Chapter 4

ASIP design flow

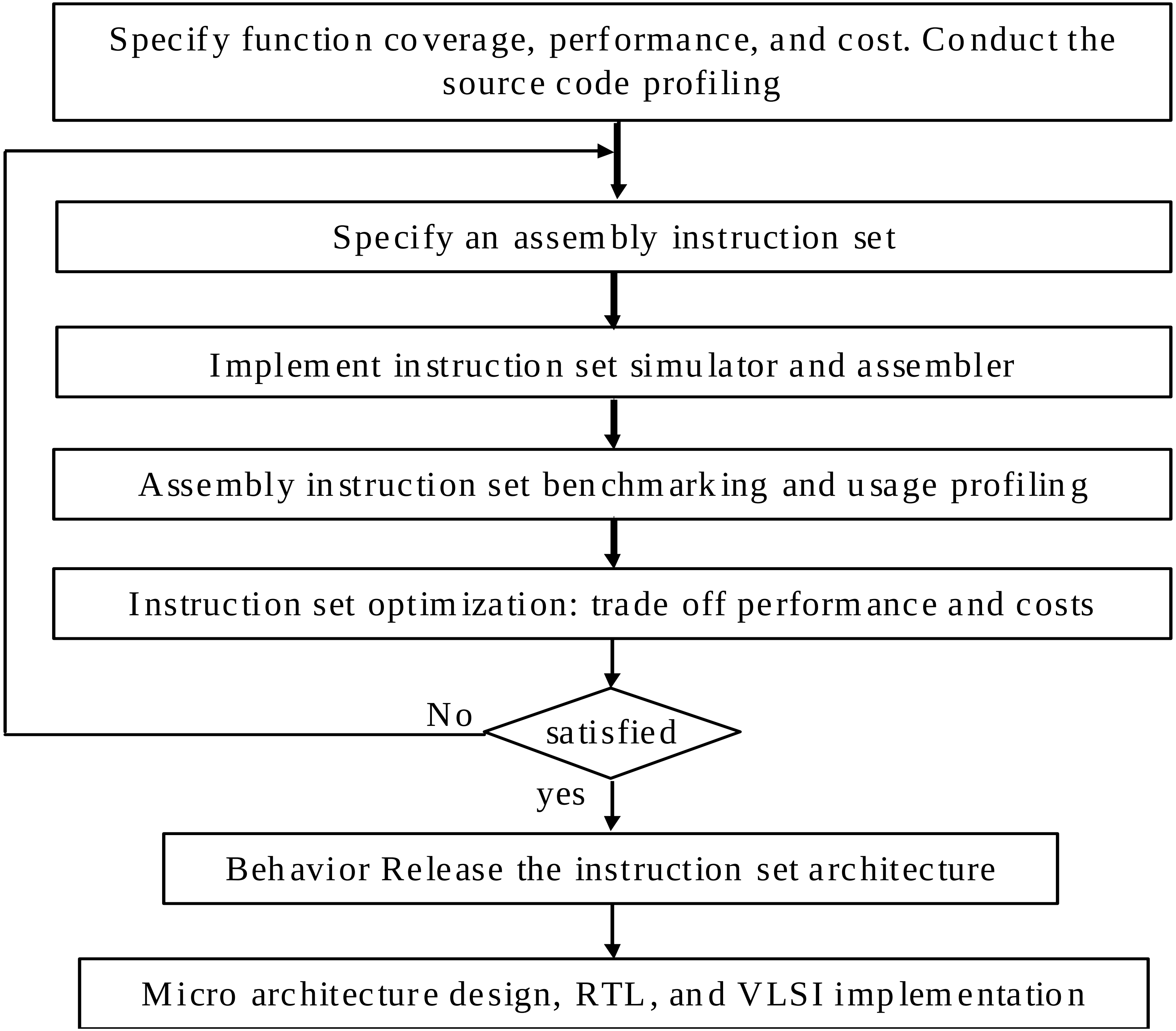
Contents

1. ASIP design flow in general
2. Profiling and architecture selection
3. Instruction set design
4. Toolchain design
5. Microarchitecture design
6. Firmware design

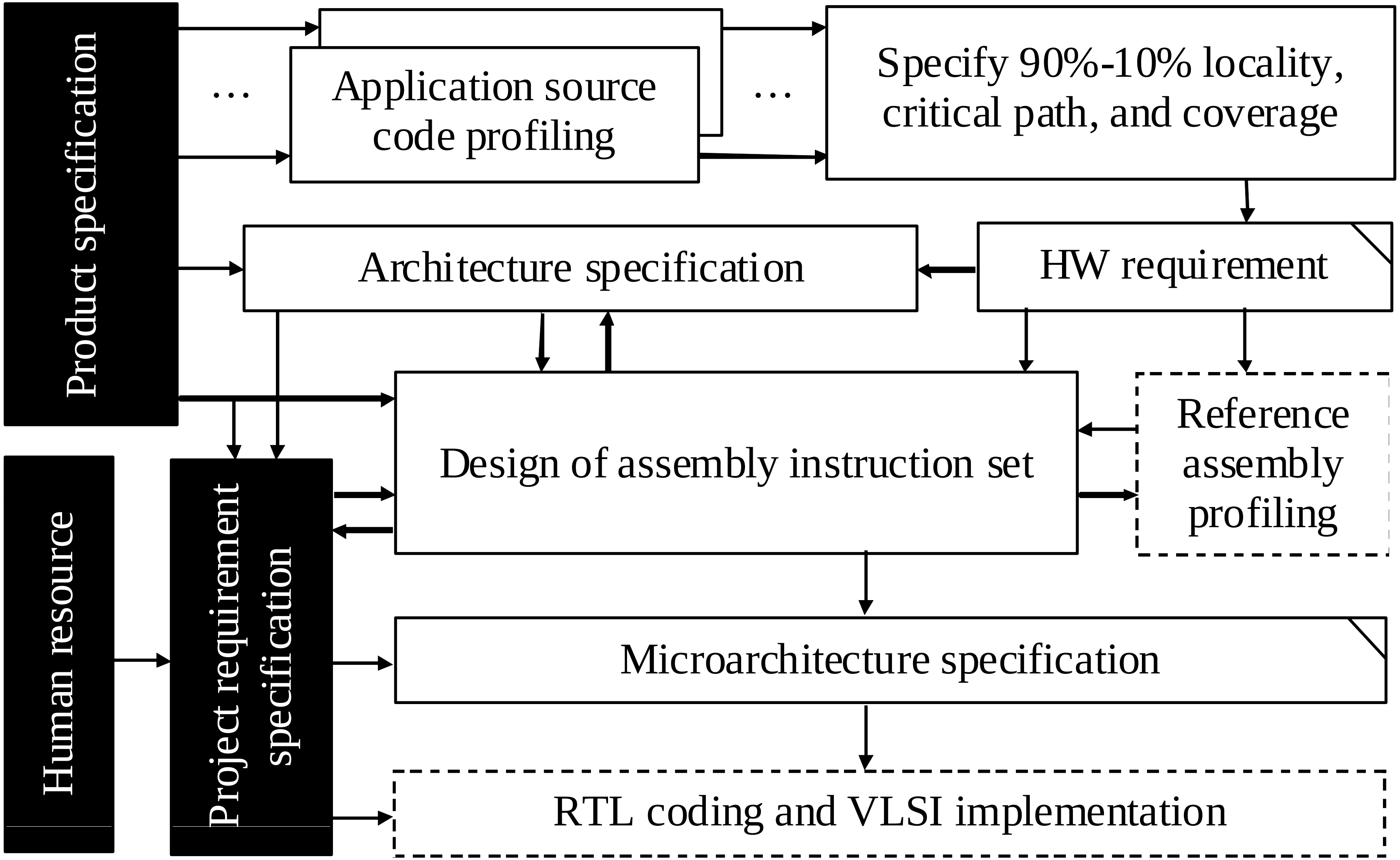
ASIP

- **for sufficient flexibility**
- **for multi-mode applications**
- **for volume productions**
- **For new and future applications**

