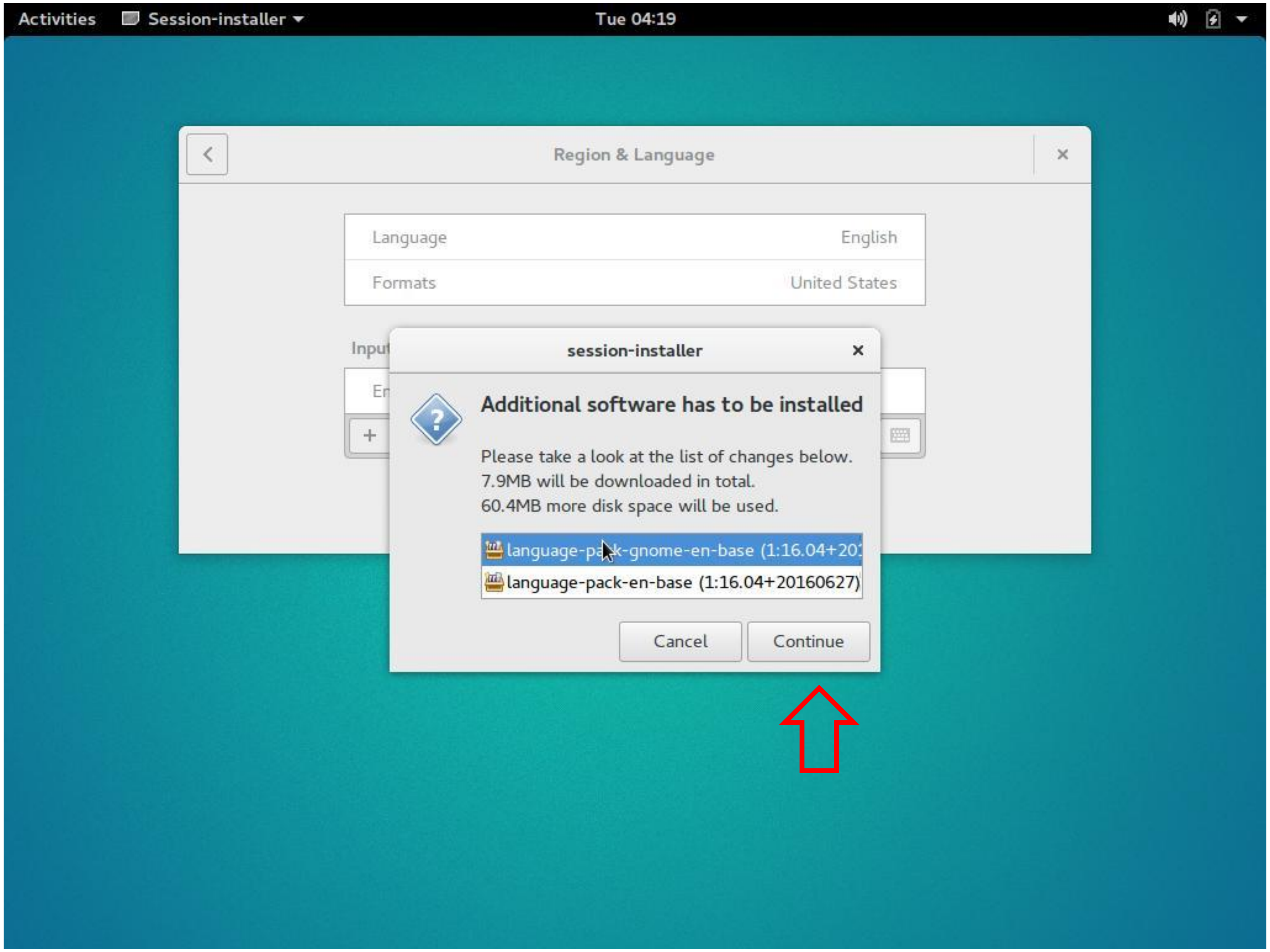
wbritish
British English dictionary words for /usr/share/dict

language-pack-en
translation updates for language English

language-pack-gnome-en
GNOME translation updates for language English

firefox-locale-en
English language pack for Firefox

Confirm the installation



Language English

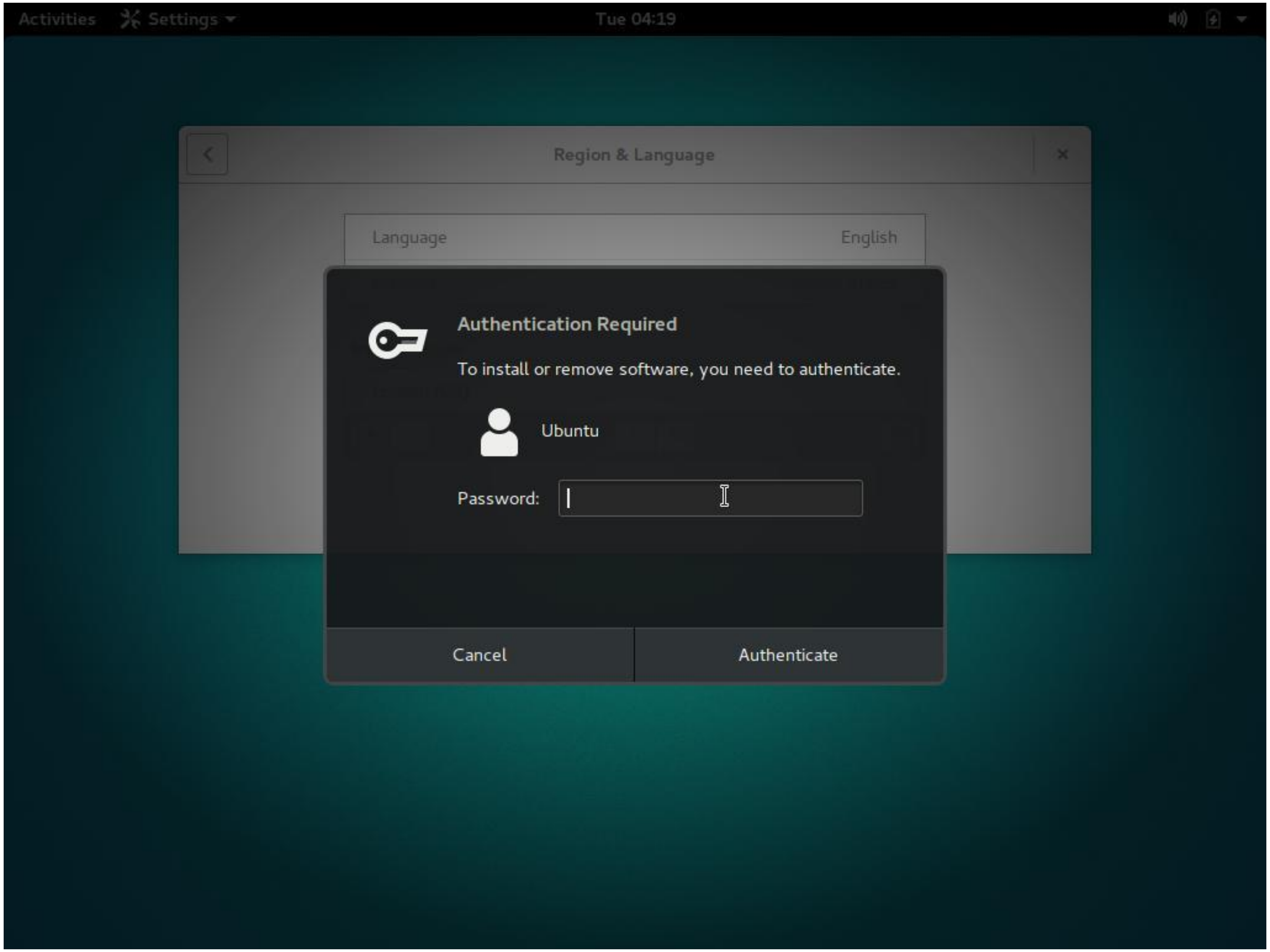
Formats United States

Additional software has to be installed

Please take a look at the list of changes below.
7.9MB will be downloaded in total.
60.4MB more disk space will be used.

language-pack-gnome-en-base (1:16.04+201
language-pack-en-base (1:16.04+20160627)

