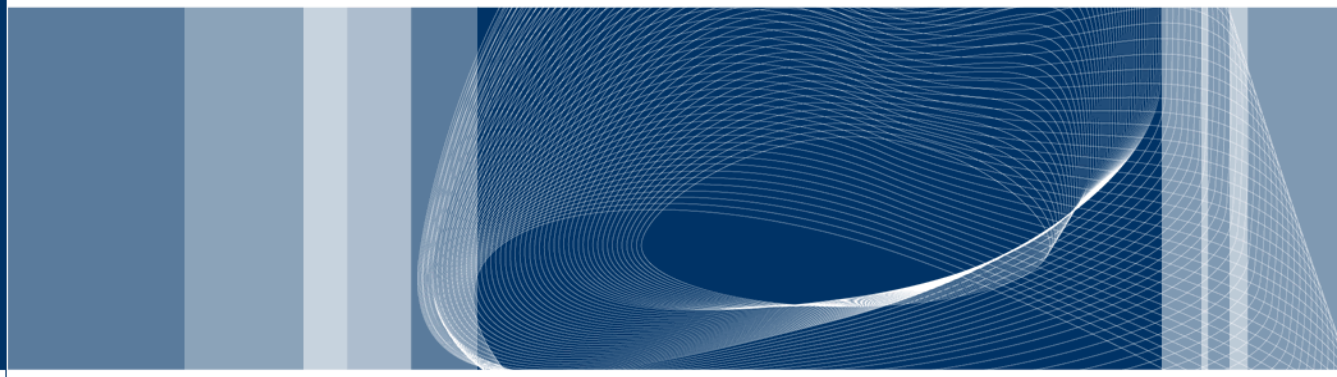
POLITECNICO DI MILANO

The 28th International Conference on Parallel Architectures and Compilation Techniques

Seattle, WA, USA, September 21st, 2019



PACT19 Tutorial

Bambu: Productive FPGA Programming for Complex Parallel Applications

Fabrizio Ferrandi (Politecnico di Milano), Vito Giovanni Castellana (PNNL), Marco Minutoli (PNNL), Marco Lattuada (Politecnico di Milano), Antonino Tumeo (PNNL)

Outline

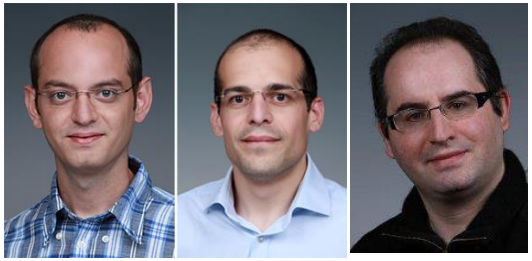
- Presentation of bambu
- Exploiting Vectorization in High Level Synthesis of Nested Irregular Loops
- Compiler Based Optimizations, Tuning and Customization of Generated Accelerators

- Target Customization and Tool Integration
- Enabling the High Level Synthesis of Data Analytics Accelerators

Material prepared by

- Politecnico di Milano
  - Fabrizio Ferrandi
  - Marco Lattuada
  - Christian Pilato
  - Pietro Fezzardi

- PNNL
  - Vito Giovanni Castellana
  - Marco Minutoli
  - Antonino Tumeo



Tutorial Material

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

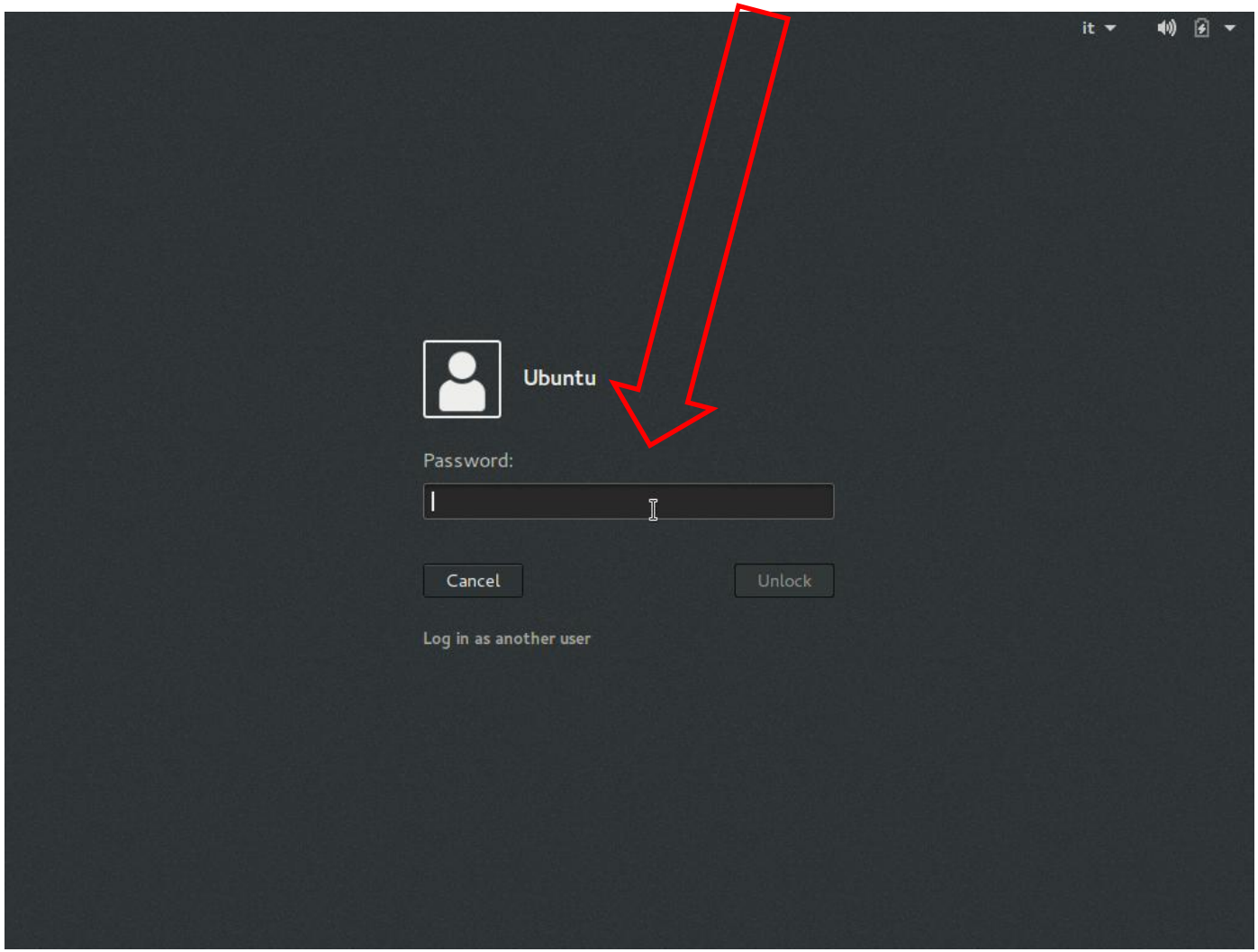
https://panda.dei.polimi.it/?page_id=838

VirtualMachine download&import

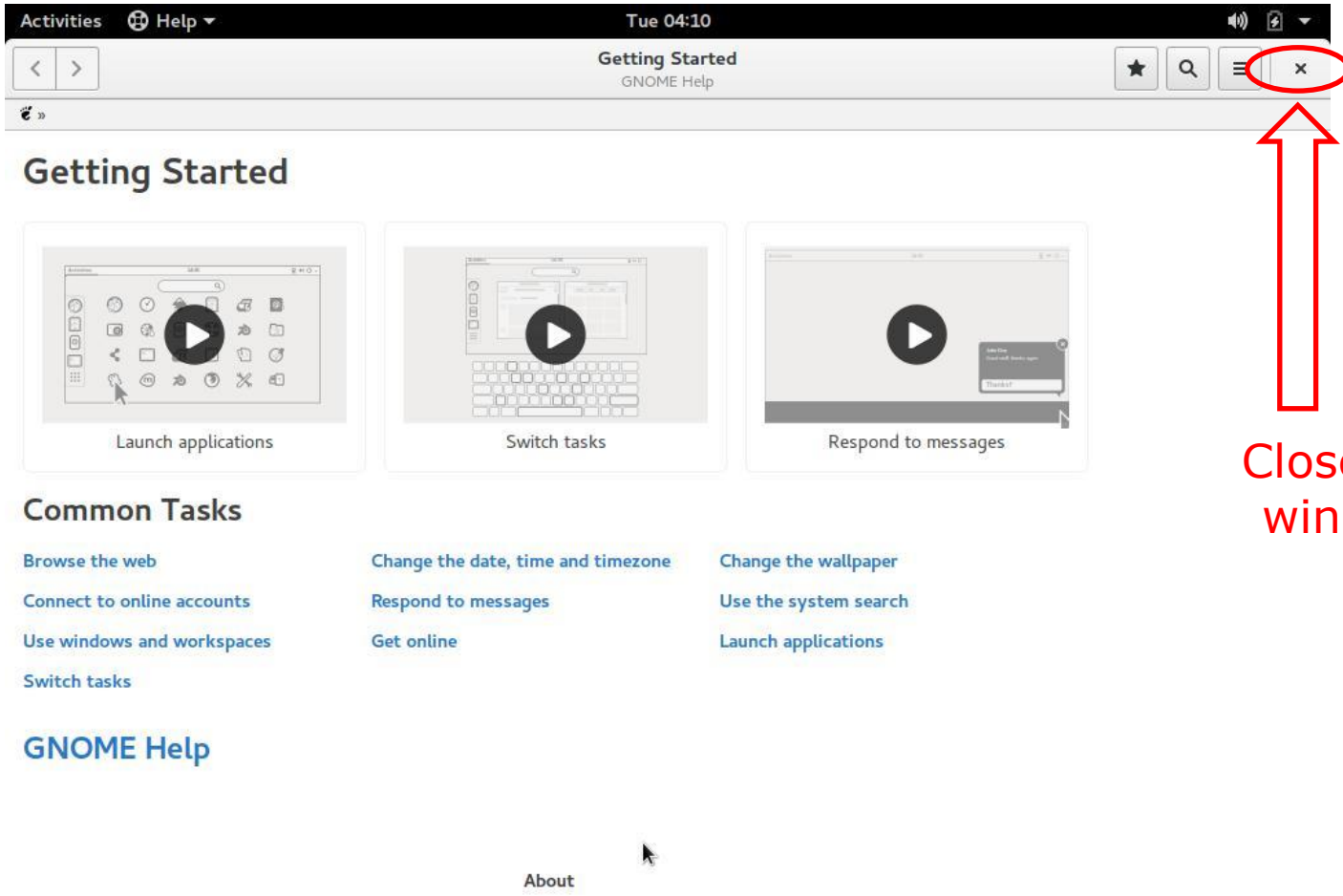
- There are two .ova files. One for 32bit operating systems and another one for 64bit operating systems.
- Once downloaded you can import the virtual machine by clicking on the VirtualBox Manager menu:
  - File
    - Import appliance…