ASIP design flow for researchers



ASIP design flow for engineers



Understanding Applications

- It takes long time to understand a complete knowledge system of an application
- ASIP designers are hardware engineers rather than application engineers
- It is not trivial that ASIP engineers really understand all system design details
- ASIP designers only need to understand the design cost, including execution behavior, code structure, hardware cost, and runtime cost – through source code profiling

Source code profiling

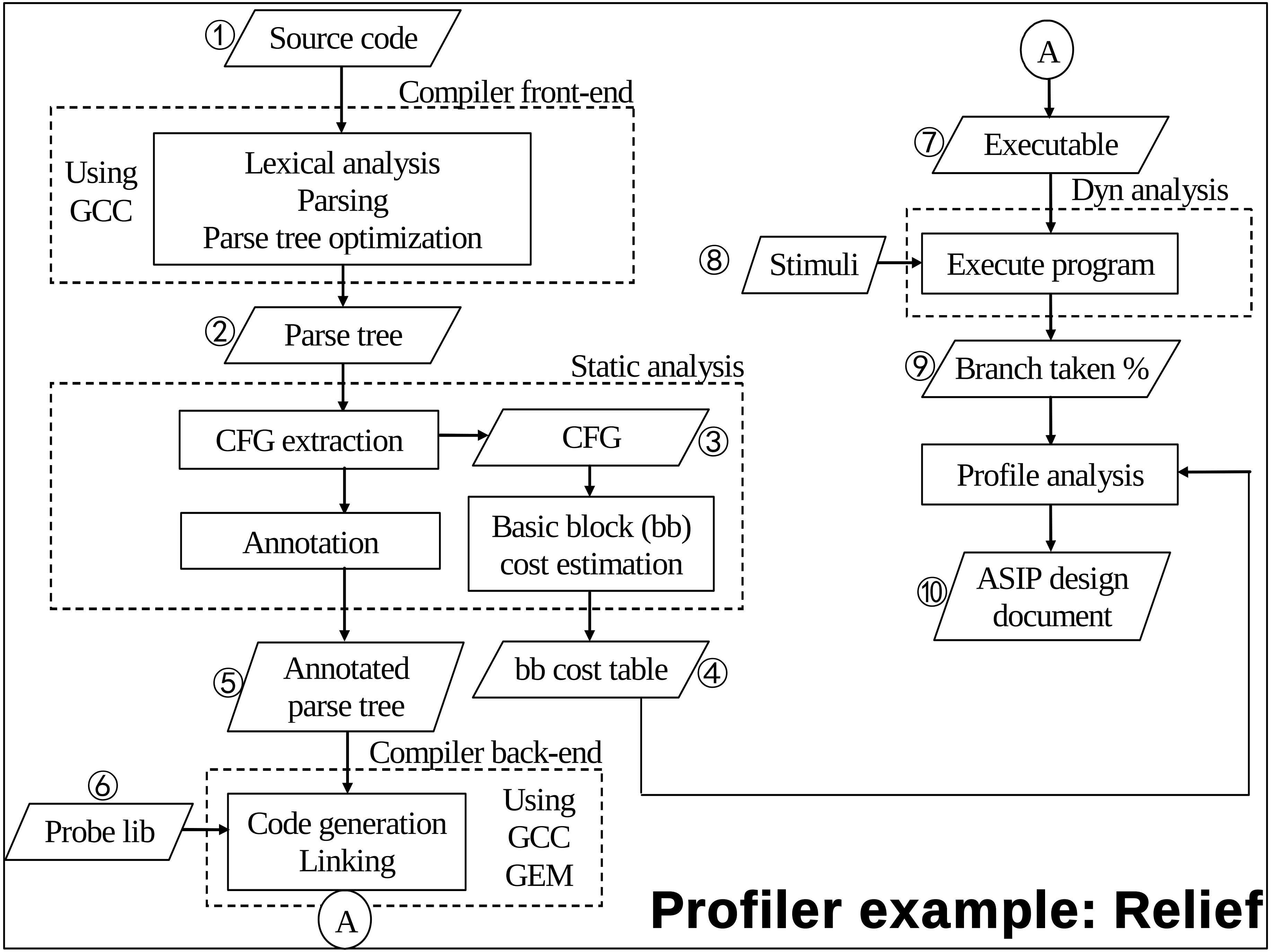
- **Design of assembly language is based on source code profiling**
- **Profiling is a technique to estimate the execution cost and memory cost of source code**
- **Profiling is to analyze the code, expose the code structure and execution behaviors,**
- **The purpose of profiling is to understand the execution behavior and the code structure.**

Static and dynamic profiling

- Static profiling is given by analyzing the source code instead of running it
  - Control flow graph
- Dynamic profiling is performed by executing the source program and accumulating the execution time
  - Through instrumentation

The result from code profiling

- Expose memory accesses, execution time, required operations,
- Expose opportunities of parallelization for further performance enhancement.
- *Coverage* requirement: capability of running different operations
- *Performance* requirement: computing capacity required for certain algorithms
- Profiling result will be the input for architecture selection and instruction set design