Type as a password: password



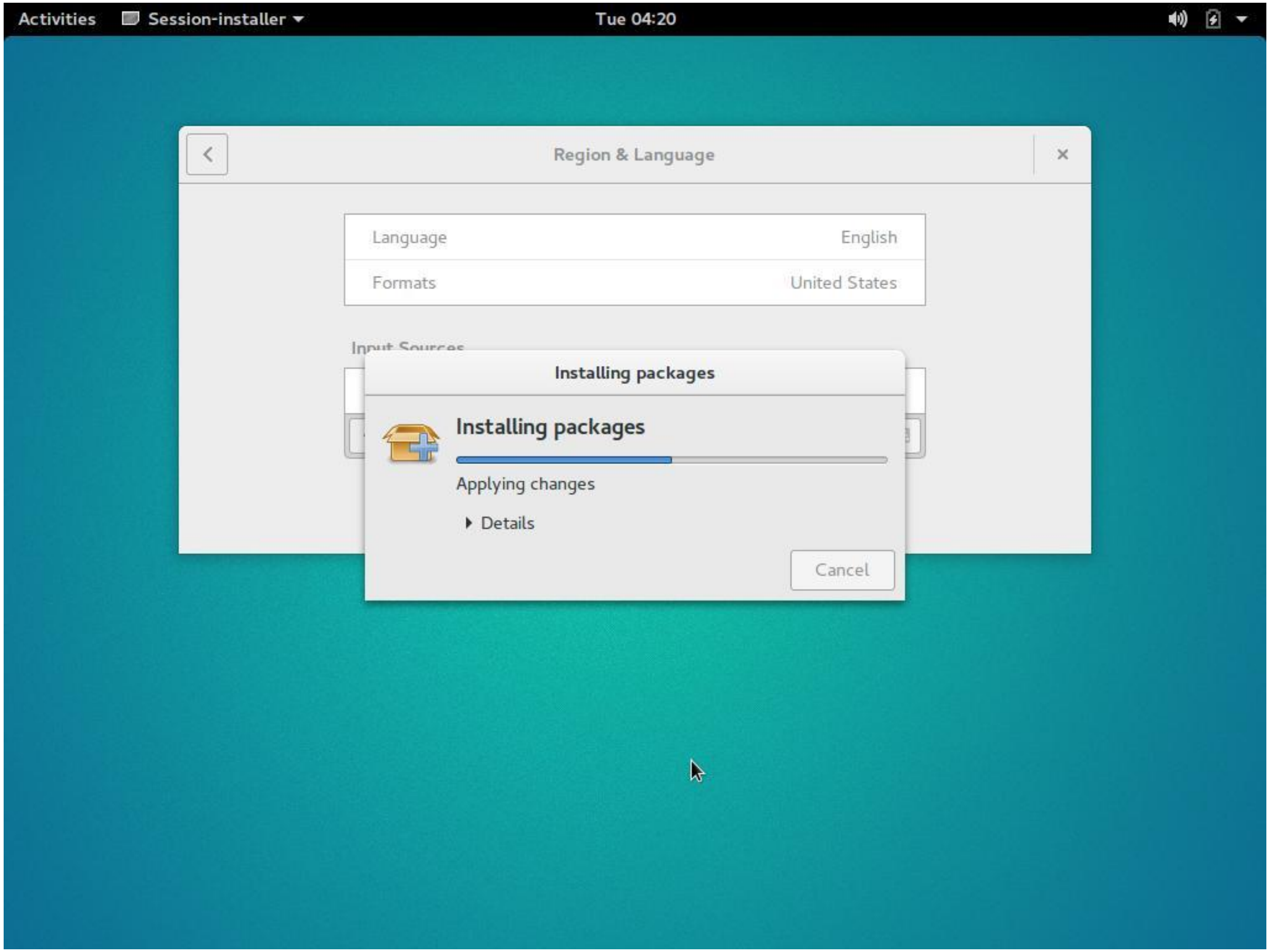
Authentication Required

To install or remove software, you need to authenticate.

Ubuntu

Password:

Wait a while



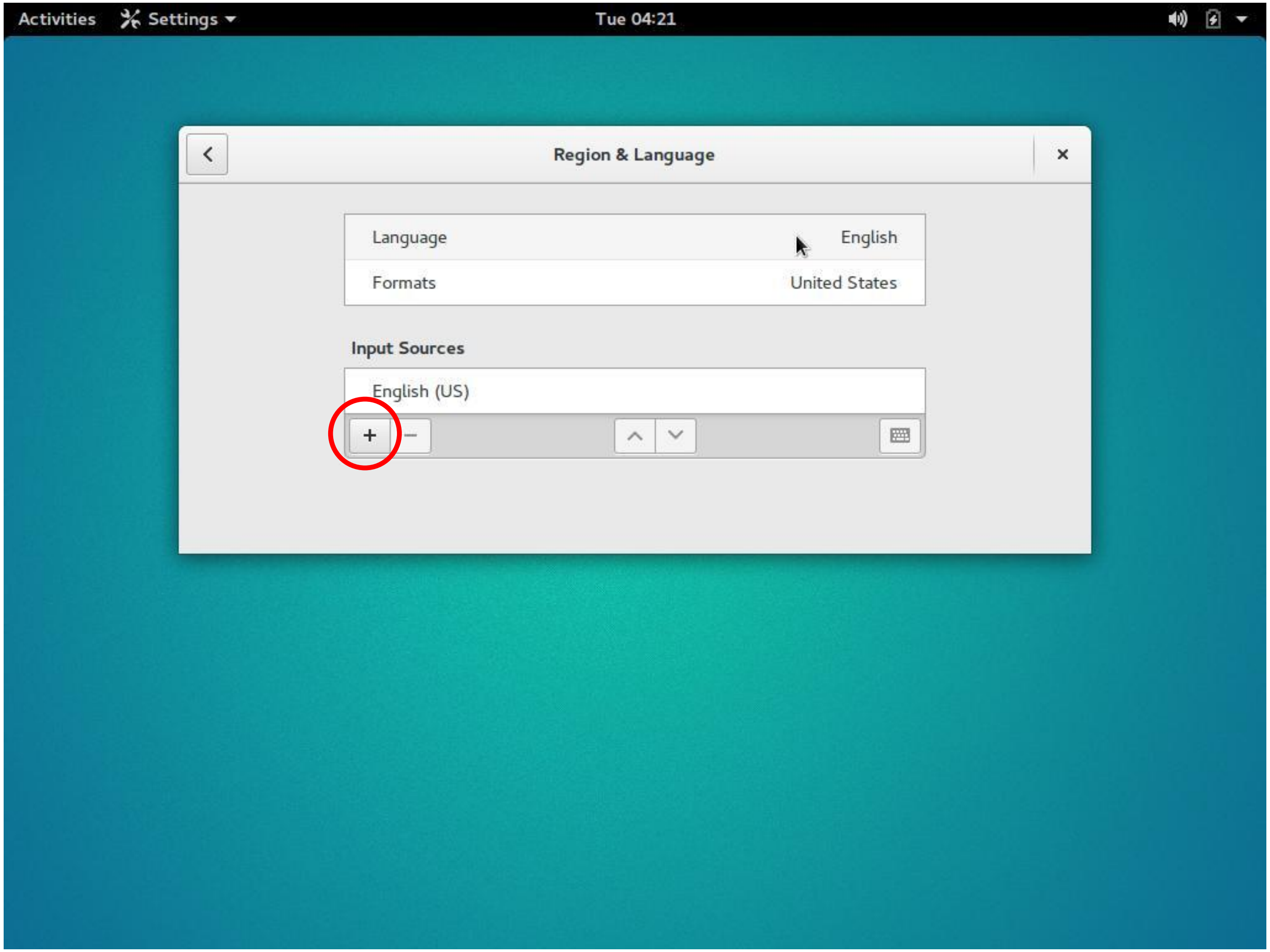
Language English

Formats United States

Installing packages

Applying changes

Add a new input source



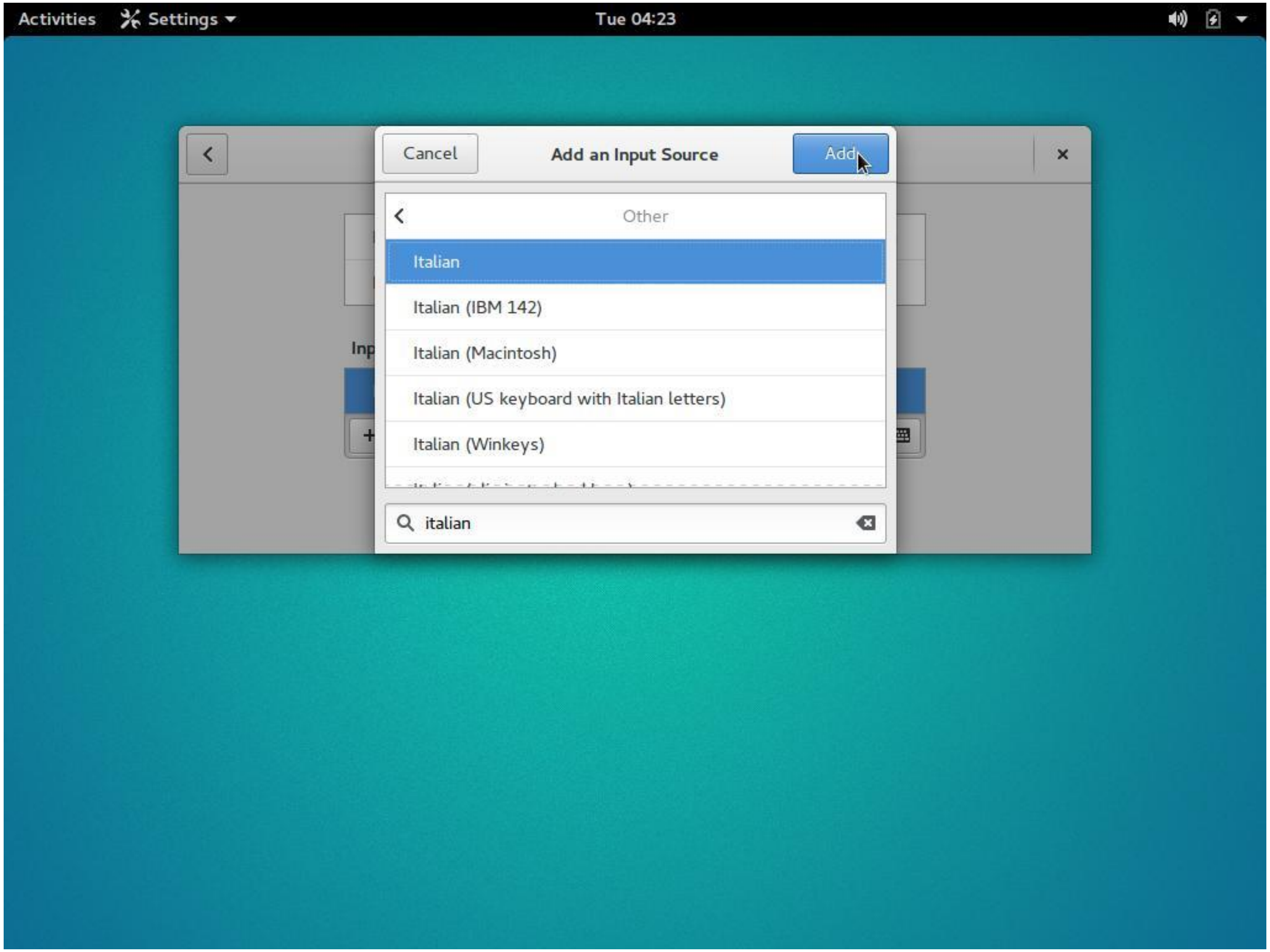
Language English

Formats United States

Input Sources

English (US)