Login

Type as a password: password



Starting screen

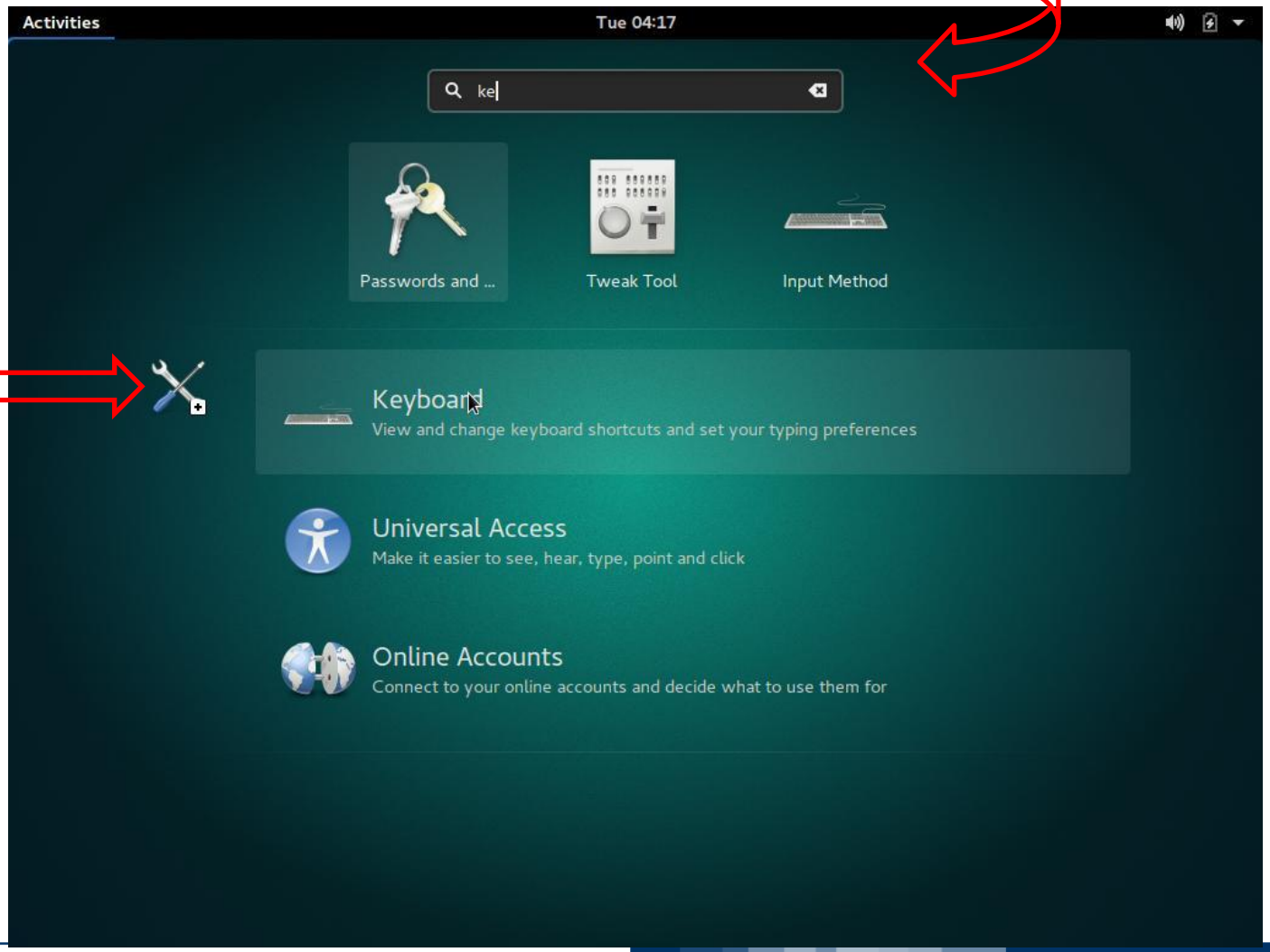


Close the window

Configure the keyboard

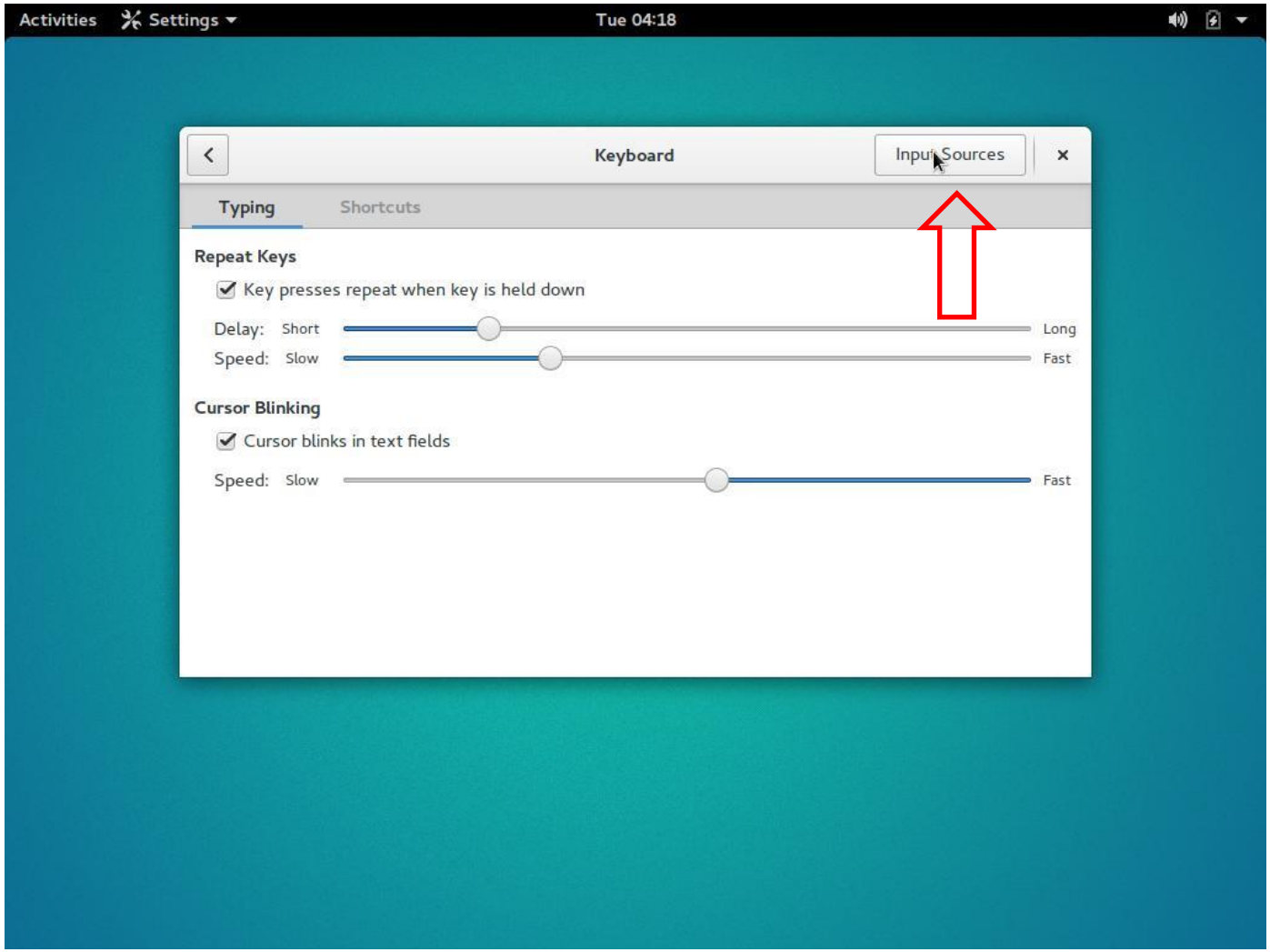
1) Click on activities

2) Search for "keyboard"