Profiler example: Relief

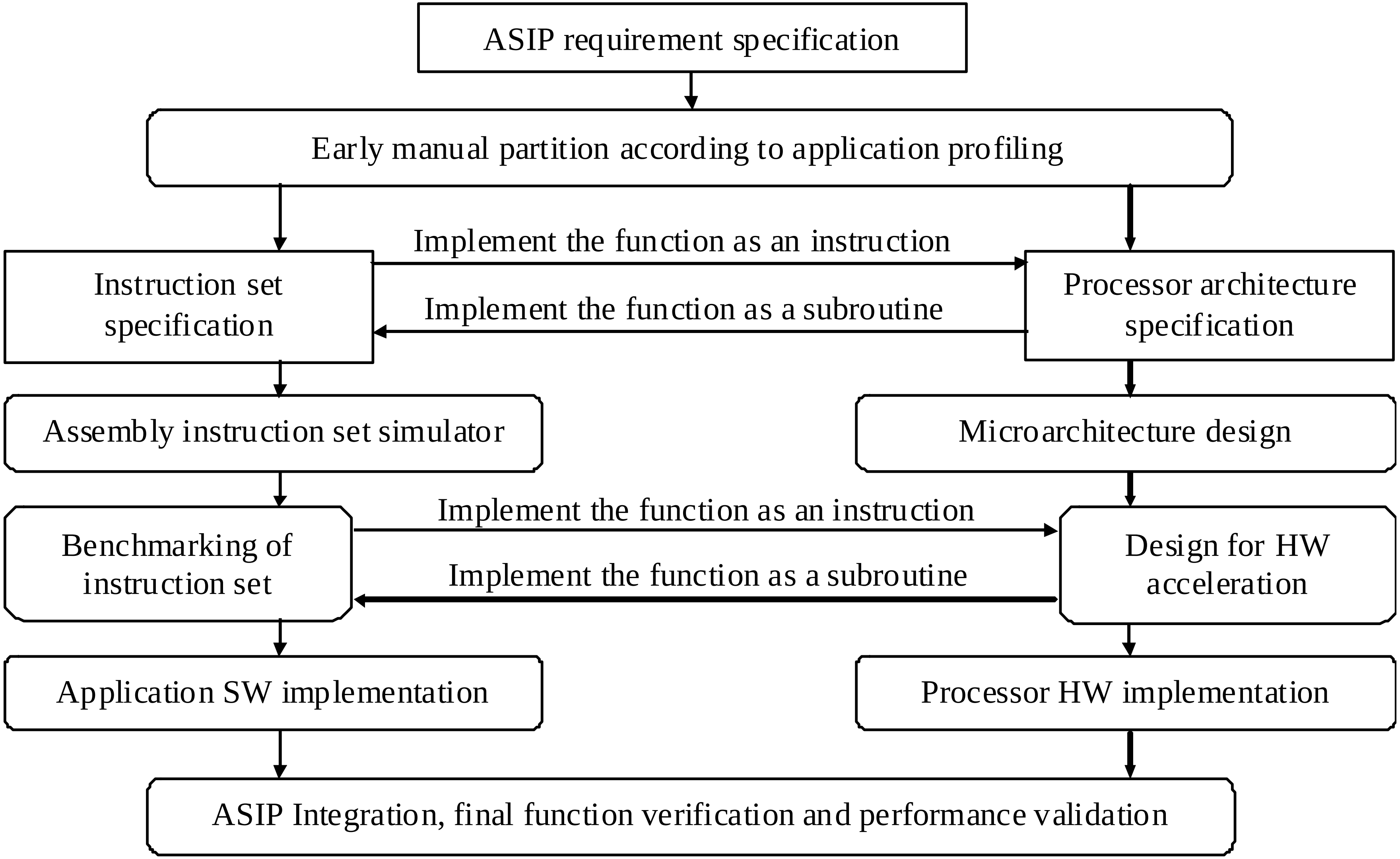
Architecture selection

- Selecting a suitable ASIP architecture for the class of applications involves decisions
  - selecting function modules,
  - interconnecting the modules, and
  - connecting the ASIP to the embedded system
- DSE in the architecture level
- or ISA design

Mapping functions to a HW module

- A system is partitioned into subsystems or functions
- Each functions allocated to a HW module.
- Modules could be either processors or functional circuits.
  - The behaviors of programmable HW modules are described by an assembly language simulator.
  - The behaviors of nonprogrammable HW modules are described by HW description languages.

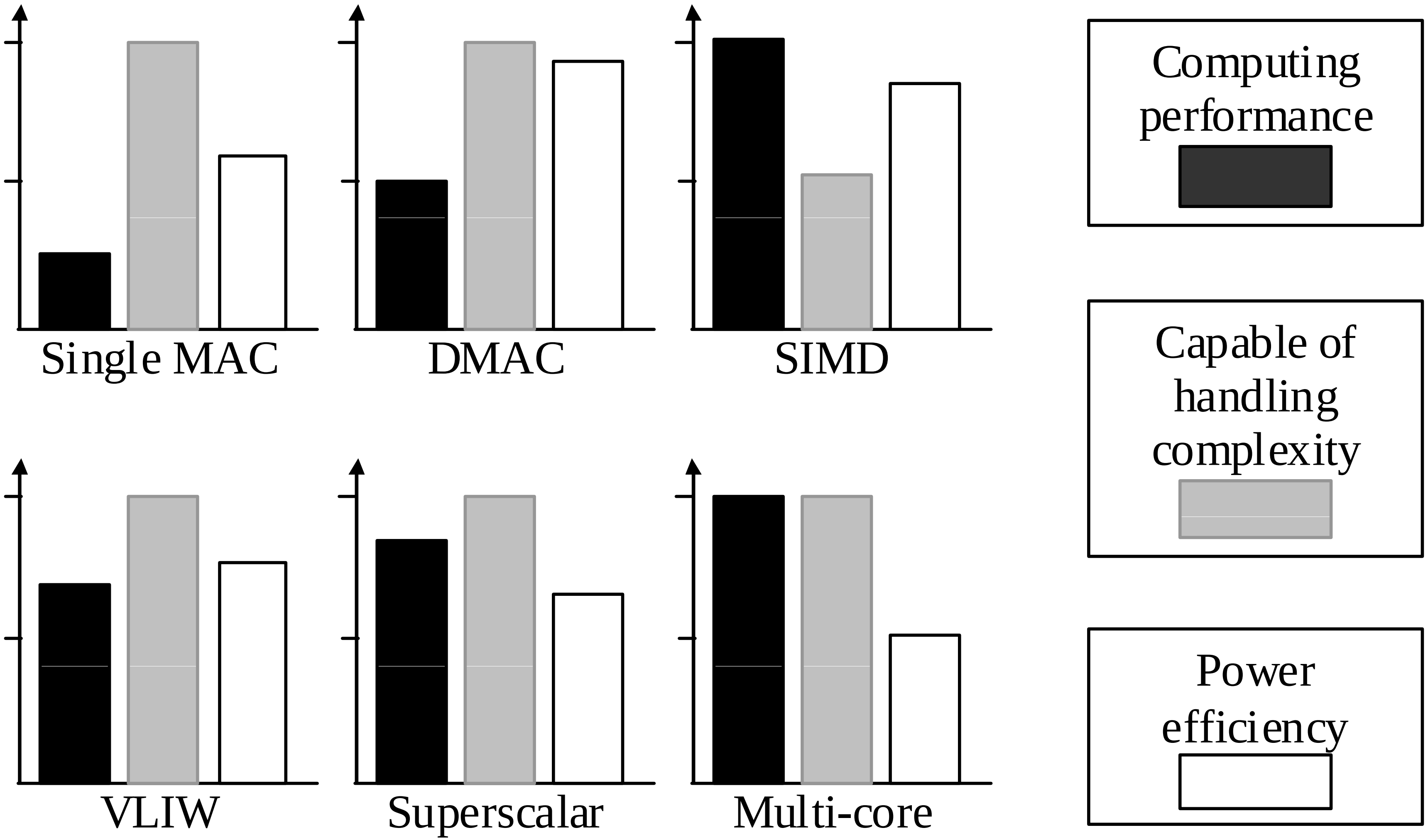
HW/SW co-design for an ASIP



Architecture templates

- Characteristics to be considered
  - computing performance
  - addressing performance
  - handling control complexities
  - power efficiency
  - scalability and how easy to be integrated