Type the keyboard language



Italian

Italian (IBM 142)

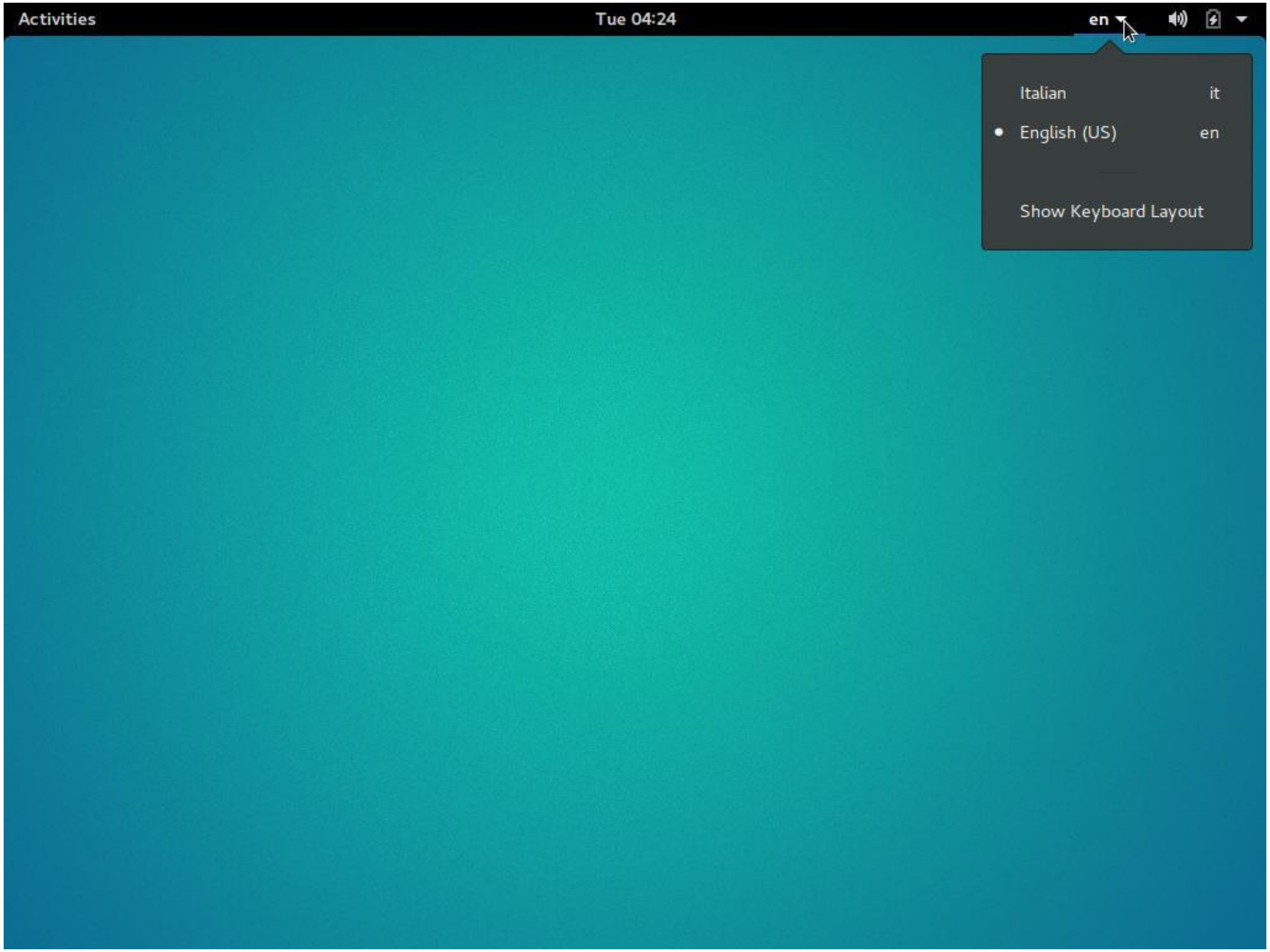
Italian (Macintosh)

Italian (US keyboard with Italian letters)

Italian (Winkeys)

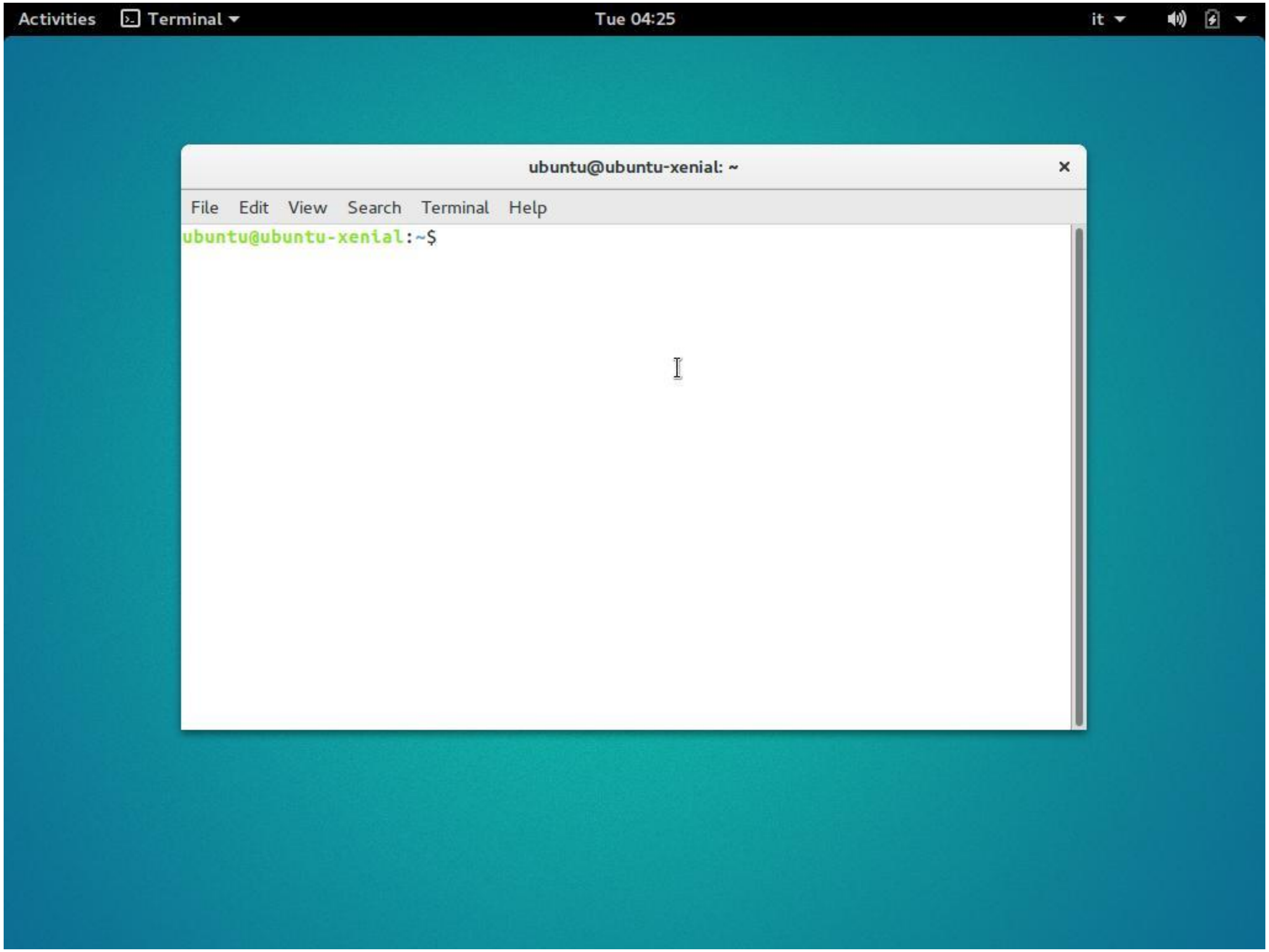
italian

Activate the keyboard layout



Open a terminal

Press simultaneously: ctrl alt t



```
ubuntu@ubuntu-xenial:~$
```

Update the git repository

- ❑ Change the directory to:

```
cd ~/PandA-bambu
```

- ❑ And then

```
git pull
```

- ❑ Switch to the FPT branch:

```
git checkout FPT17_tutorial
```

First synthesis

- Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/first
```

- Edit the file C by typing:

```
gedit module.c
```

```c
unsigned short icrc1(unsigned short crc, unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);
    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