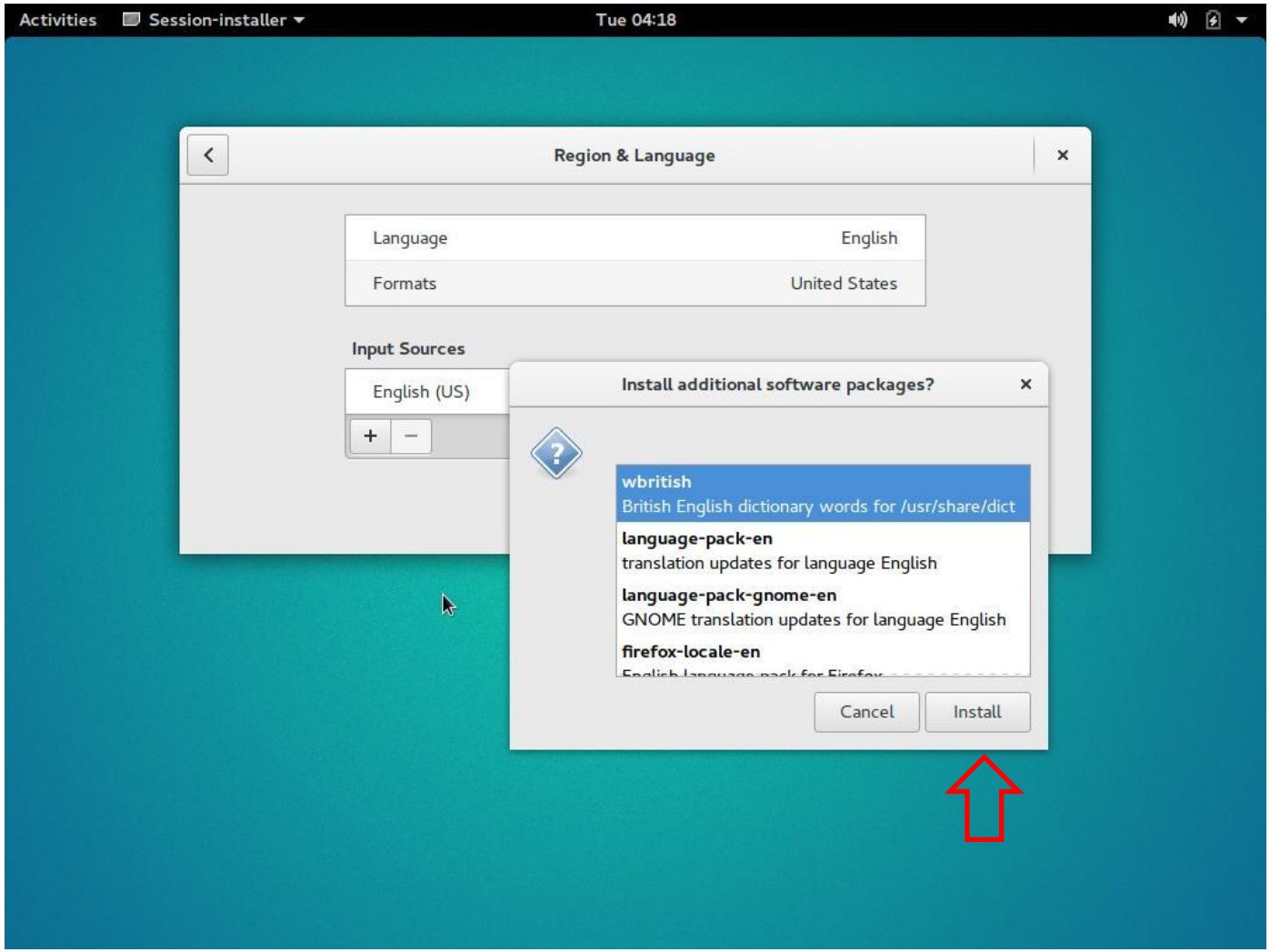


3) Select keyboard

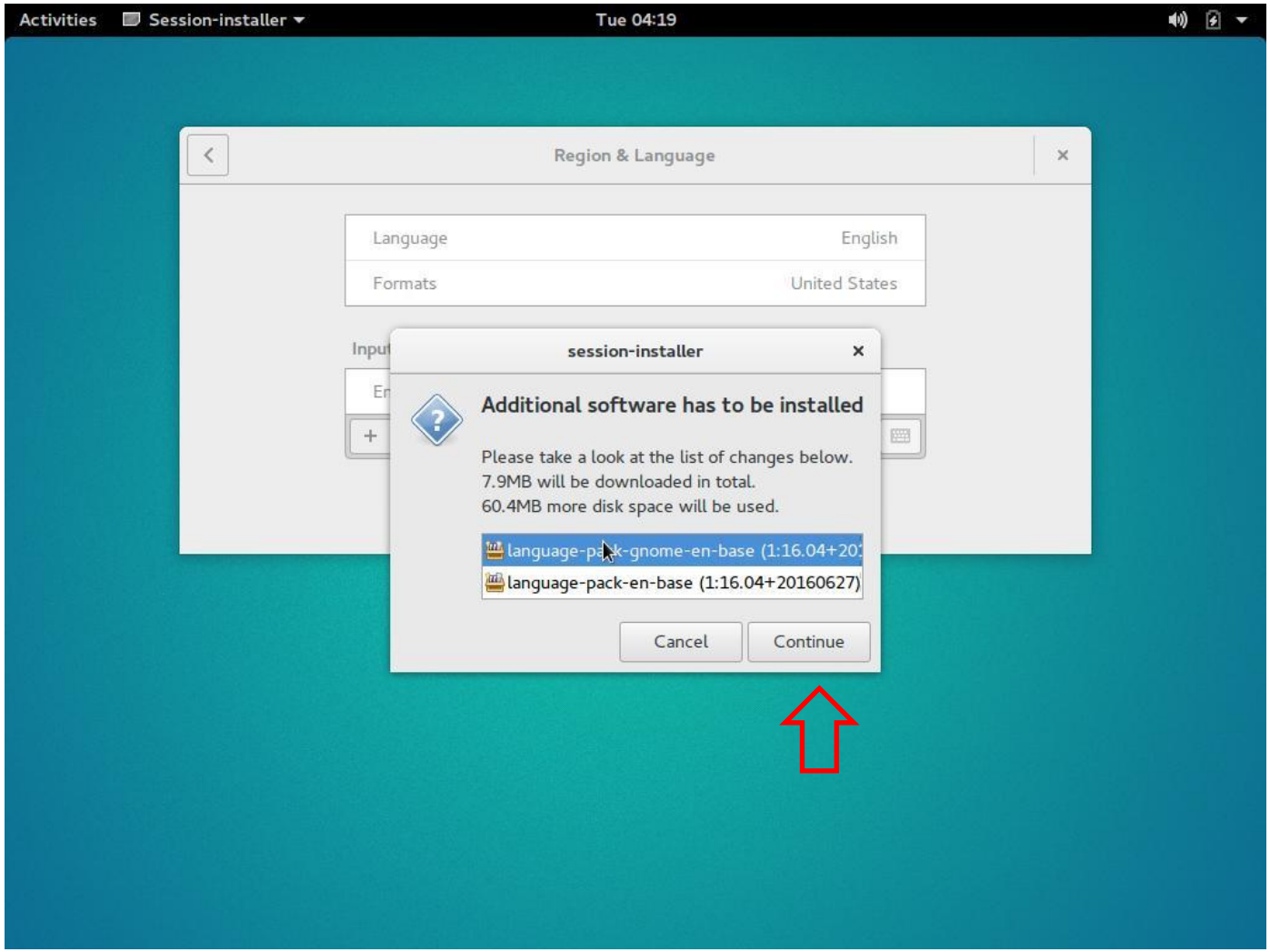
Select the input sources



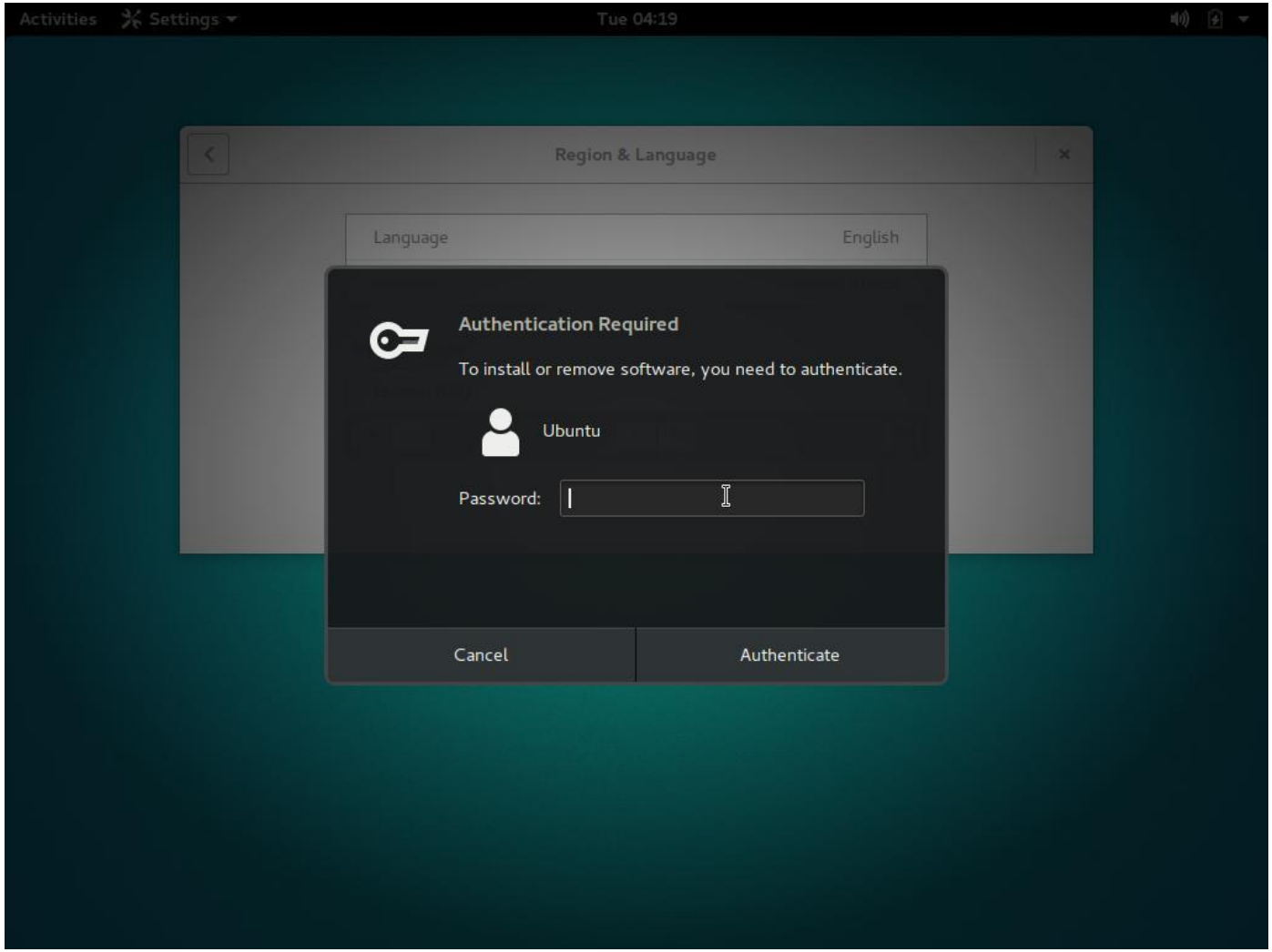
Install additional packages



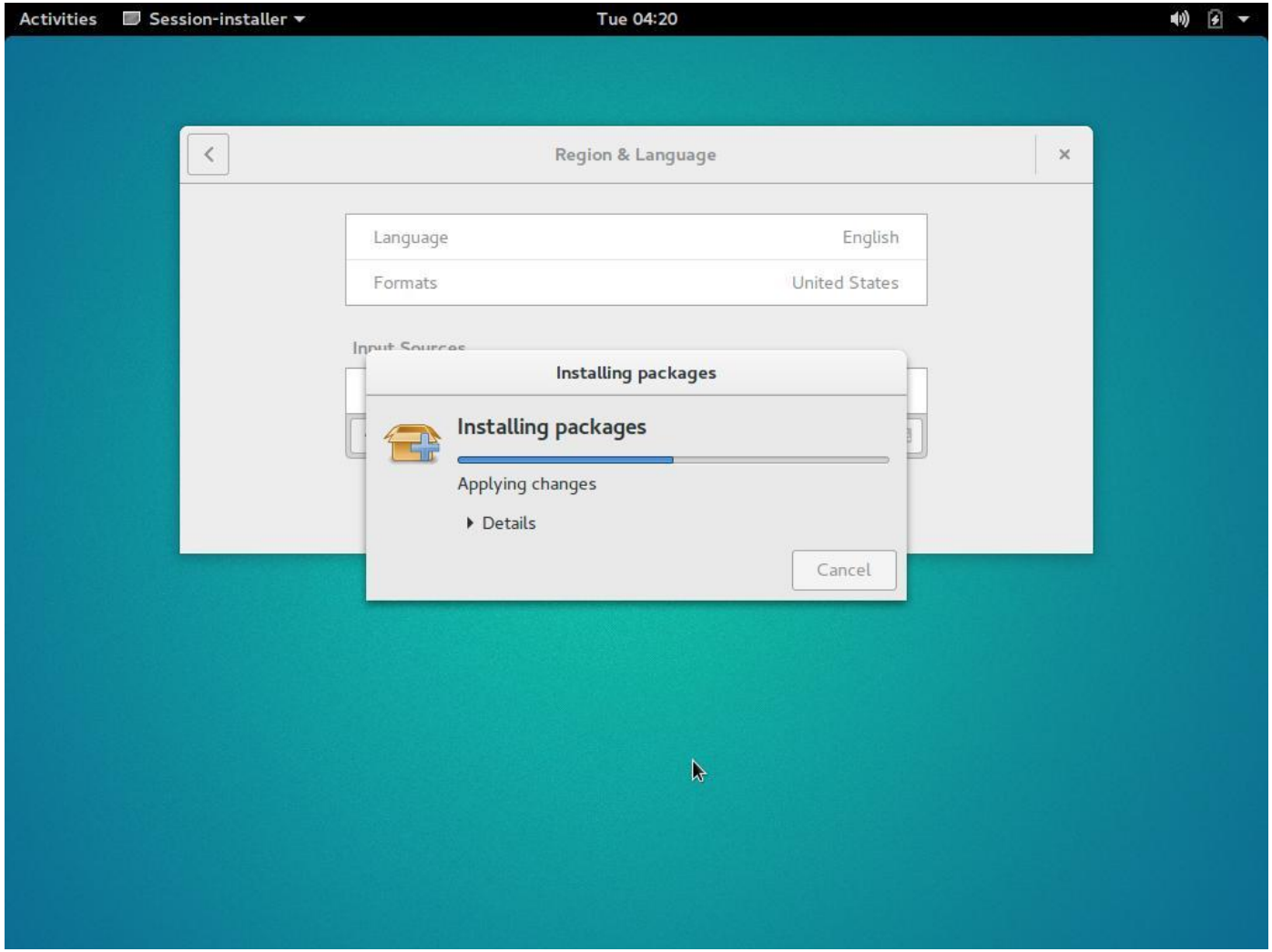
Confirm the installation