Select architecture based on templates



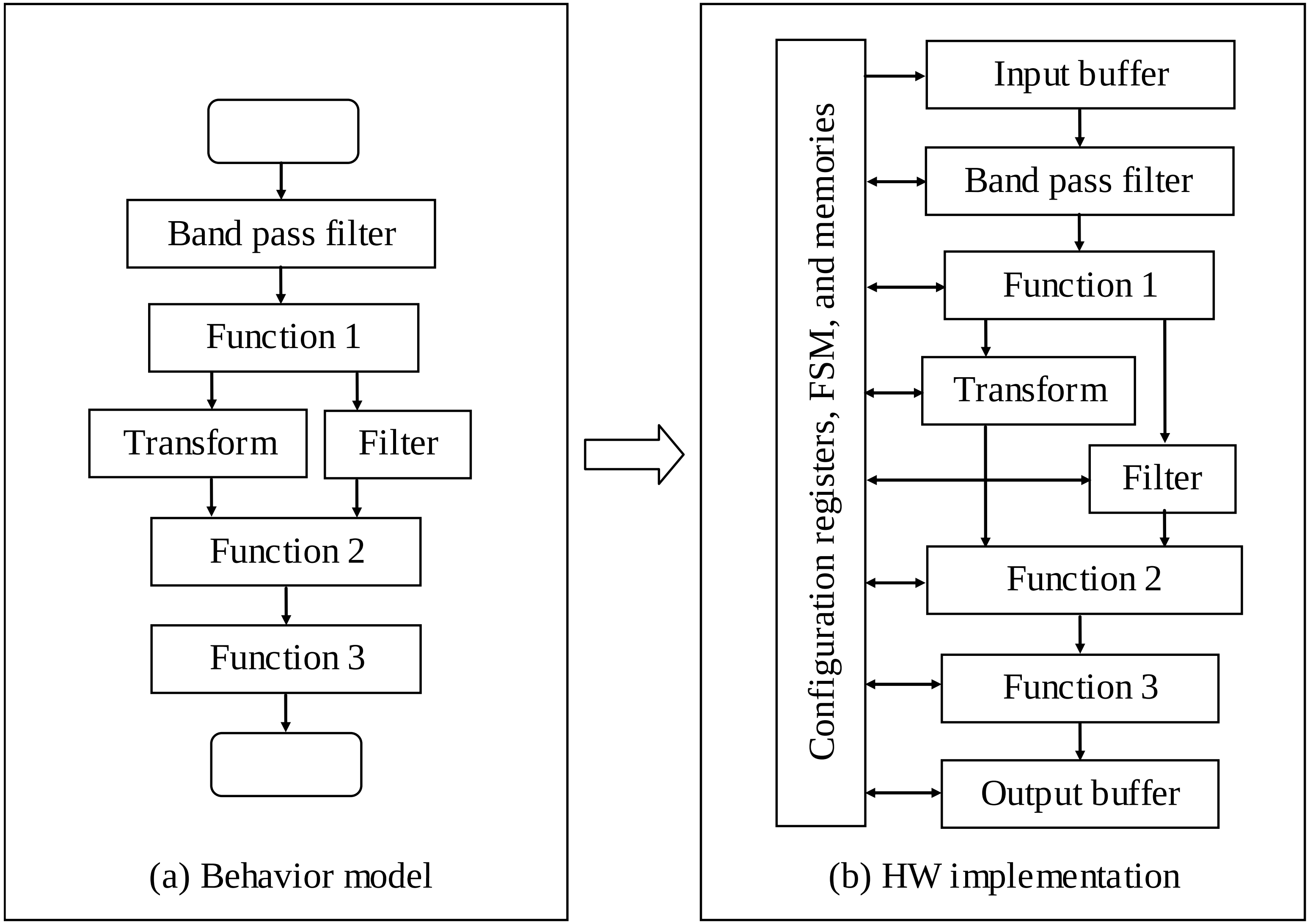
Control & Data processing

- When control complexity cannot be separated from data processing
  - VLIW or superscalar architecture is preferred
- If control complexity can be separated from data processing
  - use a RISC and a SIMD machine
  - use a RISC with SIMD datapaths

Task flow architecture

- Direct implementation of control flow graph
- Suitable when
  - Programming cost is low and
  - Complexity of hardware and system verification is manageable
- Useful when input data rate is too high to employ the conventional architecture
- Not flexible

Task flow architecture



Configurability and programmability

- Configurability: ability to change system functionality by external control inputs
- Programmability: ability to execute programs
- Configuration control is relatively stable: definitely not changing every cycles
- Program can change the hardware function at every cycle.

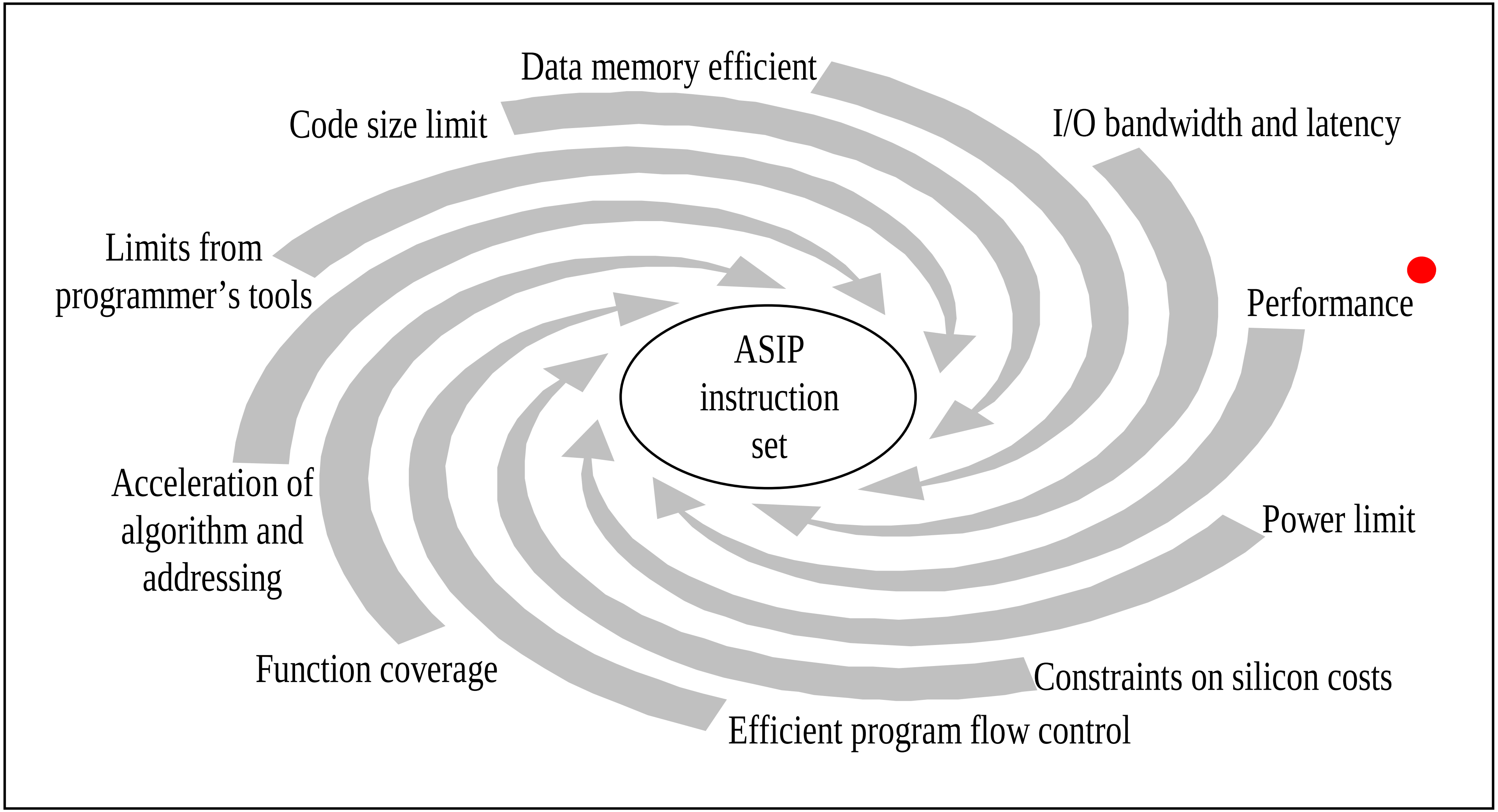
Generate a task flow architecture

- Formulate a task stream using CFG
- Balance load of each task step
- Identify dependencies and schedule the task chain with considerations of load balance
- Specify function modules and FIFO buffers between function modules in the streaming chain, expose and specify control signals
- Design FSM to generate control signals