Run bambu

- ❑ Run the script by typing:

```
./bambu.sh
```

```
 Summary of resources:
     - ASSIGN_SIGNED_FU: 2
     - IIconvert_expr_FU: 1
     - IUdata_converter_FU: 2
     - MUX_GATE: 2
     - UIdata_converter_FU: 3
     - UUdata_converter_FU: 1
     - bit_xor_expr_FU: 1
     - constant_value: 5
     - ge_expr_FU: 1
     - lshift_expr_FU: 1
     - ne_expr_FU: 1
     - plus_expr_FU: 1
     - read_cond_FU: 1
     - register_SE: 2
     - ui_bit_xor_expr_FU: 1
     - ui_cond_expr_FU: 1
     - ui_lshift_expr_FU: 1
Start reading vector           1's values from input file.

Value found for input crc: 0000000000001010
Value found for input onech: 00000010
Reading of vector values from input file completed. Simulation started.
Value found for output ex_return_port: 0010101001000010
 return_port = 10818   expected = 10818

Simulation ended after                   10 cycles.

Simulation completed with success

- HLS_output//simulation/testbench_icrc1_minimal_interface_tb.v:441: Verilog $finish
1. Simulation completed with SUCCESS; Execution time 10 cycles;
  Total cycles             : 10 cycles
  Number of executions     : 1
  Average execution        : 10 cycles
```

Simulation & Synthesis

- ❑ Testbench generated automatically
  - test_icrc1.xml
- ❑ Simulation and synthesis scripts generated automatically:
  - icrc1/simulate_icrc1_minimal_interface.sh
  - icrc1/synthesize_Synthesis_icrc1_minimal_interface.sh
- ❑ Verilog file generated at the end of the HLS step:
  - icrc1/icrc1.v

Analyze results

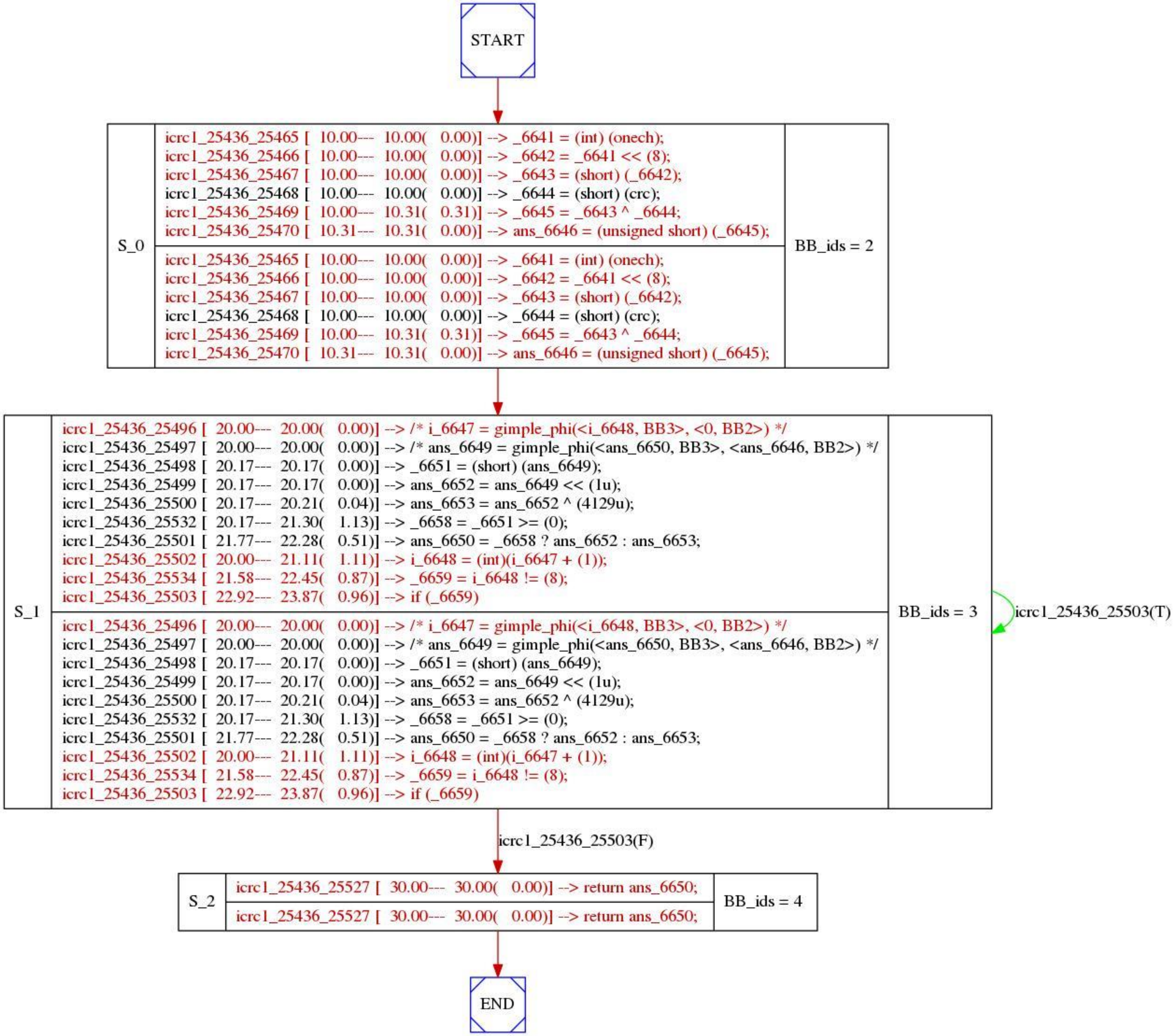
- ❑ Change directory to icrc1:

```
cd icrc1
```

- ❑ Display the FSM:

```
xdot HLS_output/dot/icrc1/HLS_STGraph.dot
```

FSM



```
icrc1_25436_25465 [ 10.00--- 10.00(  0.00)] --> _6641 = (int) (onech);
icrc1_25436_25466 [ 10.00--- 10.00(  0.00)] --> _6642 = _6641 << (8);
icrc1_25436_25467 [ 10.00--- 10.00(  0.00)] --> _6643 = (short) (_6642);
icrc1_25436_25468 [ 10.00--- 10.00(  0.00)] --> _6644 = (short) (crc);
icrc1_25436_25469 [ 10.00--- 10.31(  0.31)] --> _6645 = _6643 ^ _6644;
icrc1_25436_25470 [ 10.31--- 10.31(  0.00)] --> ans_6646 = (unsigned short) (_6645);
```

```
icrc1_25436_25465 [ 10.00--- 10.00(  0.00)] --> _6641 = (int) (onech);
icrc1_25436_25466 [ 10.00--- 10.00(  0.00)] --> _6642 = _6641 << (8);
icrc1_25436_25467 [ 10.00--- 10.00(  0.00)] --> _6643 = (short) (_6642);
icrc1_25436_25468 [ 10.00--- 10.00(  0.00)] --> _6644 = (short) (crc);
icrc1_25436_25469 [ 10.00--- 10.31(  0.31)] --> _6645 = _6643 ^ _6644;
icrc1_25436_25470 [ 10.31--- 10.31(  0.00)] --> ans_6646 = (unsigned short) (_6645);
```

```
icrc1_25436_25496 [ 20.00--- 20.00(  0.00)] --> /* i_6647 = gimple_phi(<i_6648, BB3>, <0, BB2>) */
icrc1_25436_25497 [ 20.00--- 20.00(  0.00)] --> /* ans_6649 = gimple_phi(<ans_6650, BB3>, <ans_6646, BB2>) */
icrc1_25436_25498 [ 20.17--- 20.17(  0.00)] --> _6651 = (short) (ans_6649);
icrc1_25436_25499 [ 20.17--- 20.17(  0.00)] --> ans_6652 = ans_6649 << (1u);
icrc1_25436_25500 [ 20.17--- 20.21(  0.04)] --> ans_6653 = ans_6652 ^ (4129u);
icrc1_25436_25532 [ 20.17--- 21.30(  1.13)] --> _6658 = _6651 >= (0);
icrc1_25436_25501 [ 21.77--- 22.28(  0.51)] --> ans_6650 = _6658 ? ans_6652 : ans_6653;
icrc1_25436_25502 [ 20.00--- 21.11(  1.11)] --> i_6648 = (int)(i_6647 + (1));
icrc1_25436_25534 [ 21.58--- 22.45(  0.87)] --> _6659 = i_6648 != (8);
icrc1_25436_25503 [ 22.92--- 23.87(  0.96)] --> if (_6659)
```

```
icrc1_25436_25496 [ 20.00--- 20.00(  0.00)] --> /* i_6647 = gimple_phi(<i_6648, BB3>, <0, BB2>) */
icrc1_25436_25497 [ 20.00--- 20.00(  0.00)] --> /* ans_6649 = gimple_phi(<ans_6650, BB3>, <ans_6646, BB2>) */
icrc1_25436_25498 [ 20.17--- 20.17(  0.00)] --> _6651 = (short) (ans_6649);
icrc1_25436_25499 [ 20.17--- 20.17(  0.00)] --> ans_6652 = ans_6649 << (1u);
icrc1_25436_25500 [ 20.17--- 20.21(  0.04)] --> ans_6653 = ans_6652 ^ (4129u);
icrc1_25436_25532 [ 20.17--- 21.30(  1.13)] --> _6658 = _6651 >= (0);
icrc1_25436_25501 [ 21.77--- 22.28(  0.51)] --> ans_6650 = _6658 ? ans_6652 : ans_6653;
icrc1_25436_25502 [ 20.00--- 21.11(  1.11)] --> i_6648 = (int)(i_6647 + (1));
icrc1_25436_25534 [ 21.58--- 22.45(  0.87)] --> _6659 = i_6648 != (8);
icrc1_25436_25503 [ 22.92--- 23.87(  0.96)] --> if (_6659)
```

```
icrc1_25436_25527 [ 30.00--- 30.00(  0.00)] --> return ans_6650;
```

```
icrc1_25436_25527 [ 30.00--- 30.00(  0.00)] --> return ans_6650;
```