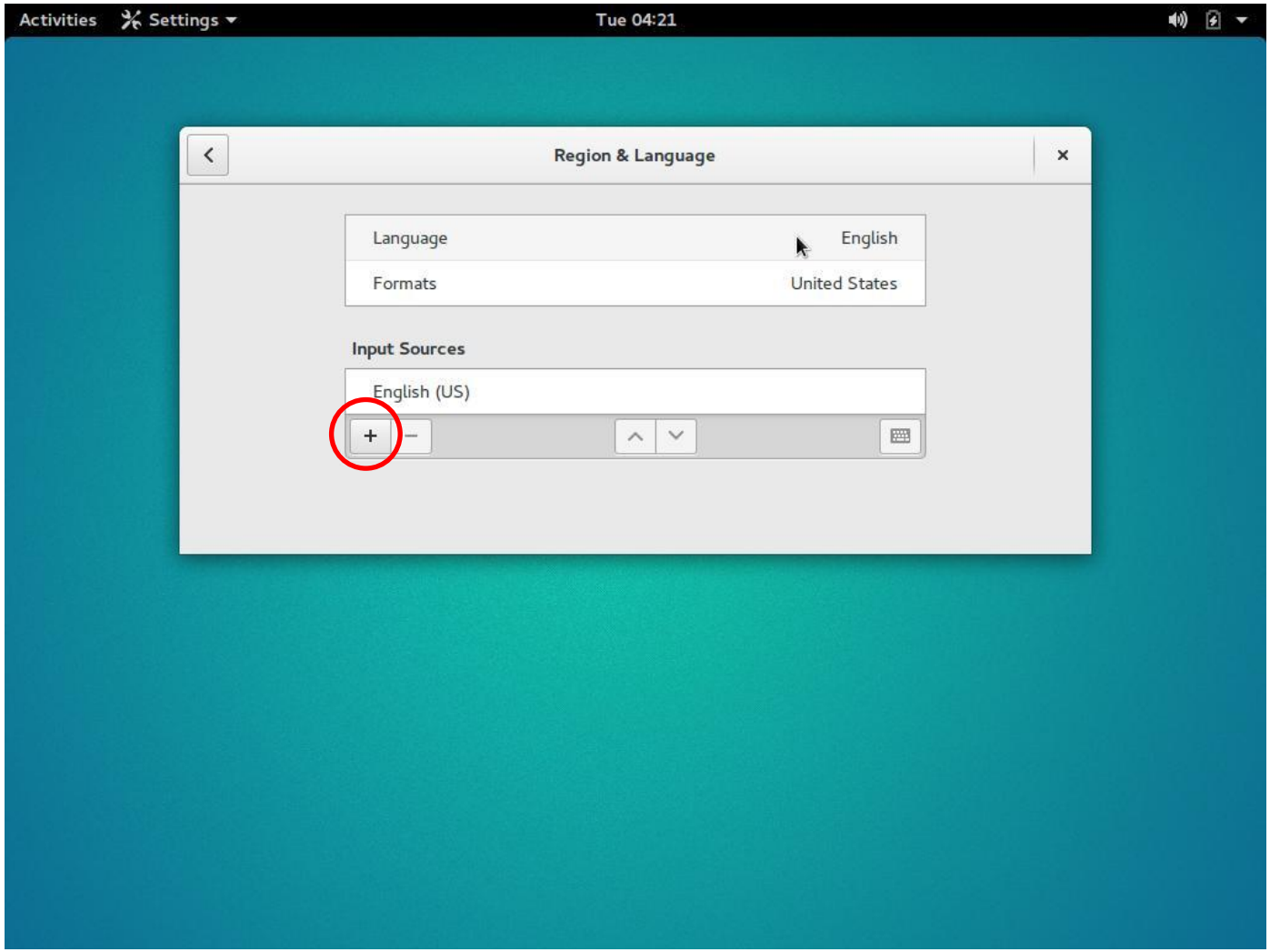
Type as a password: password



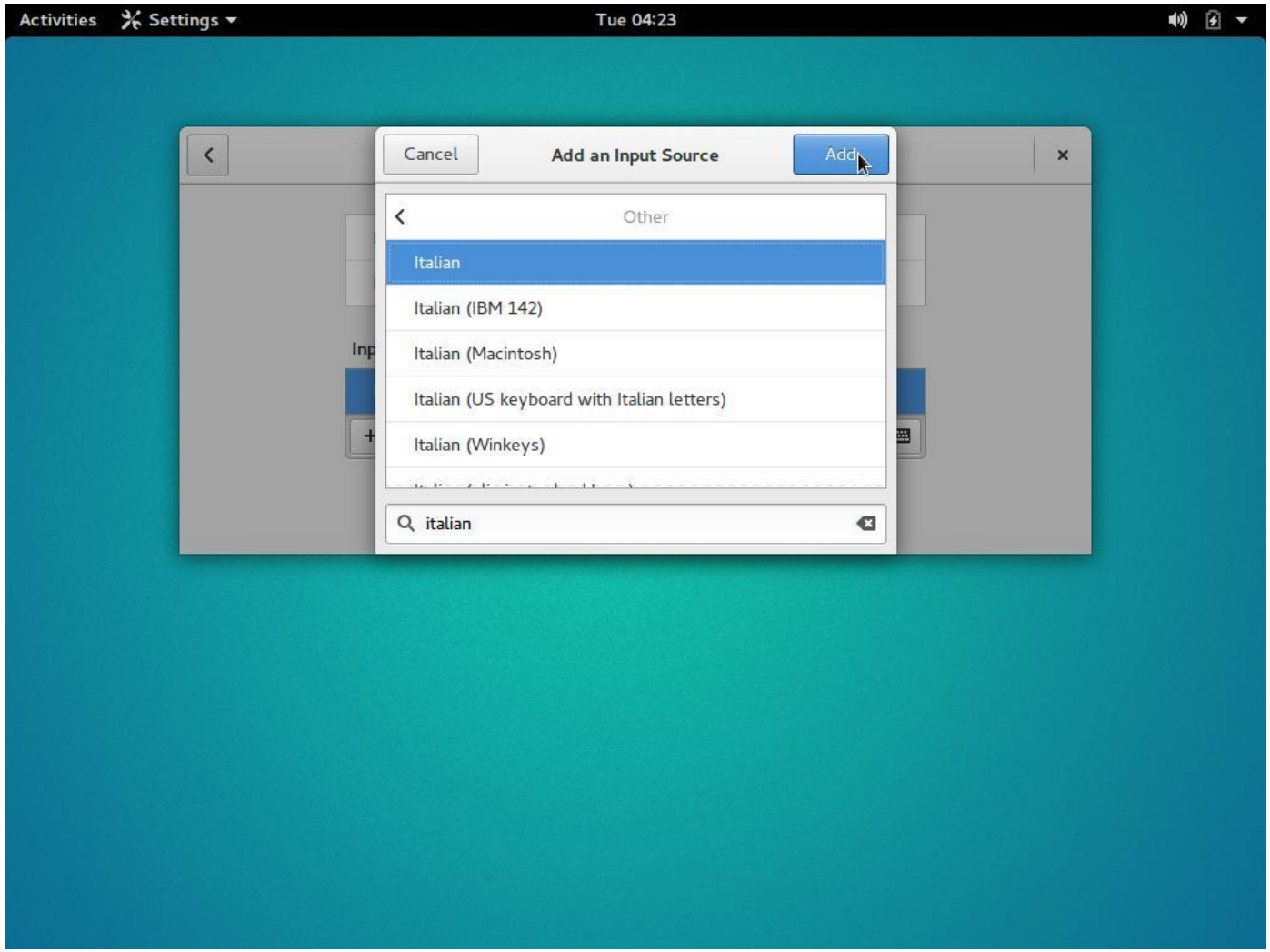
Wait a while


Activities
Session-installer
Tue 04:20
Region & Language
Language
English
Formats
United States
Input Sources
Installing packages
Installing packages
Applying changes
Details
Cancel