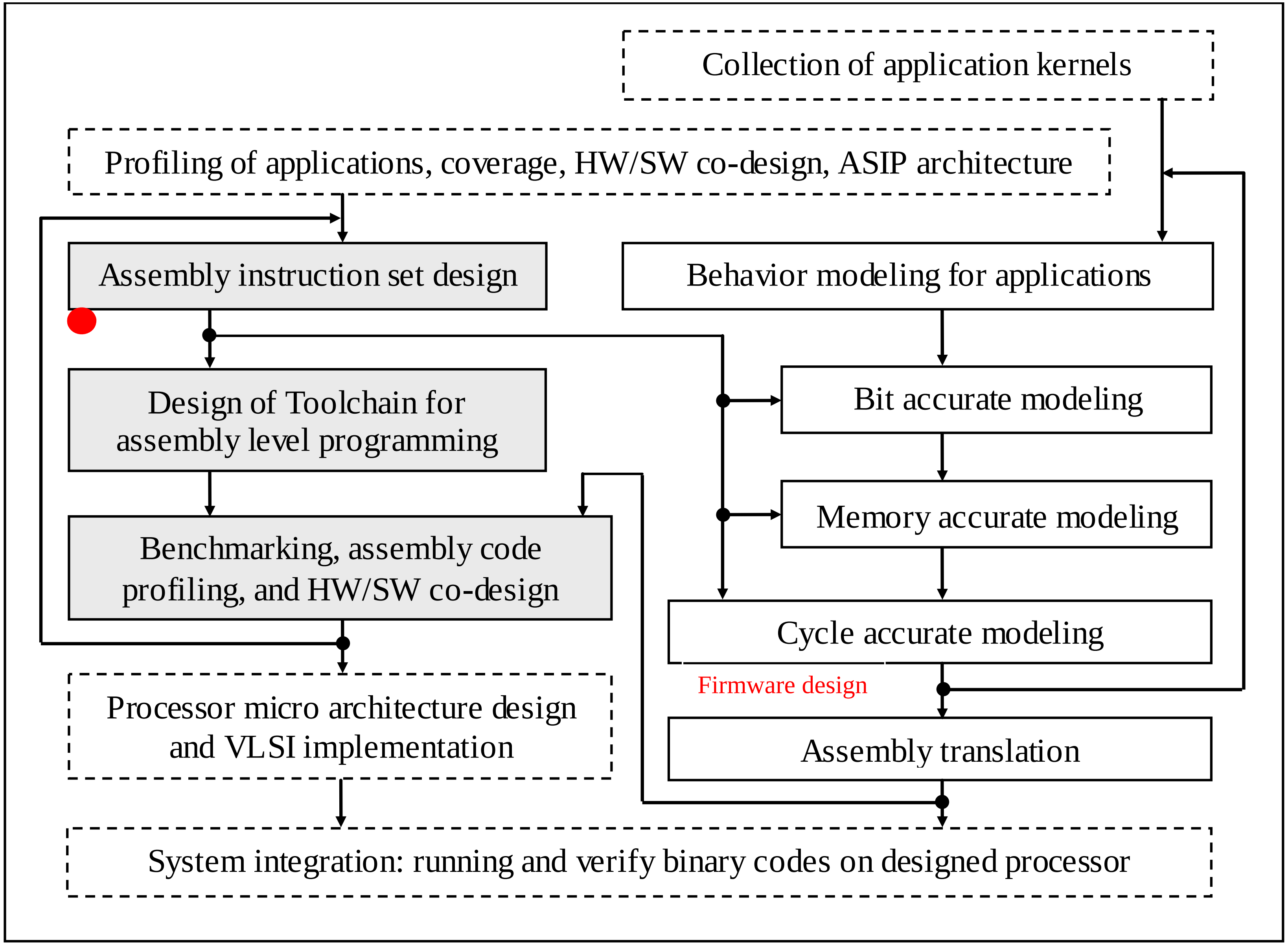
Designing instruction sets

- After the ASIP architecture is selected
- Input: profiling results and architecture
- Instruction set design includes
  - Arithmetic instructions
  - Memory accesses
  - Addressing
  - Program flow controls
  - I/O instructions
  - Accelerator control intructions