Open the RTL Verilog

```
module icrc1_minimal_interface(clock, reset, start_port, crc, onech,
done_port, return_port);
  // IN
  input clock;
  input reset;
  input start_port;
  input [15:0] crc;
  input [7:0] onech;
  // OUT
  output done_port;
  output [15:0] return_port;
  // Component and signal declarations

  icrc1 icrc1_i0 (.done_port(done_port), .return_port(return_port),
.clock(clock), .reset(reset), .start_port(start_port), .crc(crc),
.onech(onech));

endmodule
```

Bambu: an example of modern HLS tools

- HLS tool developed at Politecnico di Milano (Italy) within the PandA framework
  - Available under GPL v3 at
    - http://panda.dei.polimi.it/
    - https://github.com/ferrandi/PandA-bambu
- Example features
  - Front-end Input: interfacing with GCC for parsing C code
    - Complete support for ANSI C (except for recursion)
      - Support for pointers, user-defined data types, built-in C functions, etc..
    - Source code optimizations
      - may alias analysis, dead-code elimination, hoisting, loop optimizations, etc...
  - Target-aware synthesis
    - Characterization of the technology library based on target device
  - Verification
    - Integrated testbench generation and simulation
      - automated interaction with Iverilog, Verilator, Xilinx Isim, Xilinx Xsim, Mentor Modelsim
  - Back-end: Automated interaction with commercial synthesis tools
    - FPGA: Xilinx ISE, Xilinx Vivado, Altera Quartus, Lattice Diamond
    - ASIC: Synopsis Design Compiler

A bit of history

- PandA framework development started on 2004 as a support research infrastructure for PoliMi in the context of ICODES – FP6-IST EU-funded project
  - Parsing and analysis of TLM 2.0 SystemC descriptions (gcc v.3.5)
- In the hArtes EU-funded project (2006-2010), it was used to
  - Analyzing generic C-based application annotated with pragmas (OpenMP)
  - Extracting parallel tasks
  - Estimating performance of embedded app
  - C-to-c rewriting
- Later, in Synaptic (2009-2013) and in Faster (2011-2014) EU-funded projects, logic- and high-level synthesis has been extended
  - Bambu (HLS tool) was first released in March 2012.
- ESA funded many research on code predictability analysis, performance analysis, and integration of HLS in model-based design flows.

Why reinventing the wheel?

- From LegUp 1.0:

> Well, it turns out that none of the existing high-level synthesis tools have source code available for researchers. GAUT claims to be open-source but the code is not available for download. xPilot from UCLA is an advanced research tool but only the binary is available and it hasn't been updated since 2007. ROCCC provides an open source eclipse plugin based on SUIF and LLVM but only supports small C programs. Standard C code must be rewritten to work with ROCCC because all function parameters must be structs. Trident uses a very old version of LLVM to interface with an extensive amount of Java code, but unfortunately no longer compiles with the latest version of LLVM.

Why free SW or free HW

1. Are we classical computer hackers, skilled in SW development and with 100% control of our PCs and workstations?
   - Not too surprisingly, that is the perspective from which RMS wrote the GPL.
   - RMS defined freedom as the ability of people like himself to get read-write access to code.
   - How does this definition map to today's world?
   - How much transparent are today's systems?
     - HW vs SW
2. Funding of PandA comes mainly from public institutions

3. … because it's fun

Some References Inspiring Us

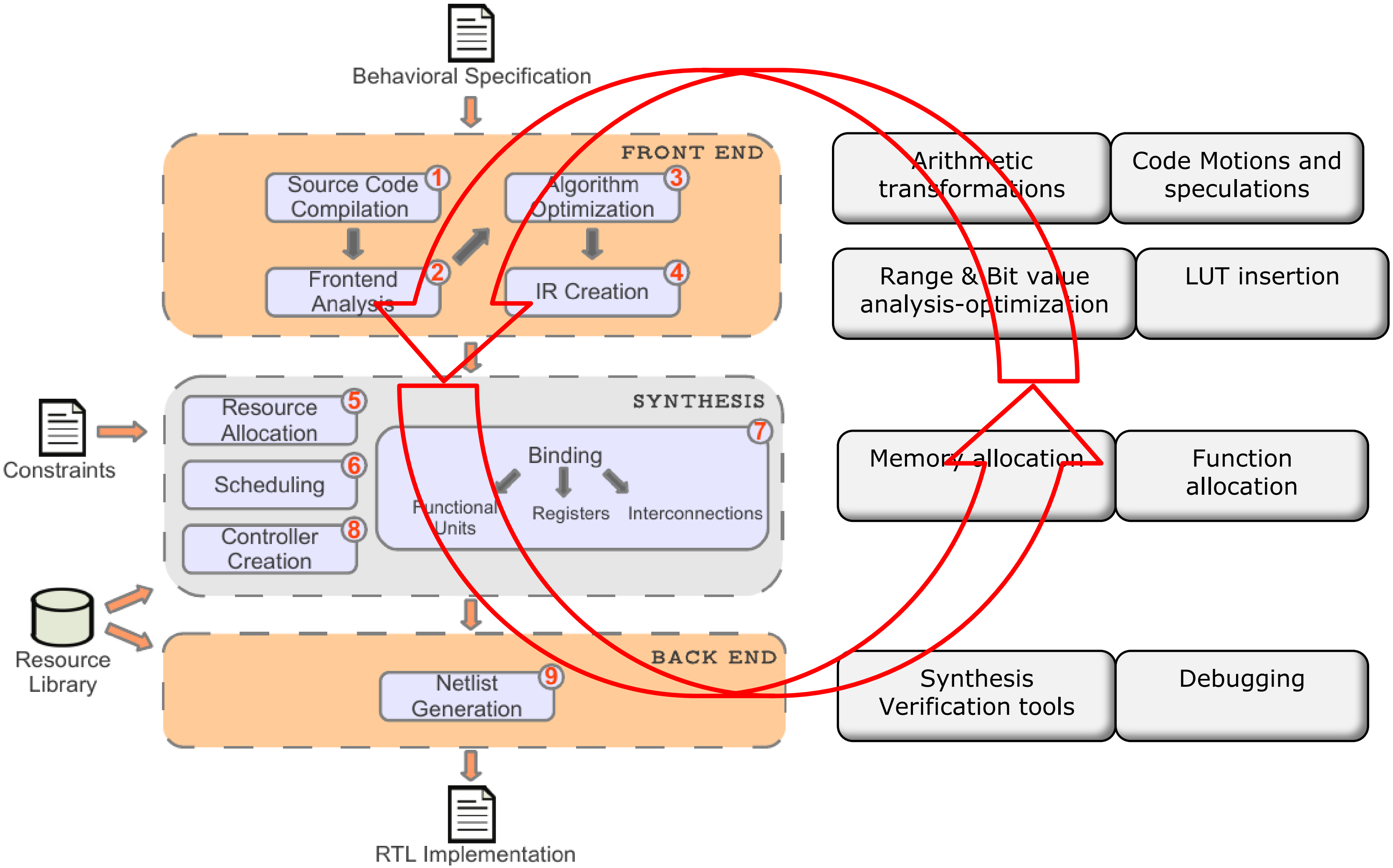
- ❑ Leon Stok, “Data path synthesis”, Integration VLSI Journal, 18(1): 1-71 (1994).
- ❑ Jason Cong, Zhiru Zhang, ”An efficient and versatile scheduling algorithm based on SDC formulation”, DAC 2006, 433-438.
- ❑ Florent de Dinechin, Bogdan Pasca: “Designing Custom Arithmetic Data Paths with FloPoCo”. IEEE Design & Test of Computers 28(4), 18-27 (2011).

- ❑ Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck, “Efficiently Computing Static Single Assignment Form and the Control Dependence Graph”, ACM Trans. Program. Lang. Syst. 13(4), 451-490 (1991).
- ❑ Preston Briggs, Keith D. Cooper, Timothy J. Harvey, L. Taylor Simpson, “Practical Improvements to the Construction and Destruction of Static Single Assignment Form”, Softw., Pract. Exper. 28(8), 859-881 (1998).
- ❑ Benoit Boissinot, Florian Brandner, Alain Darte, Benoît Dupont de Dinechin, Fabrice Rastello, “A Non-iterative Data-Flow Algorithm for Computing Liveness Sets in Strict SSA Programs”. APLAS 2011, 137-154.