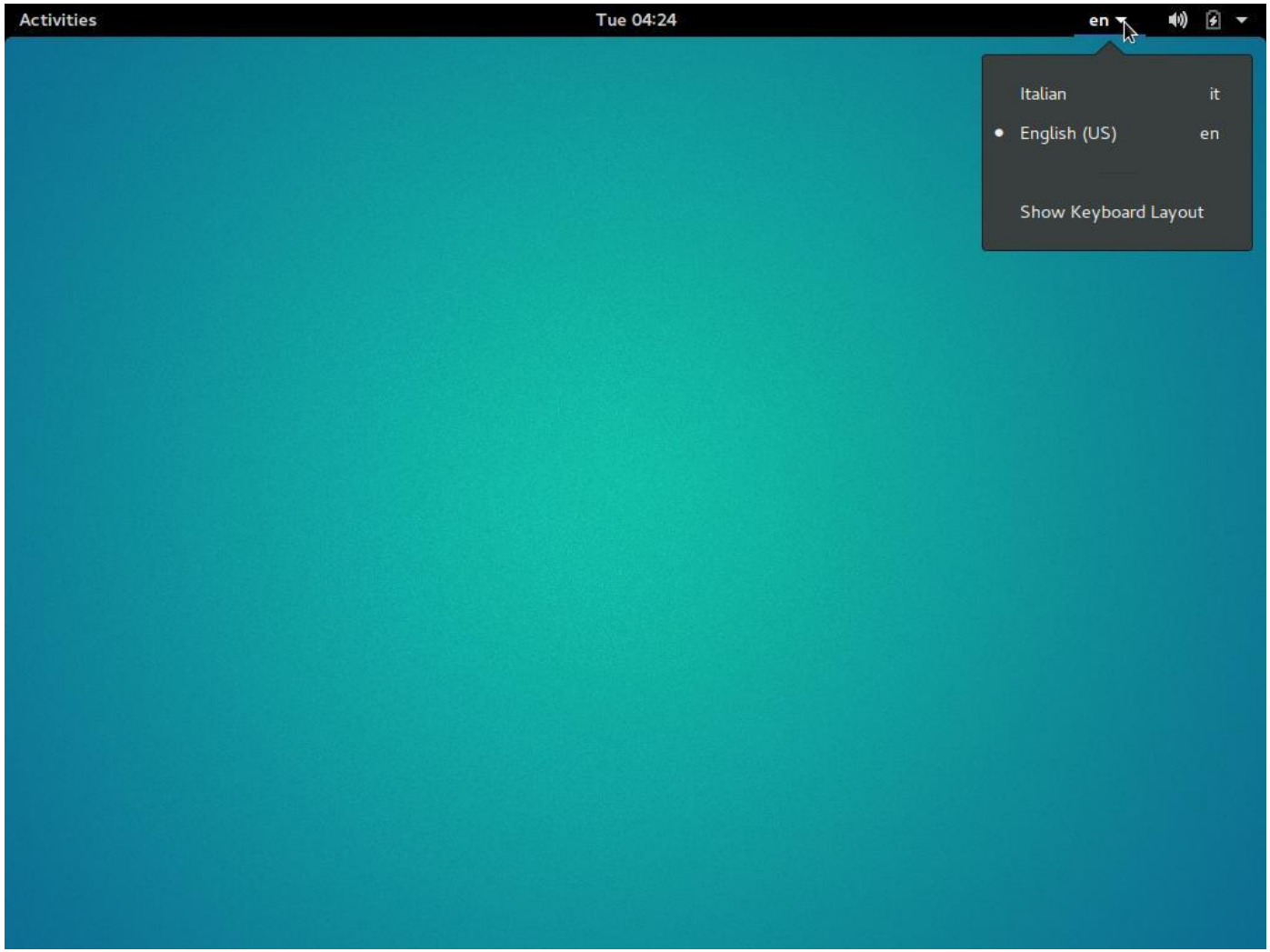
Add a new input source



Type the keyboard language

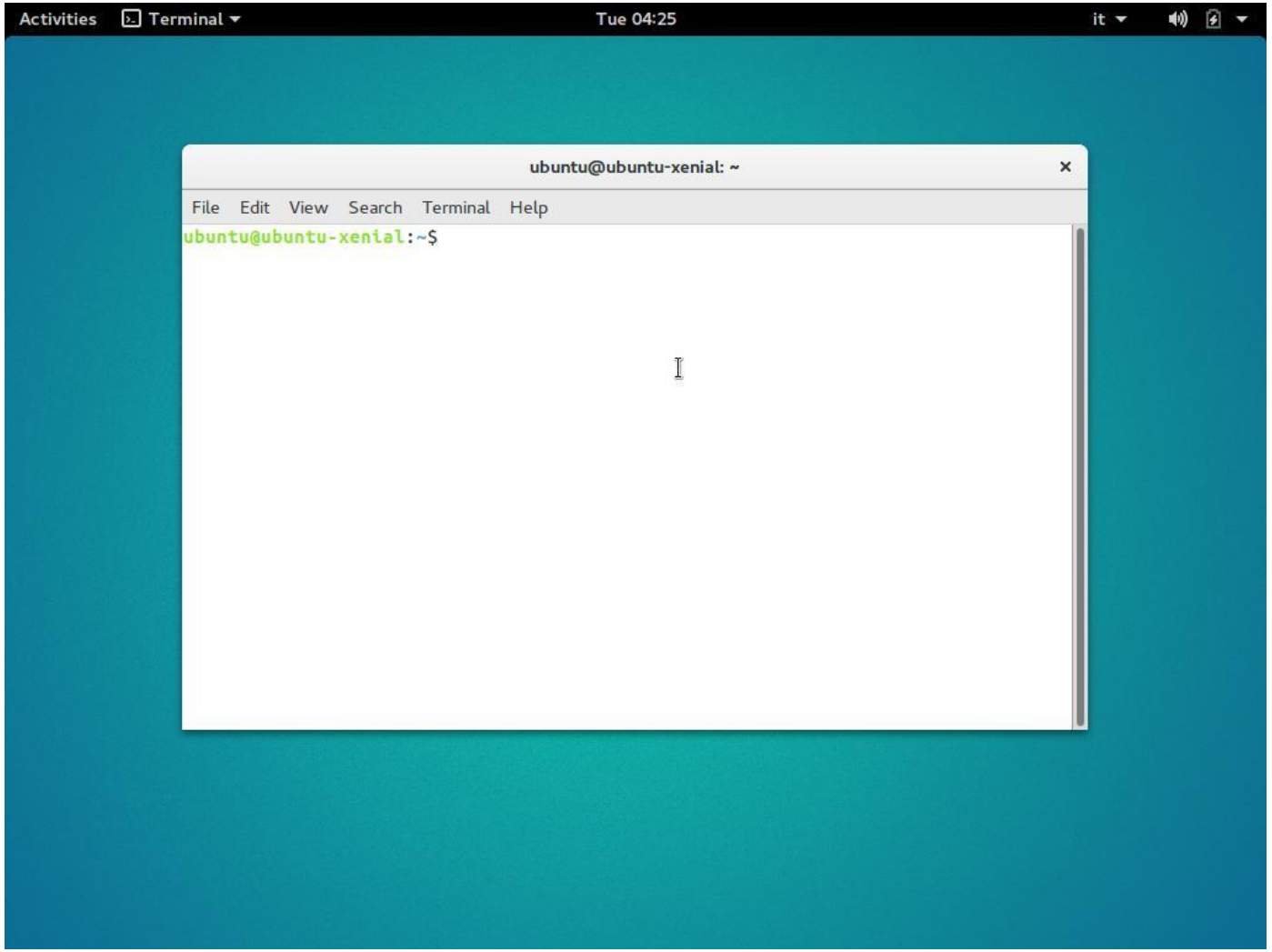


Activate the keyboard layout



Open a terminal

Press simultaneously: ctrl alt t



Update the git repository


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

```
git pull
```

First synthesis

- Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_pact_2019/Introduction/first
```

- Edit the file C by typing:

```
gedit module.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

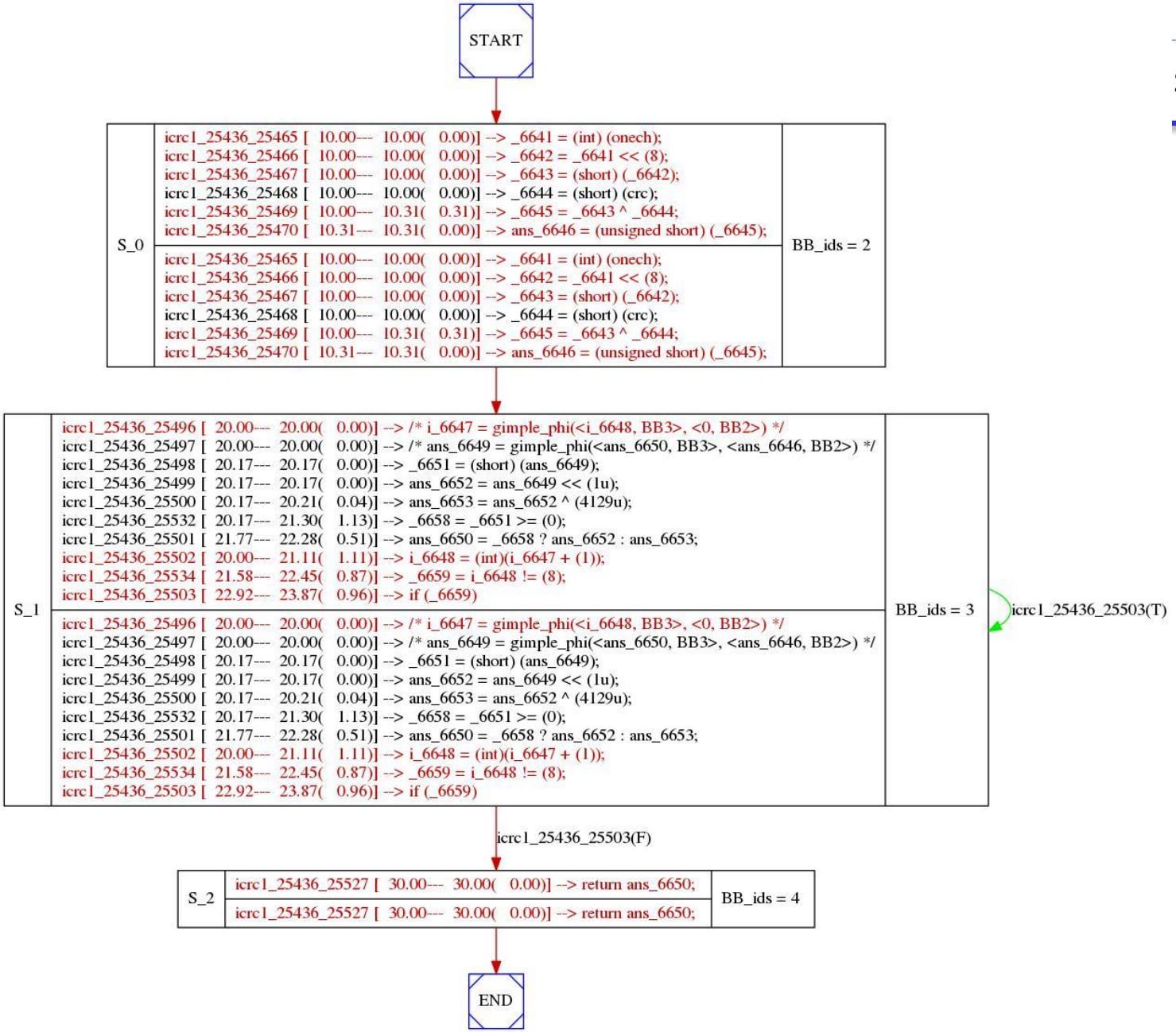
- Change directory to icrc1:

```
cd icrc1
```