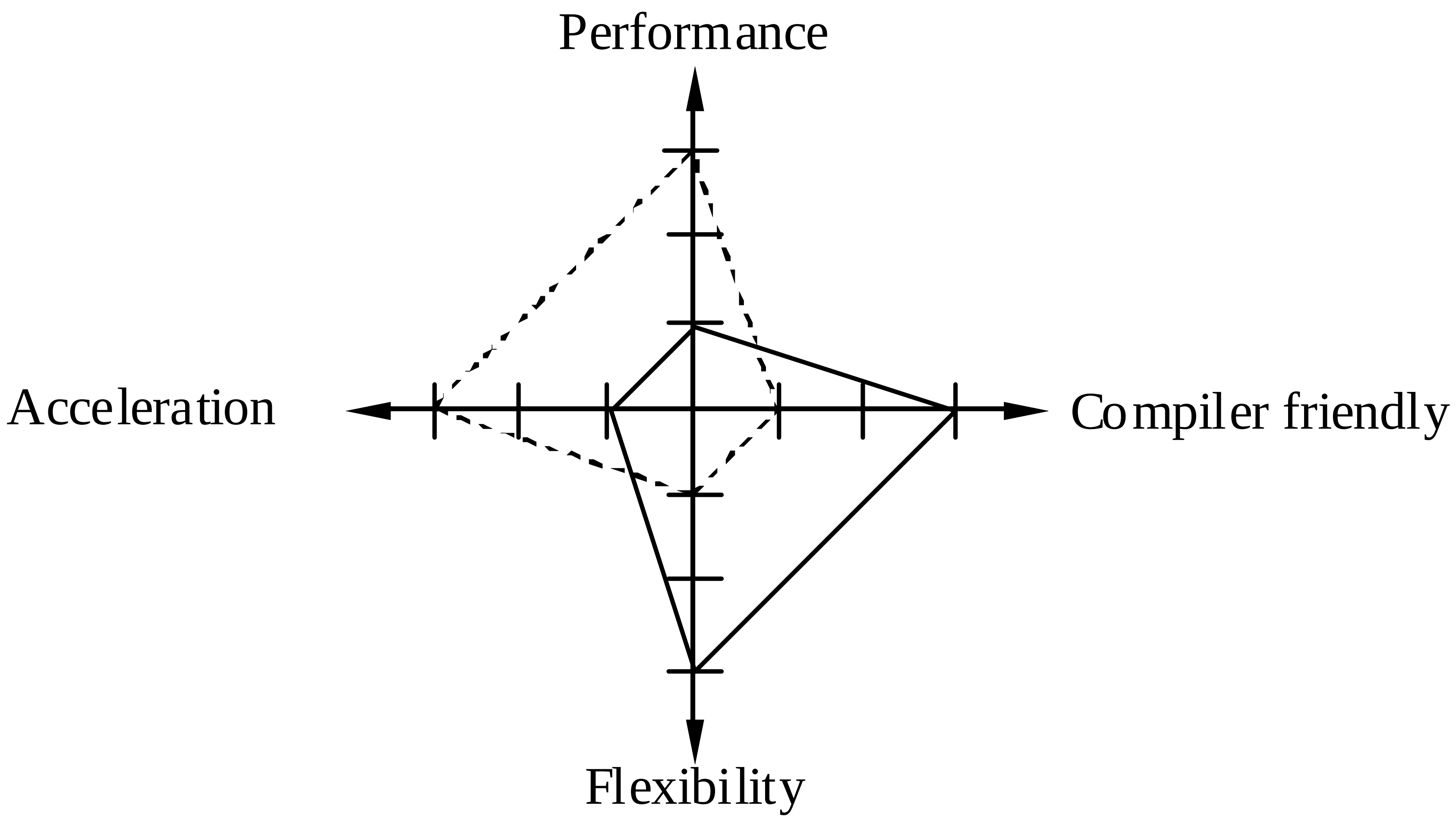
Inputs and requirements



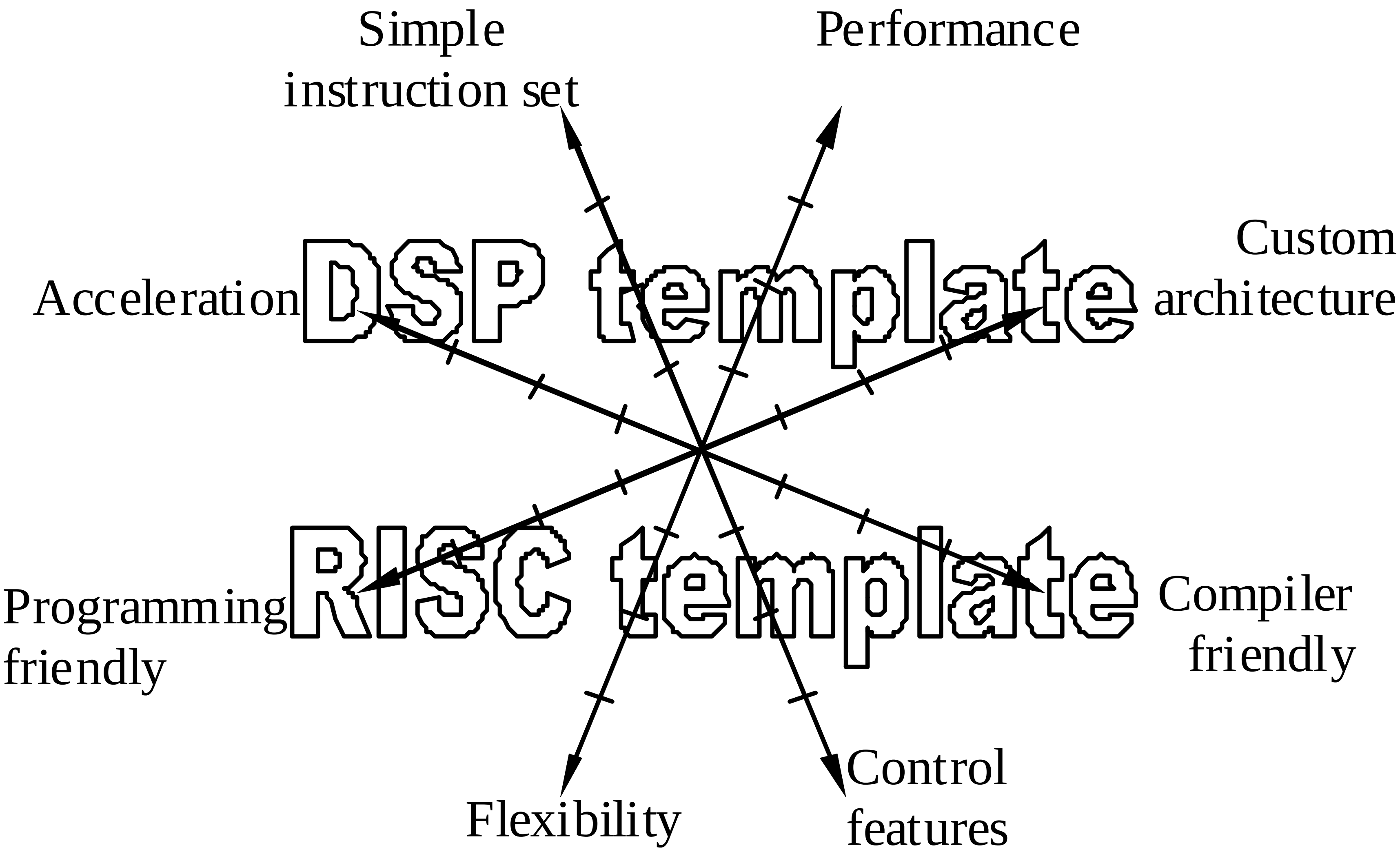
Simplified DSP ASIP design flow



Trade off among requirements



Select an instruction set template



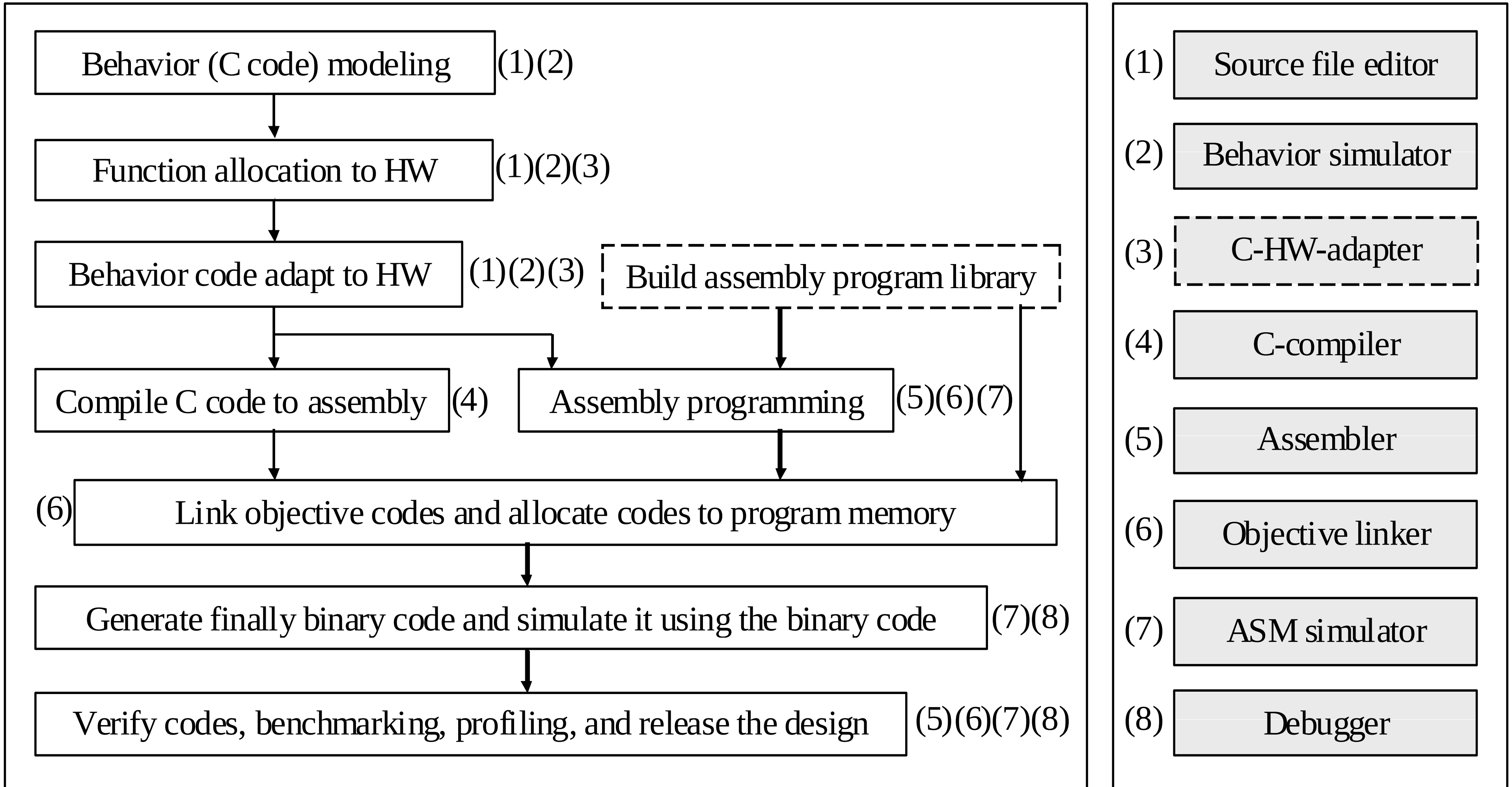
Programming toolchain

- C compiler
- Assembler
- Linker
- Instruction set simulator (ISS)
- Debugger
- Integrated design environment (IDE)

Benchmarking and Assembly code profiling

- Benchmark: program designed to measure the ASIP performance
- Benchmarking: check the cost and performance of the kernel code
- Assembly code profiling: expose the statistics of the instruction usages and the SW cost of the application