Synthesis per function

- ❑ one component per function
  - ▶ function interface
  - ▶ start and done
  - ▶ parameter passing
    - wires
    - memory interaction
- ❑ hierarchy based on call graph
  - ▶ no-recursion
  - ▶ proxy
- ❑ option
  - ▶ `-fwhole-program`

Bambu Overview



Behavioral Specification

FRONT END

Source Code Compilation 1

Frontend Analysis 2

Algorithm Optimization 3

IR Creation 4

Constraints

SYNTHESIS

Resource Allocation 5

Scheduling 6

Controller Creation 8

Binding 7

Functional Units

Registers

Interconnections

Resource Library

BACK END

Netlist Generation 9

RTL Implementation

Arithmetic transformations

Code Motions and speculations

Range & Bit value analysis-optimization

LUT insertion

Memory allocation

Function allocation

Synthesis Verification tools

Debugging

Modular Framework based on the specialization of **HLS_step**

Bambu: command-line interface

- Minimal command
  ```
  $ bambu filename.c
  ```
- Controlling the clock period (100Mhz)
  ```
  $ bambu filename.c --clock-period=10
  ```
- Select the device
  ```
  $ bambu filename.c -device-name=xc7z020,-1,clg484,VVD
  ```

Subset of synthesizable C (1)

- We support the standard GCC support
- Supported features:
  - Expressions of any kind: arithmetic, logical, bitwise, relational, conditional, comma-based expressions.
  - Types: integers, single- and double-precision floating point, _Bool and Complex, struct-or-union, bitfields, enum, typedef, pointers and arrays, type qualifiers.
  - Variable declarations, initialization, storage-specifiers
  - Functions definition and declaration, extern or static, pointer to functions, parameters passed by copy or reference, tail recursive functions.
  - Statements and blocks: labeled (case), compound, expression, selection (if,switch), iteration(while,do,for), jump (goto,continue,break,return)
  - All preprocessor directives

  - Unaligned memory accesses and dynamic pointers resolution
  - GCC vectorization

Subset not supported

- struct returned by copy
- Non-tailing recursive functions

Second example

- Search and insertion in a binary tree
- Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/second
```

- Edit the file C by typing:

```
gedit module.c
```

  - Two data structures: stack and binary tree
  - Static memory allocators
  - Tail recursive functions
  - Use of pointer to pointers (some HLSs have problems)

libbambu

- assert, puts, putchar, read, open, close, write, printf, exit, abort
- libc functions: bswap32,memcmp, memcpy, memmove, memset, malloc, free, memalign, alloca_with_align, calloc, bcopy, bzero, memchr, mempcpy, memrchr, rawmemchr, stpcpy, stpncpy, strcasecmp, strcasestr, strcat, strchr, strchrnul, strcmp, strcpy, strcspn, strdup, strlen, strncasecmp, strncat, strncmp, strncpy, strndup, strnlen, strpbrk, strrchr, strsep, strspn, strstr, strtok
- libm functions: acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, ceil, cexpi, copysign, cos, cosh, drem, erf, exp, exp10, expm1, fabs, fdim, finite, floor, lfloor, fma, fmax, fmin, fmod, fpclassify, frexp, gamma, lgamma, tgamma, hypot, ilogb, infinity, isinf, isnan, j0, j1, jn, ldexp, log, log2, log10, log1p, modf, nan, nearbyint, nextafter, pow, pow10, remainder, remquo, rint, lrint, llrint, round, lround, llround, scalb, scalbln, scalbn, signbit, significand, sin, sincos, sinh, sqrt, tan, tanh, trunc.

Third example

- ❑ Crypto core built composing user defined libraries
- ❑ Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/third
```

- ❑ Run the script by typing:

```
./multi.sh
```

- ❑ `tree-panda-gcc` could be used to create a custom library that could be deployed by `bambu` by passing `-l` and `-L` options.

Autotools based project described in directory
`examples/crypto_designs/multi-keccak`

synthesis of multiple files

- `bambu` uses the text-based IR and the builtin linker to build a single in-memory representation and perform HLS

  ```
  $ bambu file1.c file2.c --top-fname=fname
  ```

- In case the option `--top-fname` is not used, `bambu` automatically identifies which function can act as the top one.

Synthesis per function: interface

- ❑ Based on C description
  - ▶ Global variables
  - ▶ Top function parameters
- ❑ No restriction on top function signature
- ❑ C semantic
  - ▶ Scalar parameters: input
  - ▶ struct/union parameters: input
  - ▶ array/pointer: input/output
  - ▶ return value: output
  - ▶ Global variables: input/output

Interface analysis

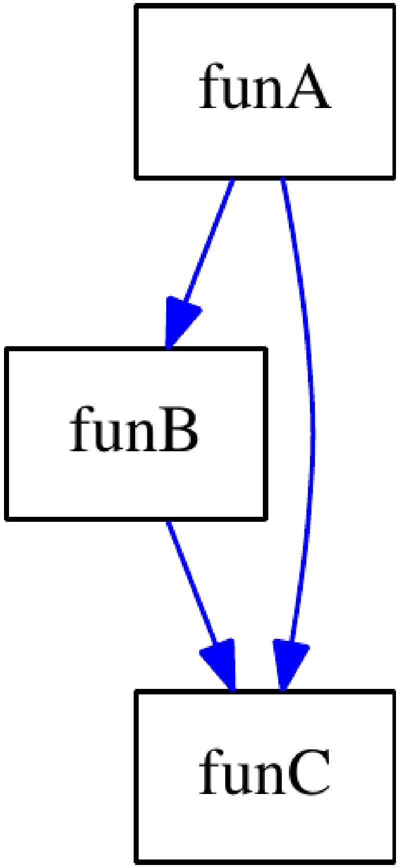
```
int global;

float my_top_function(int * first, struct s second, int
third[10], float * fourth)
{…}
```

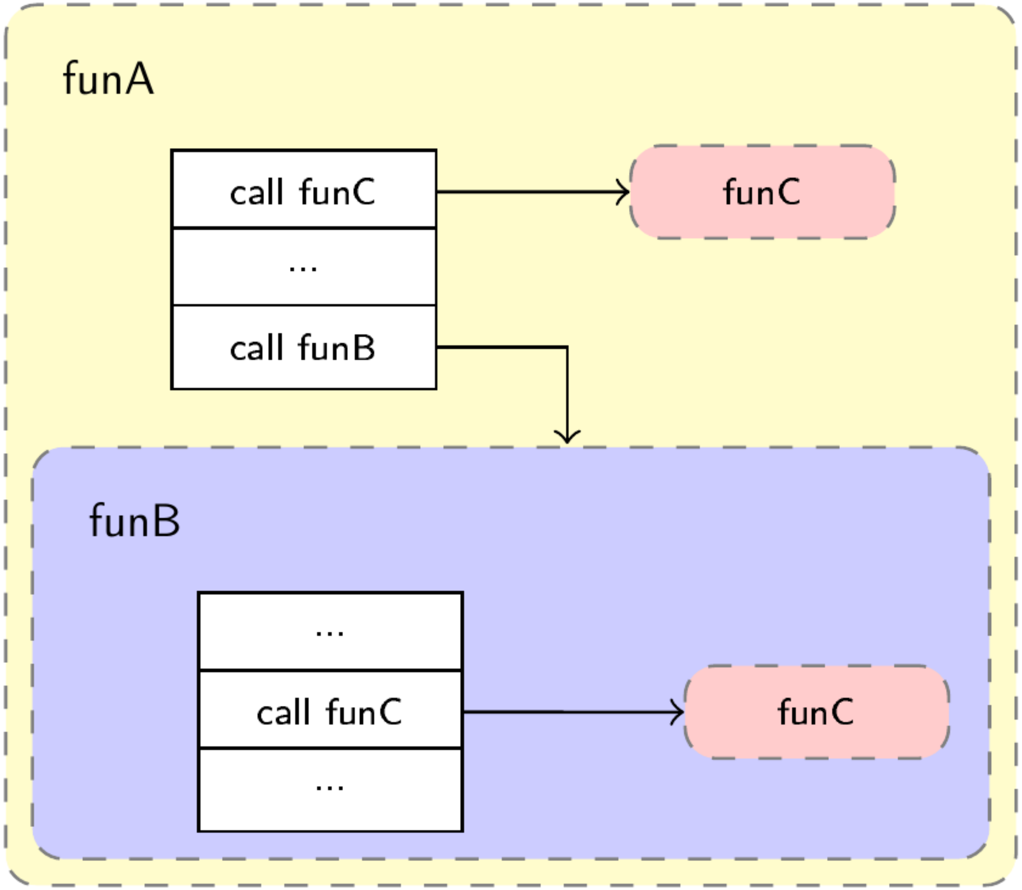
- Input parameters:
  - Second
- Input/ouptut parameters:
  - First, third, fourth
- Ouptut parameters:
  - Return value
- Shared memory:
  - global