- Display the FSM:

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

FSM


START
S_0
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);
BB_ids = 2
S_1
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)
BB_ids = 3
icrc1_25436_25503(T)
icrc1_25436_25503(F)
S_2
icrc1_25436_25527 [30.00--- 30.00(0.00)] --> return ans_6650;
BB_ids = 4
END

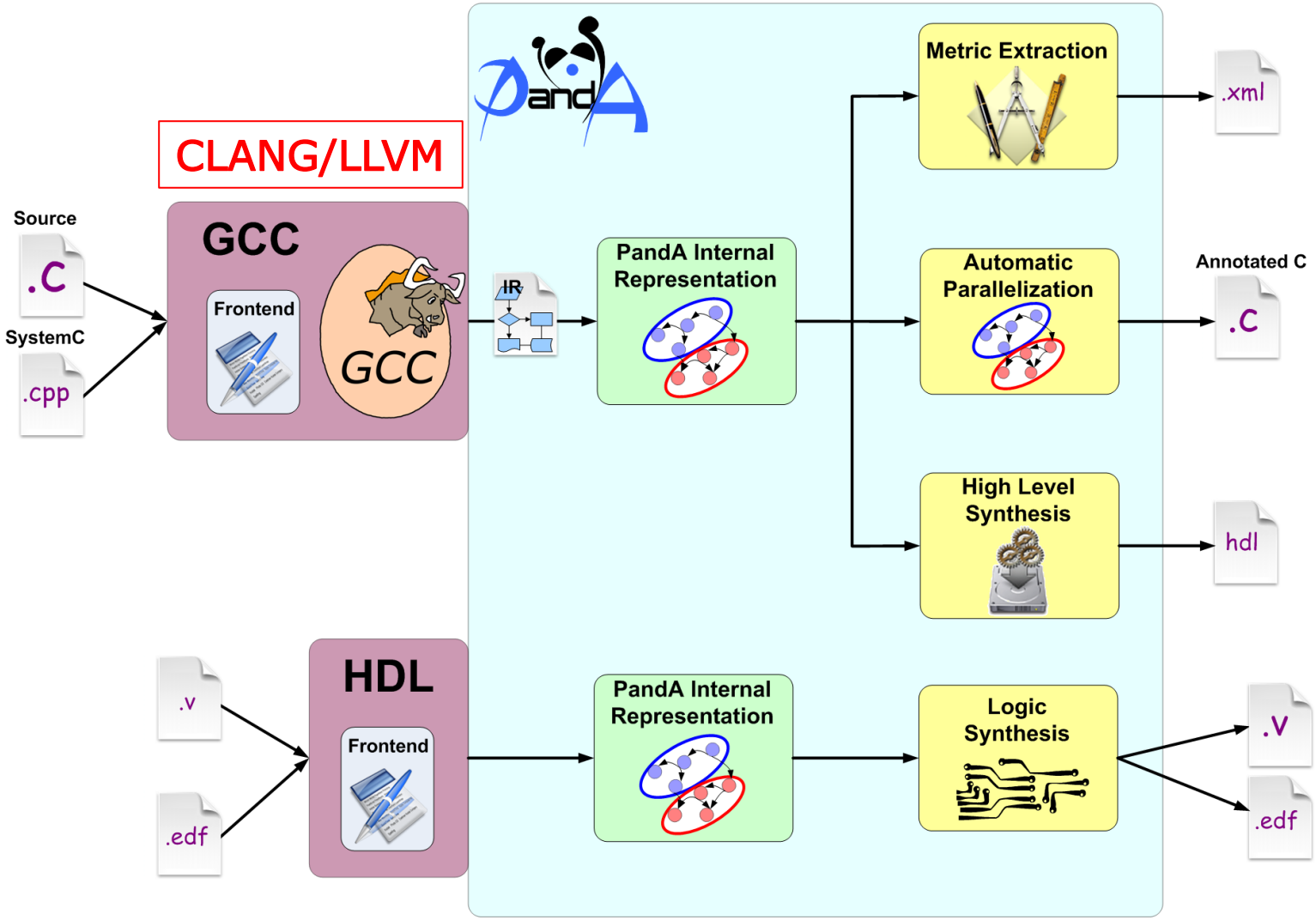
Open the RTL Verilog


```
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
```

A bit of history about PandA

- 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.

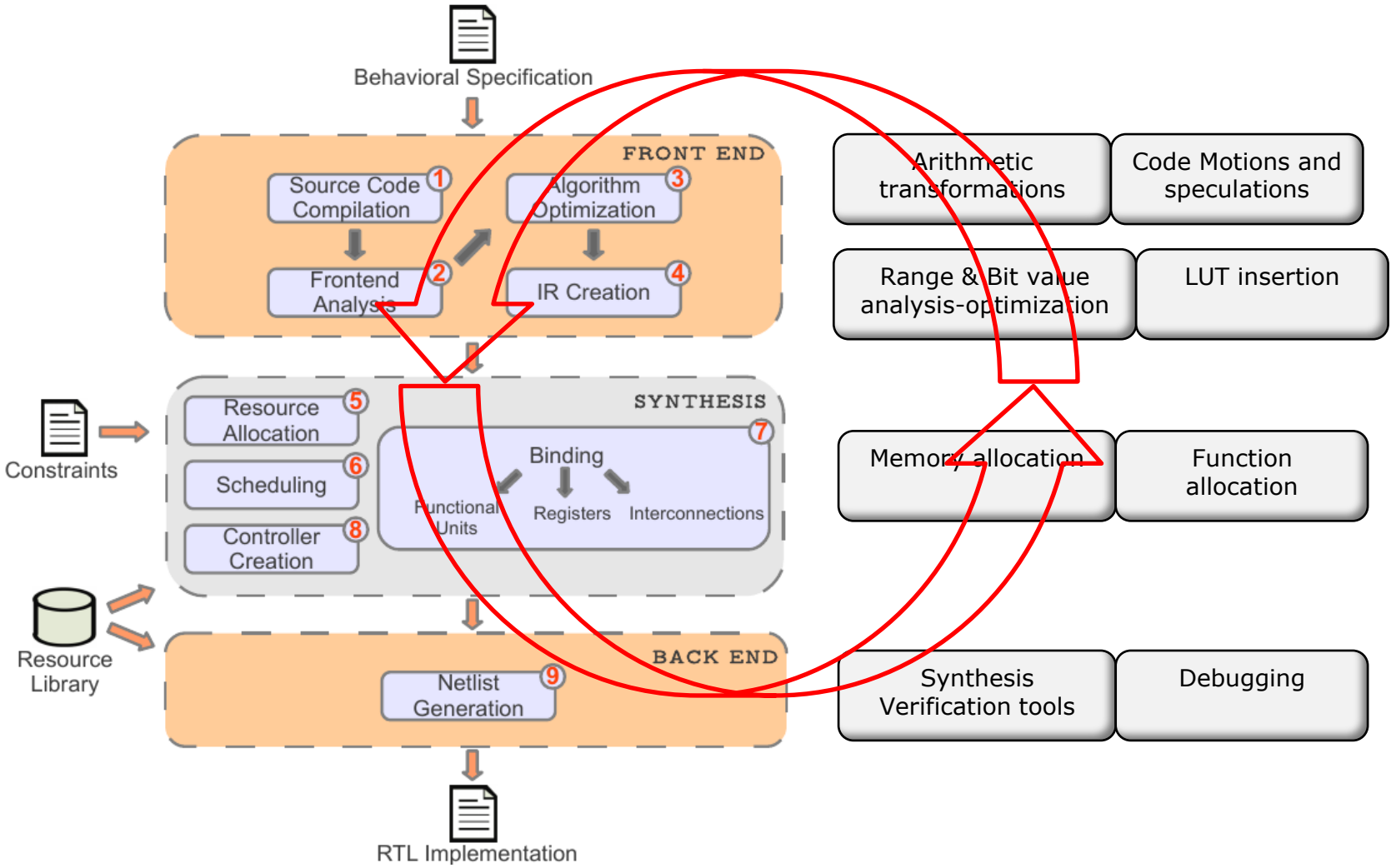
Framework Overview



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/CLANG-LLVM 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

Bambu Overview



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 what standard compilers accept as input (CLANG/LLVM and GCC)
- 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_pact_2019/Introduction/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_pact_2019/Introduction/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

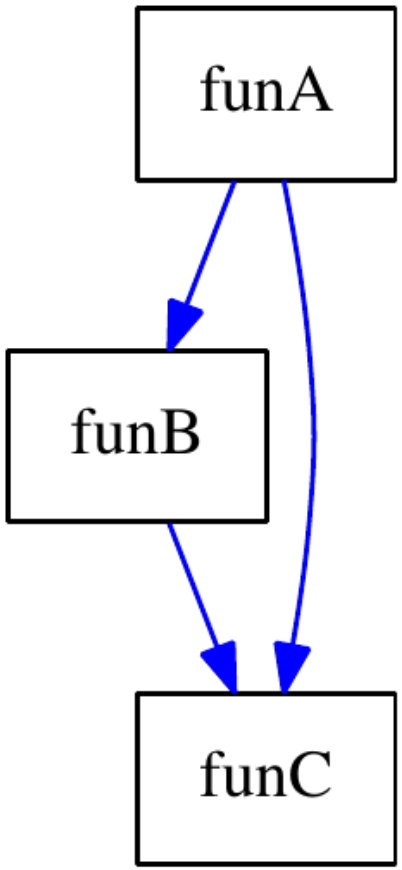
- `bambu` uses the text-based IR and the builtin linker to build a single in-memory representation and perform HLS
  - `$ bambu file1.c file2.c --top-fname=fname`
- In case the option `--top-fname` is not used, `bambu` automatically identifies which function can act as the top one.
- With clang link time optimization is used.