Relations between Toolchain and FW design flow



(a) The method translating source code to qualified assembly code

(b) Toolchain

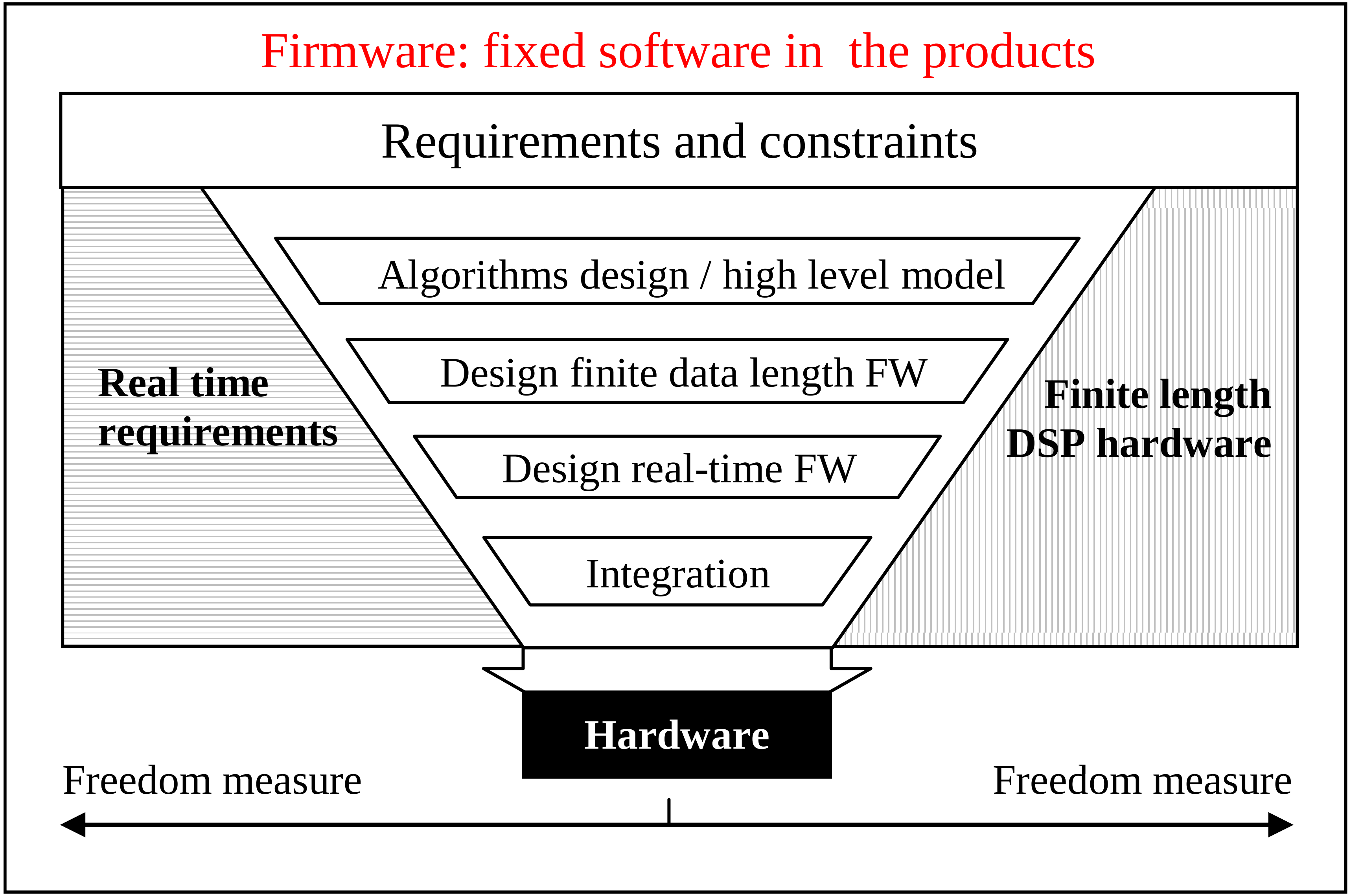
Adaptation of the c code to HW

- Adapt to the ASIP hardware features to avoid confusing the C compiler
  - Finite-length data type
  - Parallel or accelerated instructions
  - Memory size constraints
- A C-HW adapter as a special parser
  - Parsing results can be used to modify the C source code.

C-HW adapter

- Expose three cases to guide the designers
  - the legacy hardware features of early design
  - the opportunities to use compiler features or acceleration features of the selected hardware
  - The opportunities for parallel executions and memory accesses
- To reduce the gap between the C code and assembly code
  - Library functions and special library adapting ASIP features

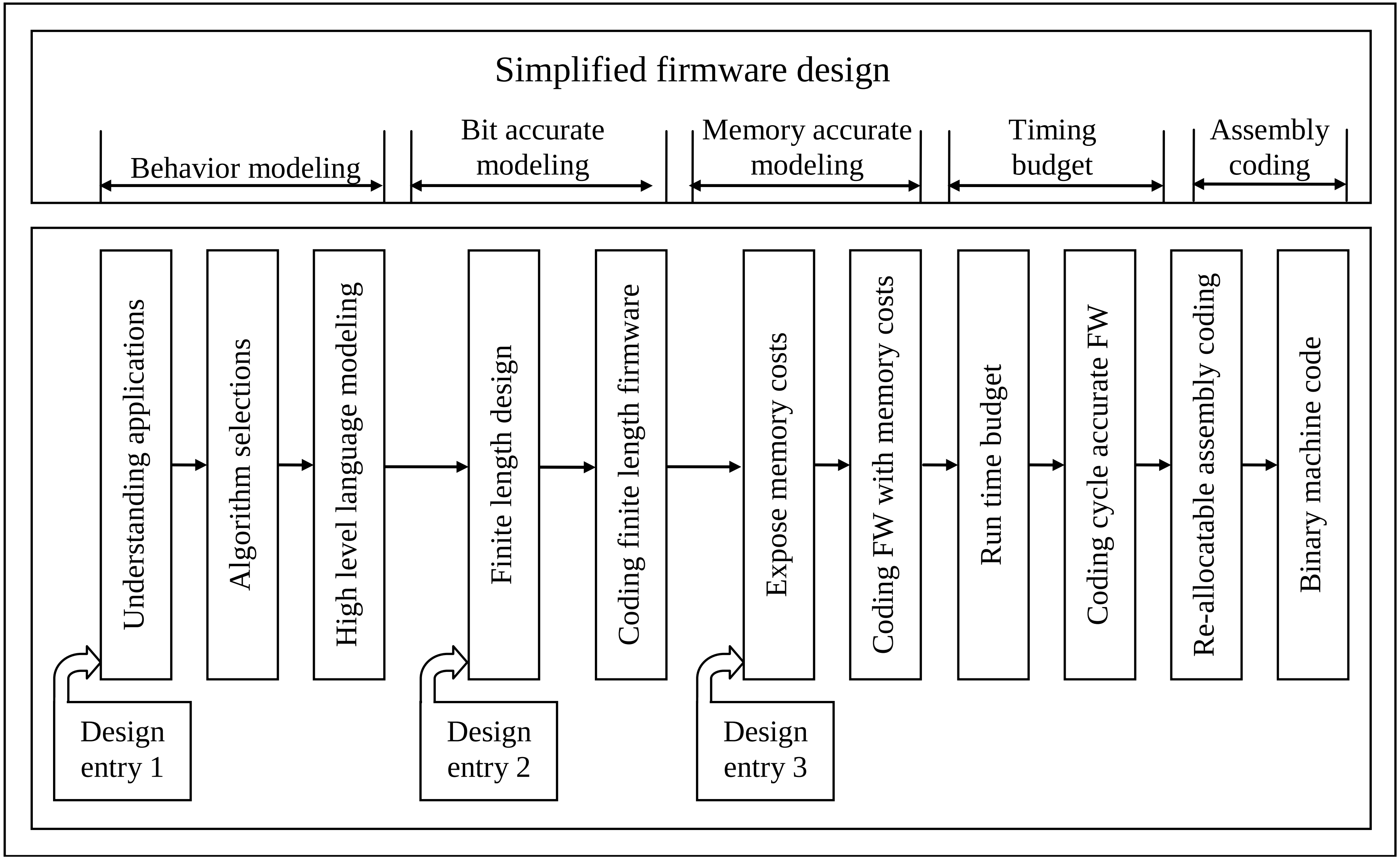
FW design


Firmware: fixed software in the products
Requirements and constraints
Algorithms design / high level model
Design finite data length FW
Design real-time FW
Integration
Real time requirements
Finite length DSP hardware
Hardware
Freedom measure
Freedom measure

FW design

- Behavior modeling
- HW dependent SW
  - Bit accurate source code
  - Memory accurate source code
  - Cycle accurate code
- Assembly coding and optimization