Synthesis per function: hierarchy

- One component per function

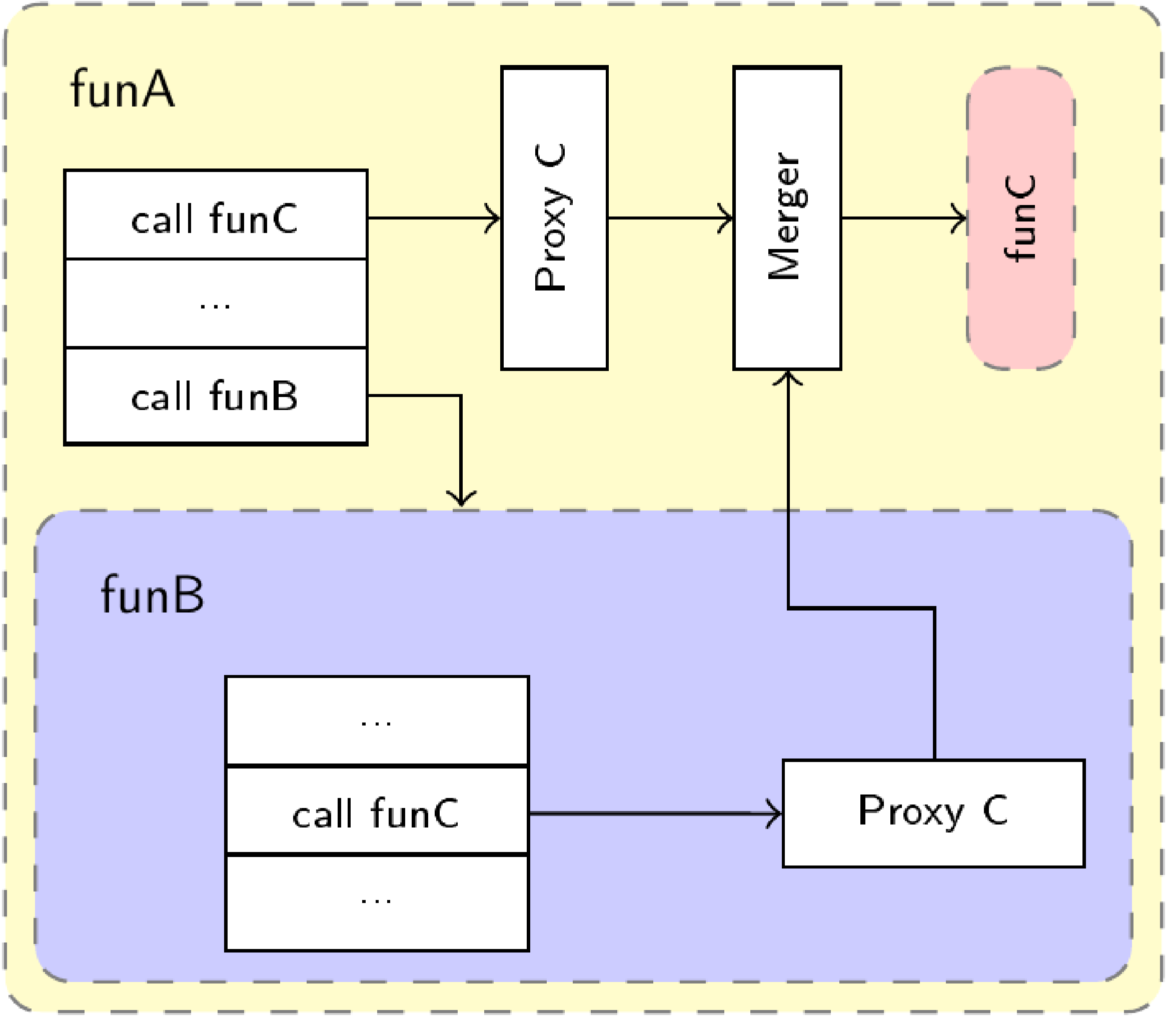


call graph



RTL hierarchy

Synthesis per function: proxy



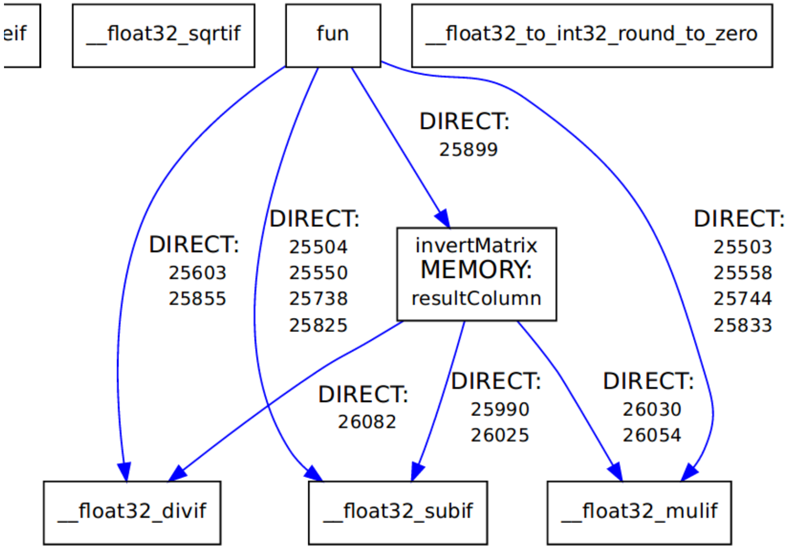
Fourth example

- Lu-decomposition with single precision floating point arithmetic

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/fourth
```

- Run the script by typing:

```
./bambu.sh
```



Option: `--do-not-expose-globals`

- bambu assumes that all global variables could be accessed by external CPU/accelerators
- It is possible to change this default with option:
  - `--do-not-expose-globals`
  - All global variables are considered local to the compilation units as it they are declared `static`.

Option `--fwhole-program`

- From GCC man:
  - Assume that the current compilation unit represents the whole program being compiled. All public functions and variables with the exception of "main" and those merged by attribute "externally_visible" become static functions and in effect are optimized more aggressively by interprocedural optimizers.

- You can apply this optimization when the top function is **`main`** or when you specify an alternative top function (`--top-fname`) provided that the option `--do-not-expose-globals` is also given (currently supported only with GCC 4.9)

Fifth example

- Integration of existing IPs written in Verilog that receives structs passed by pointers

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/fifth
```

```
#include "module_lib.h"

void my_ip(uint8_t command, uint32_t param1, uint32_t param2) {
    static module1_output_t module1_output;
    static module2_output_t module2_output;
    switch(command) {
        case 0:
            module1(param1, param2 >> 16, &module1_output);
            break;
        case 1:
            module2(param1, &module2_output);
            break;
        case 2:
            printer1(module1_output.output1, module1_output.output2, module1_output.output3,
module1_output.output4);
            break;
        case 3:
            printer2(module2_output.output1, module2_output.output2, module2_output.output3);
            break;
        default:
            break;
    }
}
```

Integration of hand- written components in the HLS flow

- Function mapped on IPs has to be declared as extern:
  - `extern void module1(uint32_t input1, uint16_t input2, module1_output_t *outputs);`
- C code has to be passed with the following option
  - `--C-no-parse=module1.c,…`
- Binding between function **module1** and component **module1** has to be specified with a XML file and passed as an option to bambu
  - `$ bambu … module_lib.xml`
- Check these examples:
  - `examples/IP_integration`
  - `examples/breakout`
  - `examples/pong`
  - `examples/led_example`

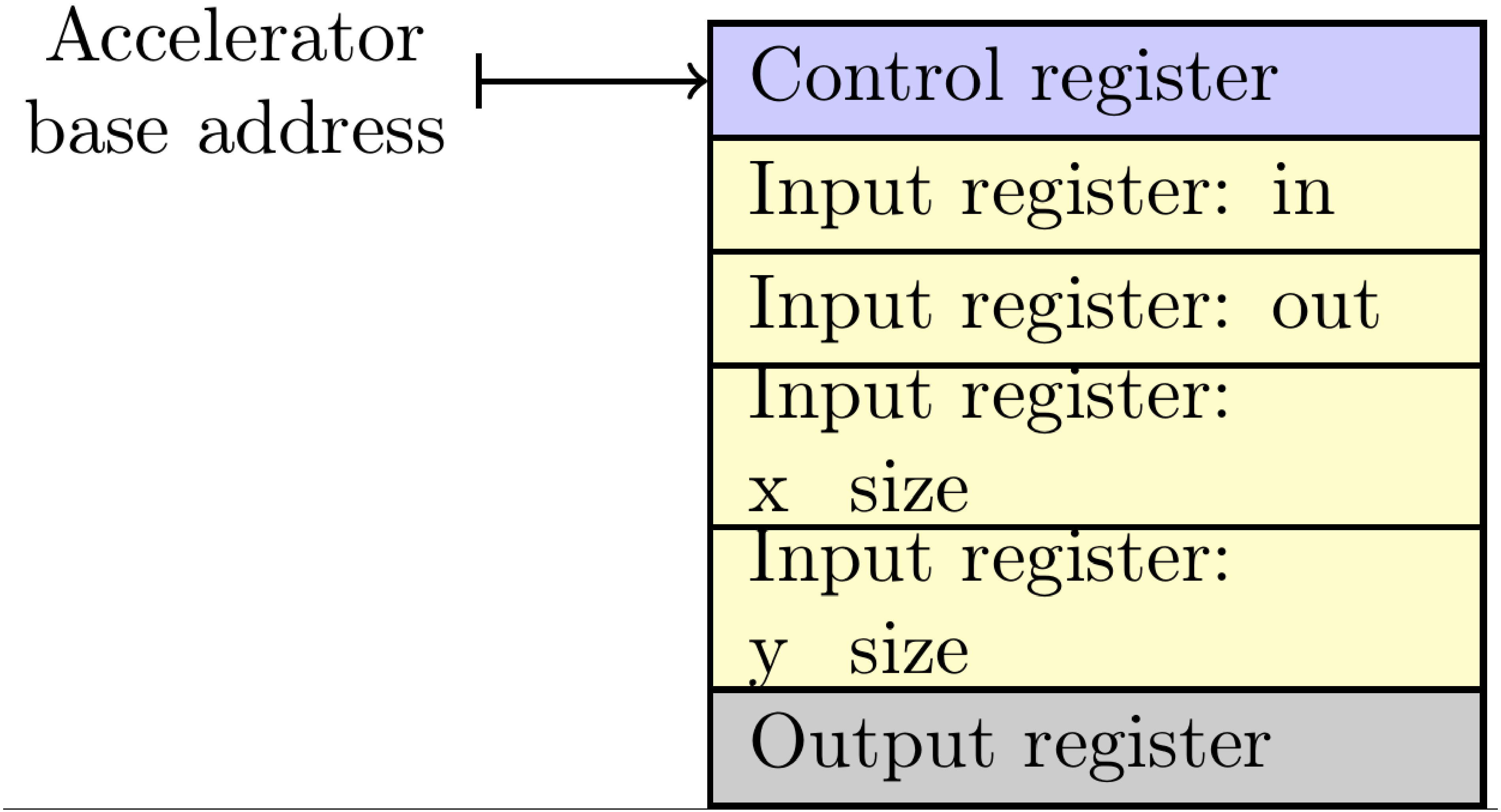
Synthesis of function pointers

```
int laplacian(char *, char *, int, int);
int make_inverse_image(char *, char *, int, int);
int sharpen(char *, char *, int, int);
int sobel(char *, char *, int, int);

int (*pipeline[MAX_DEPTH])(char *, char *, int, int);

void UserApp(char *in, char *out, int x_size, int y_size) {
  // ...
  // Pipeline configuration using function pointers
  add_filter(0, make_inverse_image);
  add_filter(1, sharpen);
  // ...
  // execute is synthesized in hardware
  execute(in, out, x_size, y_size);
}
void execute(char *in, char *out, int x_size, int y_size) {
  int i = 0;
  for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {
    if (pipeline[i] == 0) break;
    // here other hw accelerator are called
    // using function pointers
    int res = pipeline[i](in, out, x_size, y_size);
    if (res != 0) return;
    swap(in, out);
  }
  move_if_odd(i, in, out);
}
```