Synthesis per function

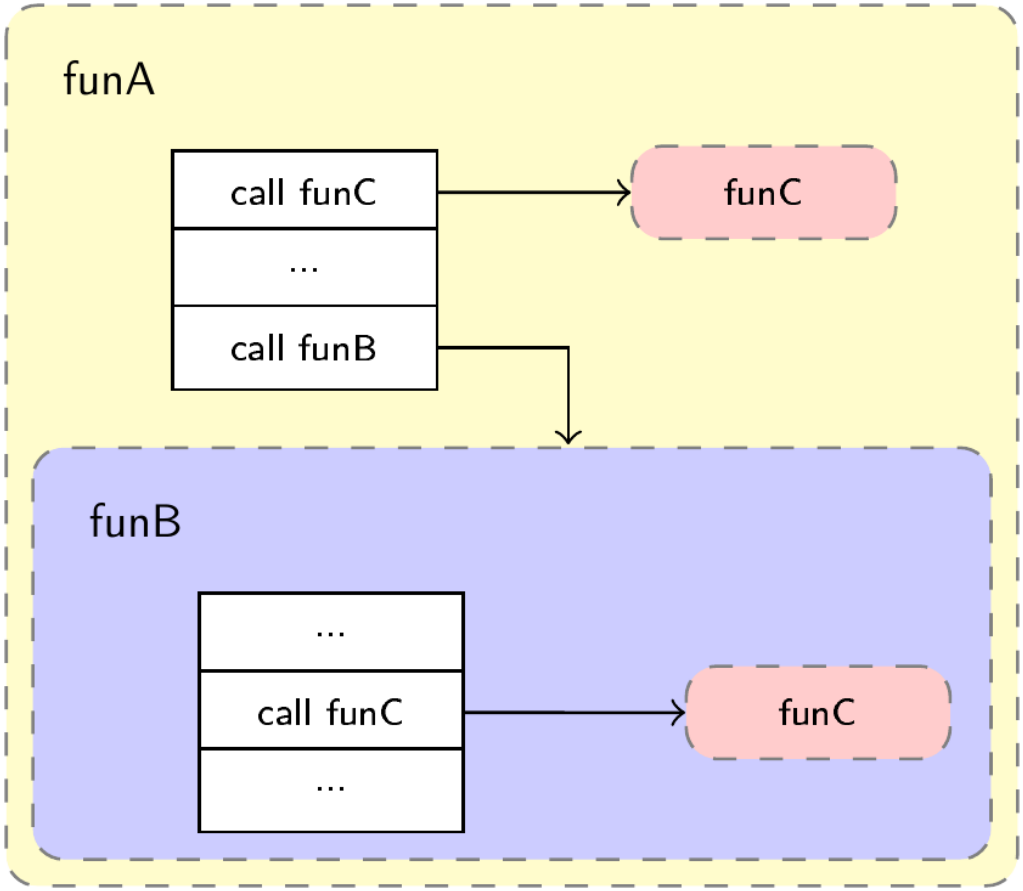
- one component per function
  - function interface
  - start and done
  - parameter passing
    - wires
    - memory interaction
    - none (ap_none), acknowledge (ap_ack), valid (ap_vld), ovalid (ap_ovld), handshake (ap_hs), fifo (ap_fifo) and array (ap_memory)
- hierarchy based on call graph
  - no-recursion
  - proxy

Synthesis per function: hierarchy

- One component per function

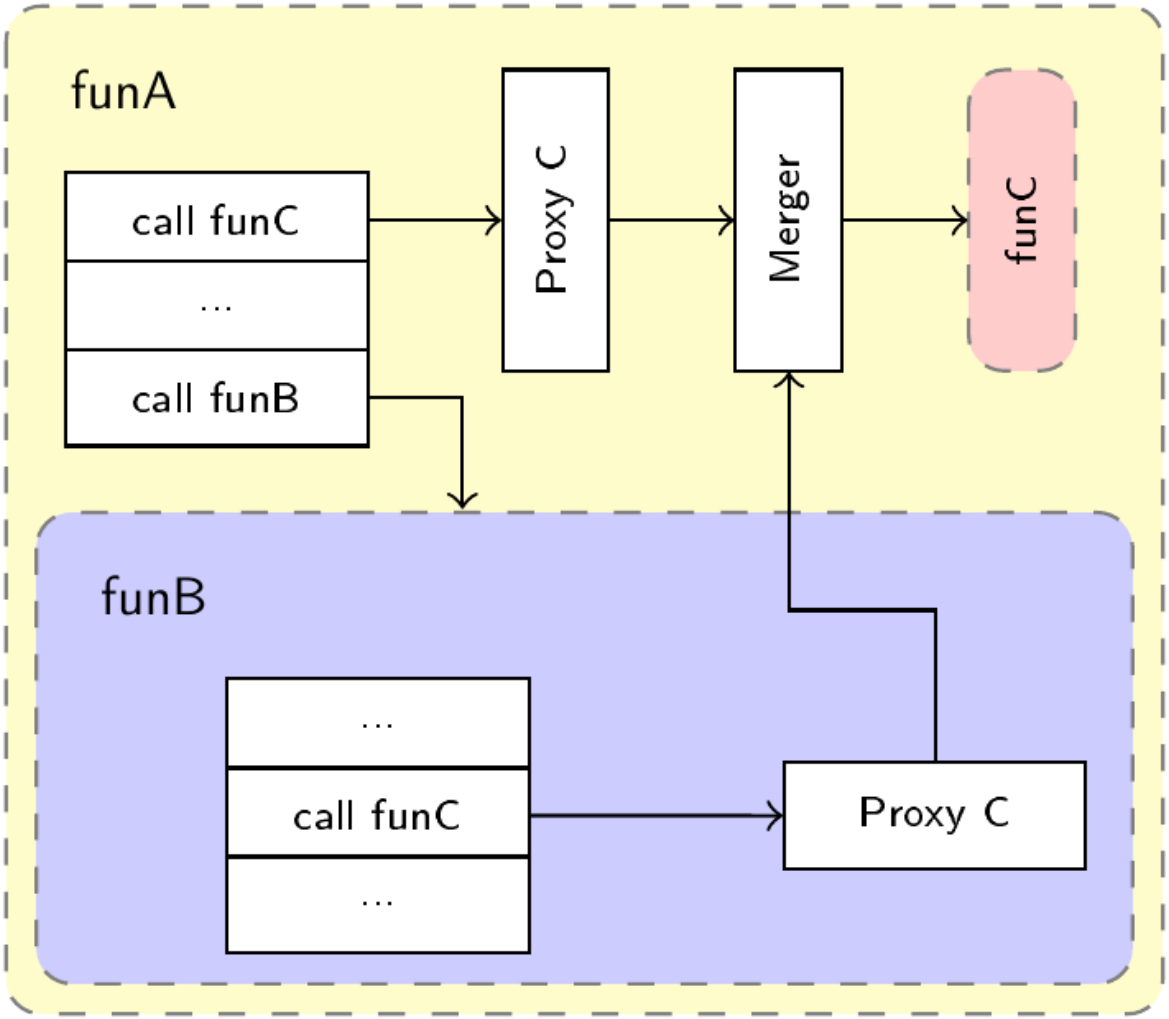


call graph



RTL hierarchy

Synthesis per function: proxy



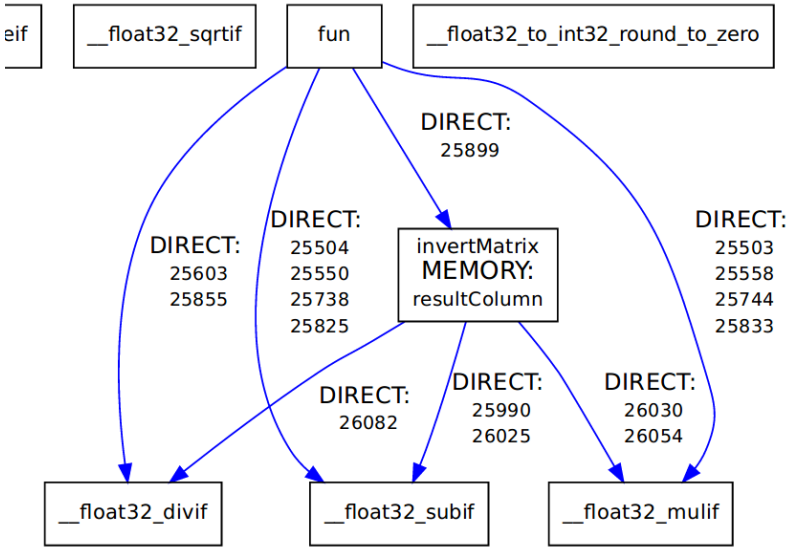
Fourth example

- Lu-decomposition with single precision floating point arithmetic

```
cd ~/PandA-bambu/documentation/tutorial_pact_2019/Introduction/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`.

Fifth example

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

```
cd ~/PandA-bambu/documentation/tutorial_pact_2019/Introduction/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`

Sixth example

- Parametric quick sort

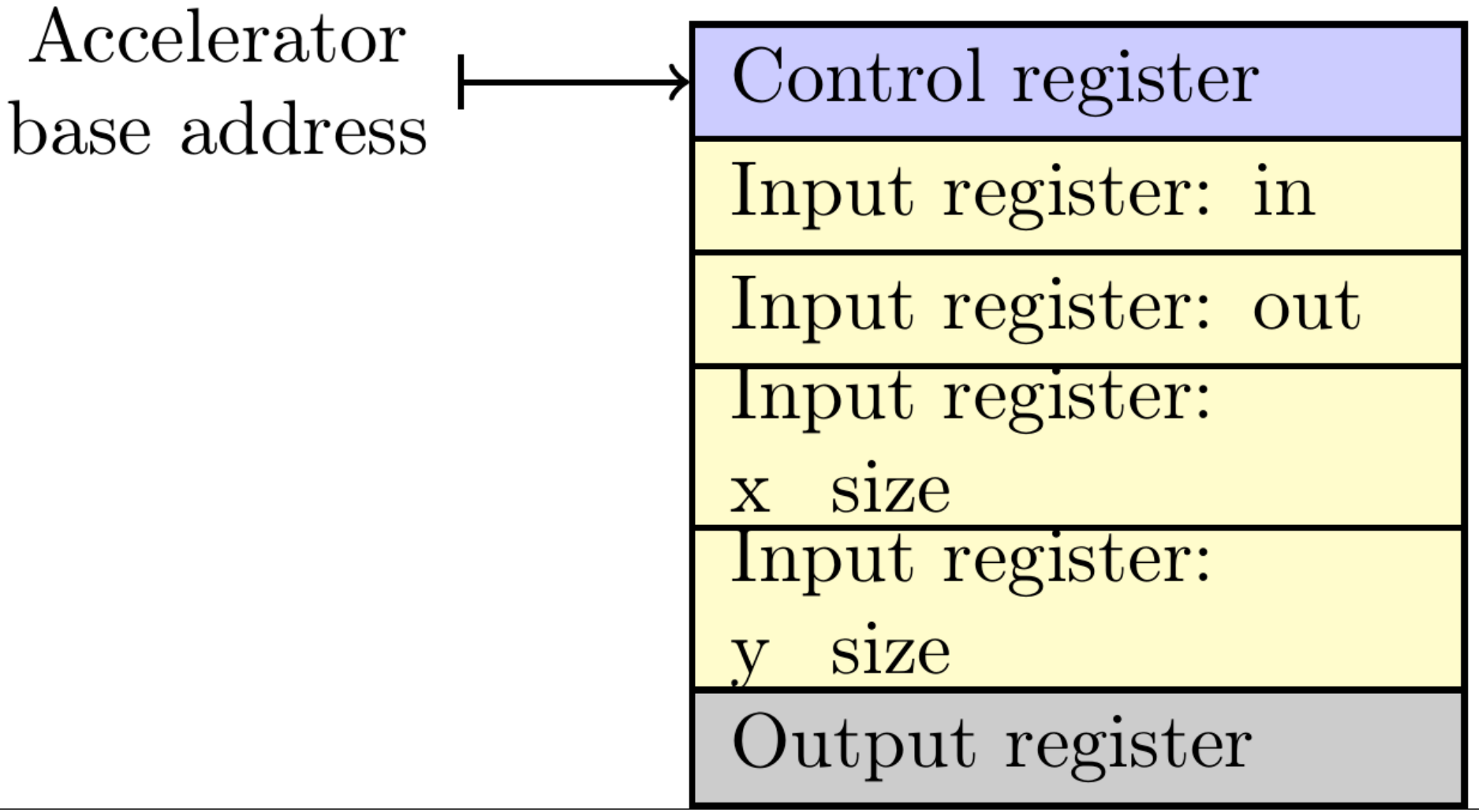
```
cd ~/PandA-bambu/documentation/tutorial_pact_2019/Introduction/sixth
```

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

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);
}
```

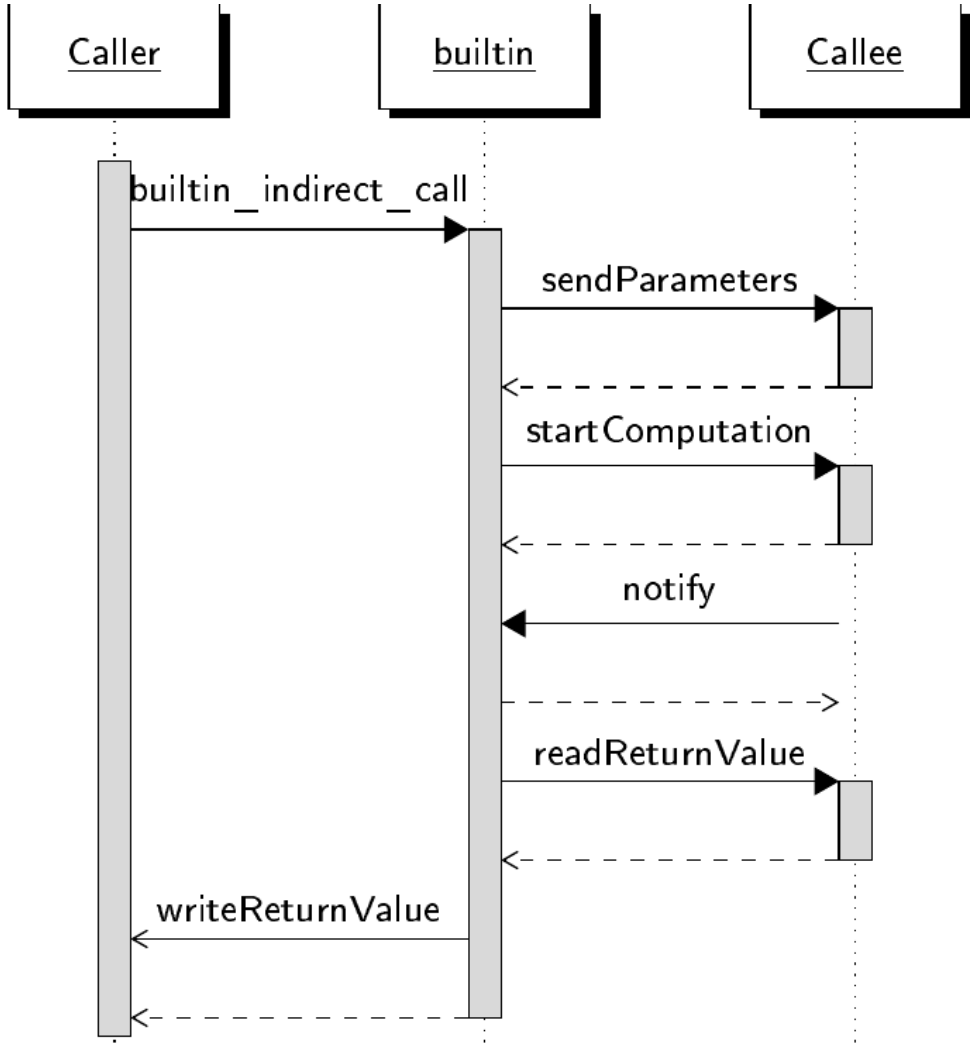
Adding a memory mapped interface to filters



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)

C++ support

- Support of C++ is ongoing:
  - templates
  - C++11 and beyond
  - ac_types from Mentor Graphics could be used
  - ap_types from Xilinx support by wrapping ac_types

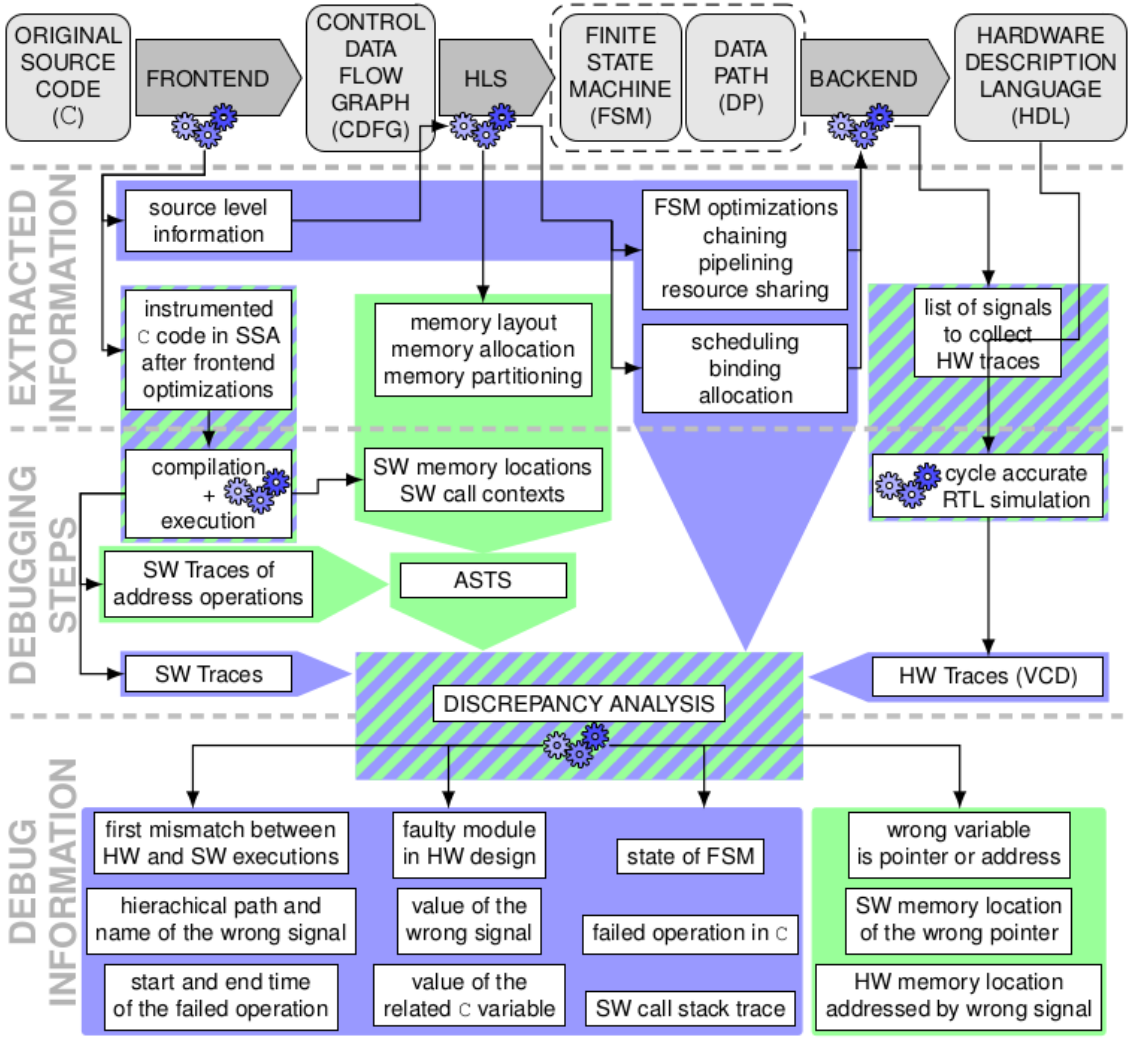
```
#include <algorithm>
int gcd(int x, int y )
 {
        if( x < y )
           std::swap( x, y );

        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

Discrepancy Analysis Debug Flow



Please don’t shoot the PandA player

and visit

http://panda.dei.polimi.it