Adding a memory mapped interface to filters



Accelerator base address

Control register

Input register: in

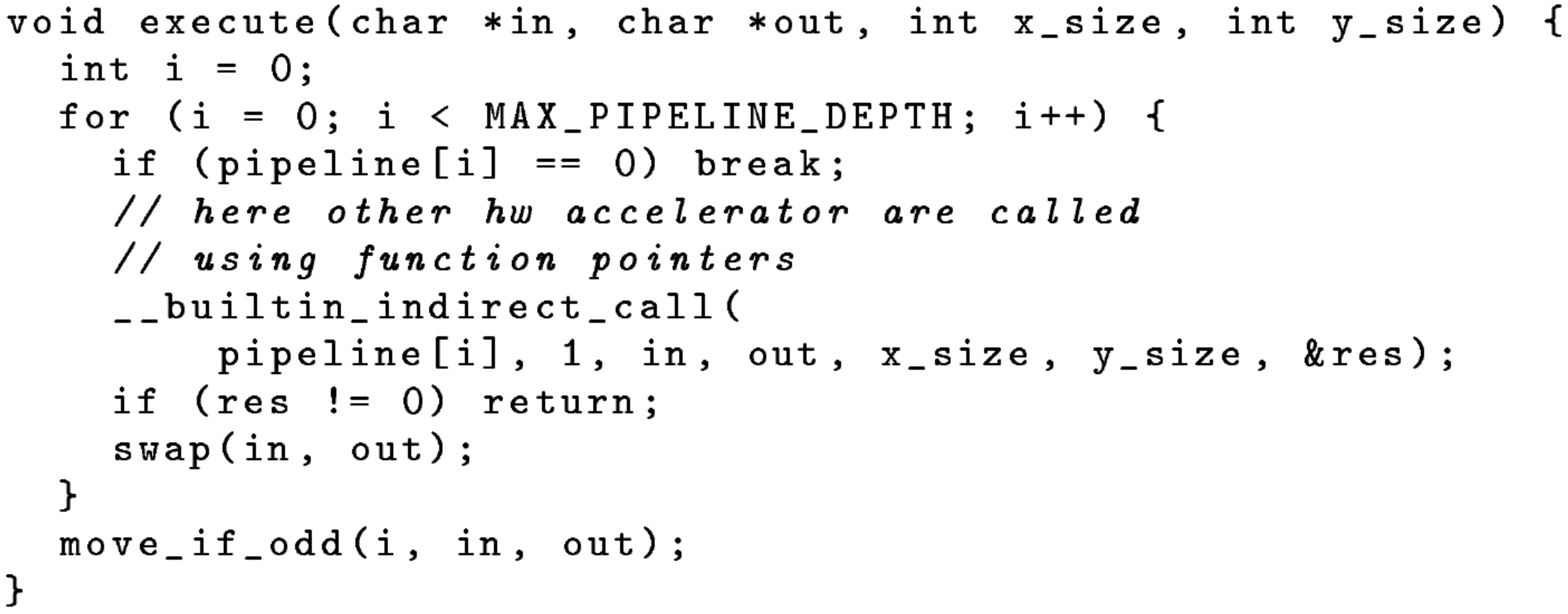
Input register: out

Input register: x size

Input register: y size

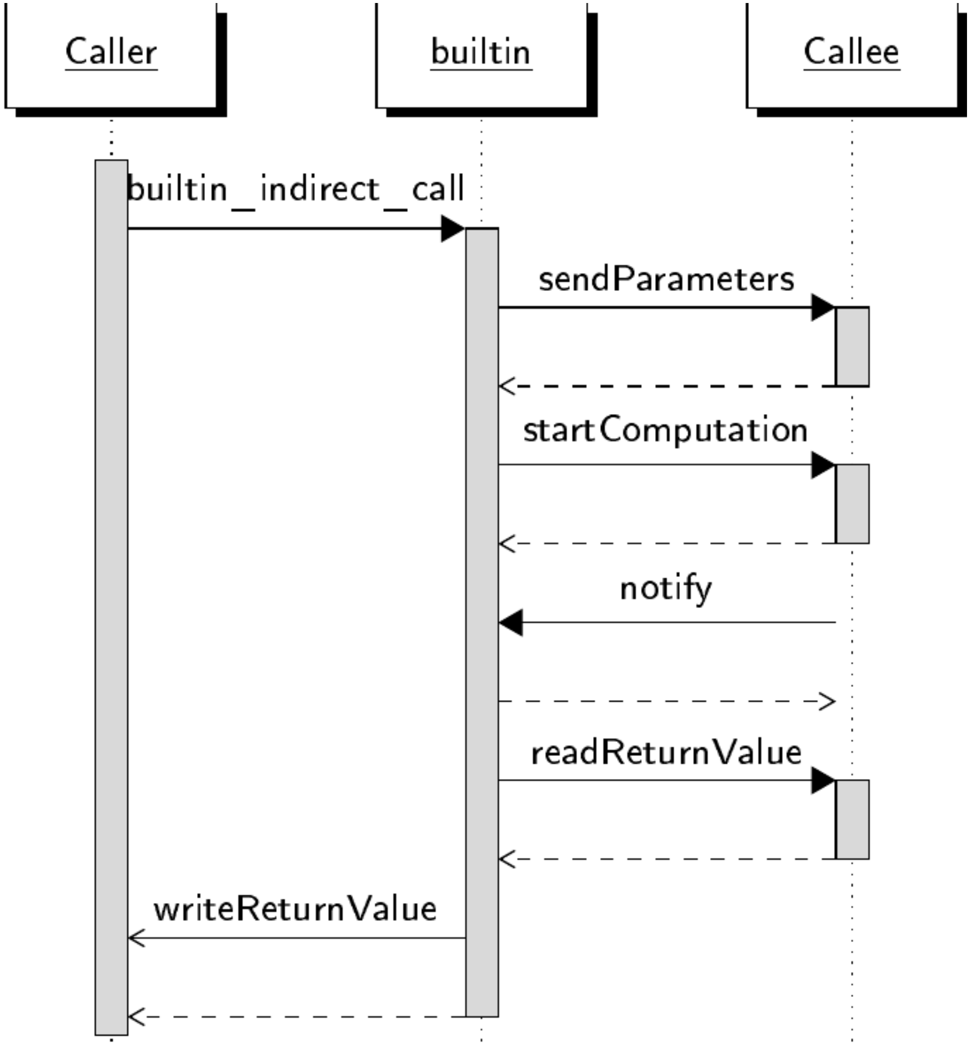
Output register

Code transformation performed



```
void execute(char *in, char *out, int x_size, int y_size) {
  int i = 0;
  for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {
    if (pipeline[i] == 0) break;
    // here other hw accelerator are called
    // using function pointers
    __builtin_indirect_call(
        pipeline[i], 1, in, out, x_size, y_size, &res);
    if (res != 0) return;
    swap(in, out);
  }
  move_if_odd(i, in, out);
}
```

Sequence diagram for function indirect call



Call mechanism complexity: #cycles=Wl(Np+1)+lhs(Wl+Rl)

Sixth example

- ❑ Parametric quick sort

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/sixth
```

- ❑ Quick sort parametric with respect the comparison function
- ❑ Run the script and check the results

C++ support

- Support of c++ is ongoing:
  - templates
  - C++11 and beyond
  - ac_types from Mentor Graphics could be used
    - ac_int and ac_fixed currently cannot be used in the function interfaces
    - not c++11 friendly

```
#include <algorithm>
int gcd(int x, int y )
 {
       if( x < y )
          std::swap( x, y );

       while( y > 0 )
       {
          int f = x % y;
          x = y;
          y = f;
       }
       return x;
 }
```

Fortran support (1)

```
*     euclid.f (FORTRAN 77)
*     Find greatest common divisor using the Euclidean algorithm

      FUNCTION NGCD(NA, NB)
        IA = NA
        IB = NB
    1   IF (IB.NE.0) THEN
          ITEMP = IA
          IA = IB
          IB = MOD(ITEMP, IB)
          GOTO 1
        END IF
        NGCD = IA
        RETURN
      END
```

By default parameters are passed by reference

Fortran support (2)

```
*     euclid.f (FORTRAN 77)
*     Find greatest common divisor using the Euclidean algorithm

      FUNCTION NGCD(NA, NB)
        integer, value :: NA ! input
        integer, value :: NB ! input
        IA = NA
        IB = NB
    1   IF (IB.NE.0) THEN
          ITEMP = IA
          IA = IB
          IB = MOD(ITEMP, IB)
          GOTO 1
        END IF
        NGCD = IA
        RETURN
      END
```

Here parameters are passed by copy

- ❑ Fortran support in progress…

OpenMP support

- #pragma omp simd is supported

```
#pragma omp declare simd
void add (int accelnum, int startidx, int endidx)
{
   int sum = 0;
   int i;
   for (i = 0; i < OPS_PER_ACCEL; i++) {
      sum += array[i+startidx];
   }
   output[accelnum] = sum;
}

main() {
...
#pragma omp simd
   for (i = 0; i < NUM_ACCELS; i++)
   {
      add(i, i * OPS_PER_ACCEL, (i + 1)*OPS_PER_ACCEL);
   }
```

- OmpMP support in progress…