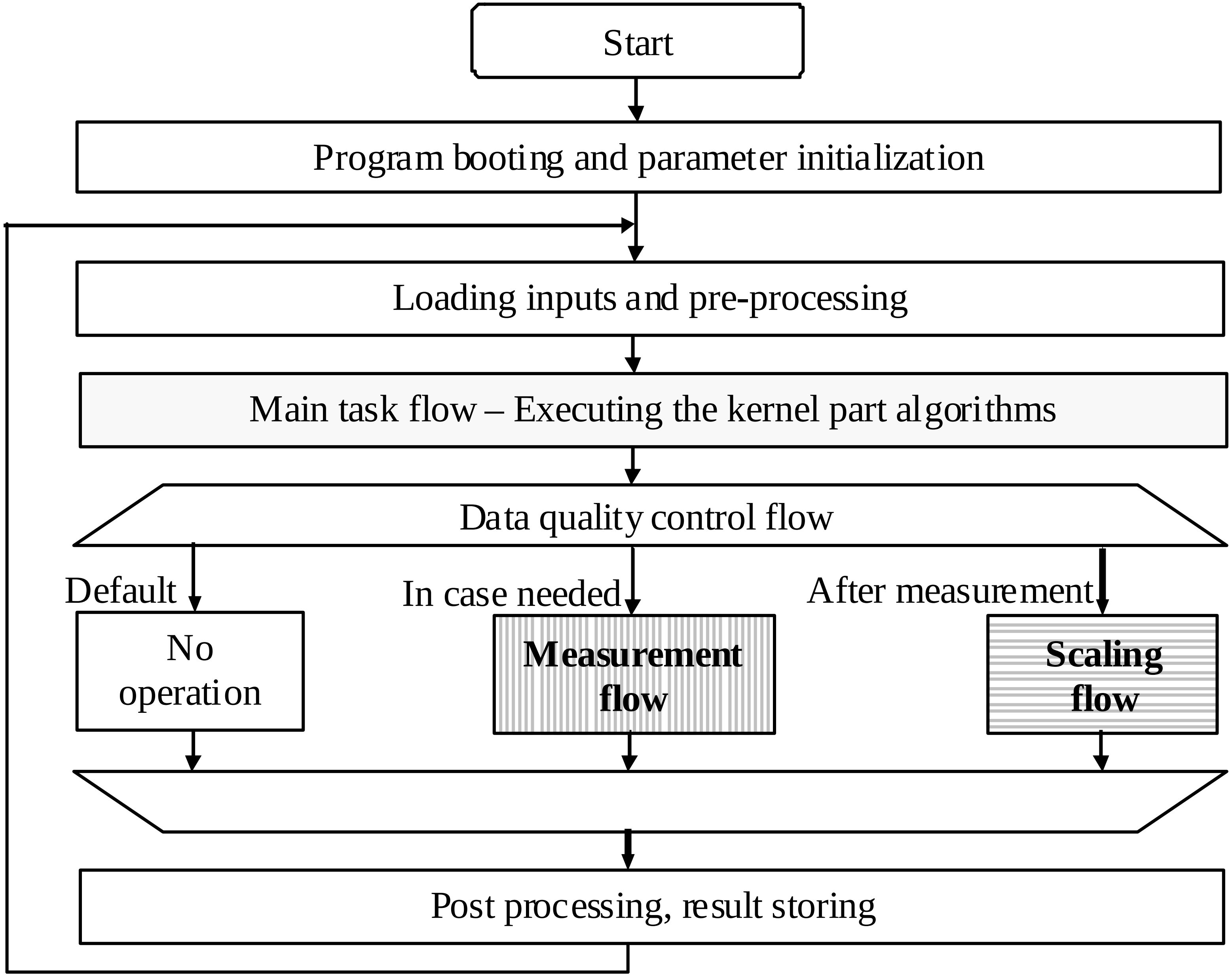
FW design flow (single application)



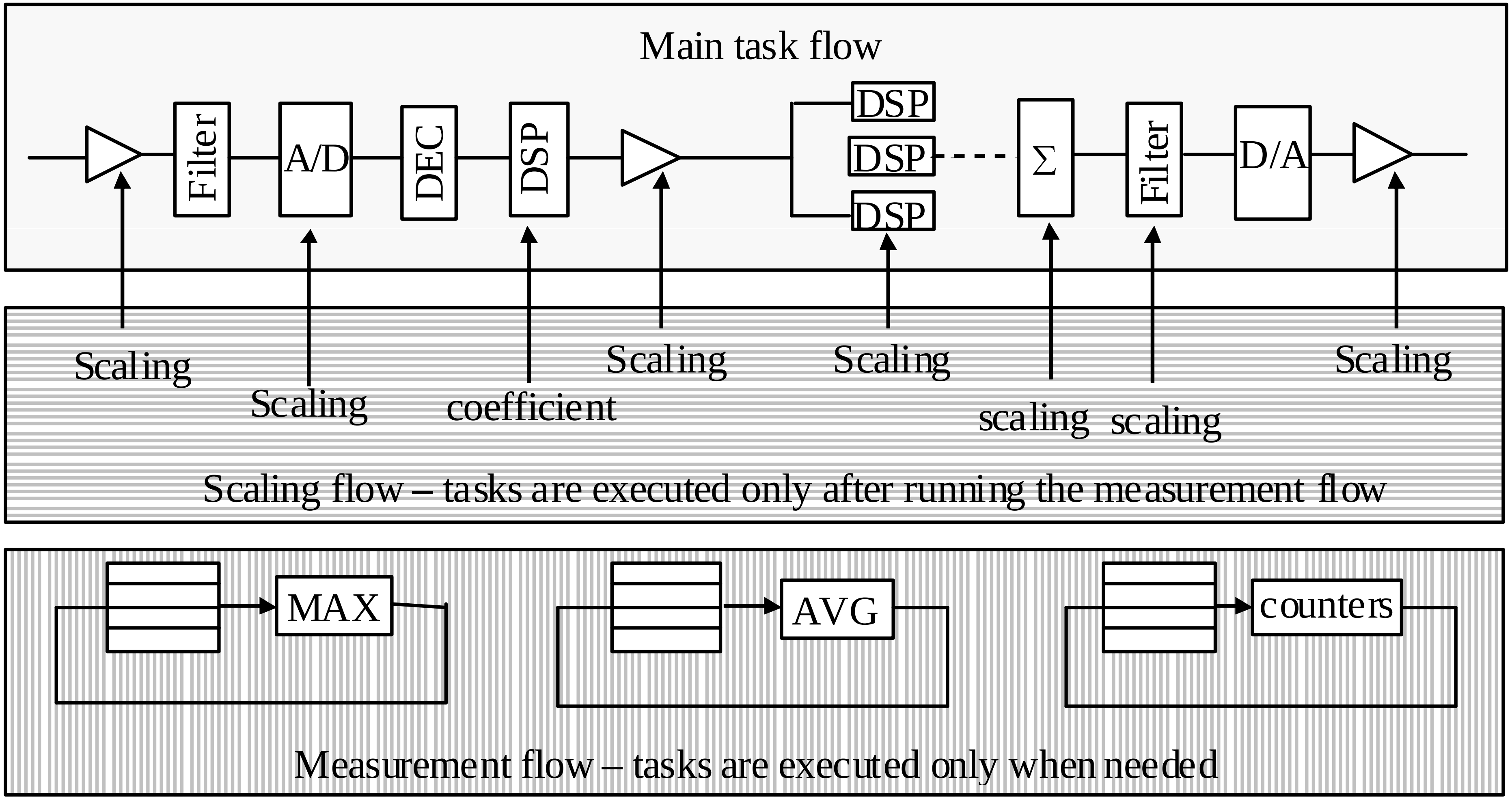
Bit accurate finite precision FW

- adapt the C-code to the finite precision hardware and compare it to the original code, for example, a floating-point vesion
- Find poor precision or SNR on the results
- Improve the precision by:
  - Inserting quality measurements subroutines
  - Inserting data scaling subroutines

Firmware in a fixed point processing



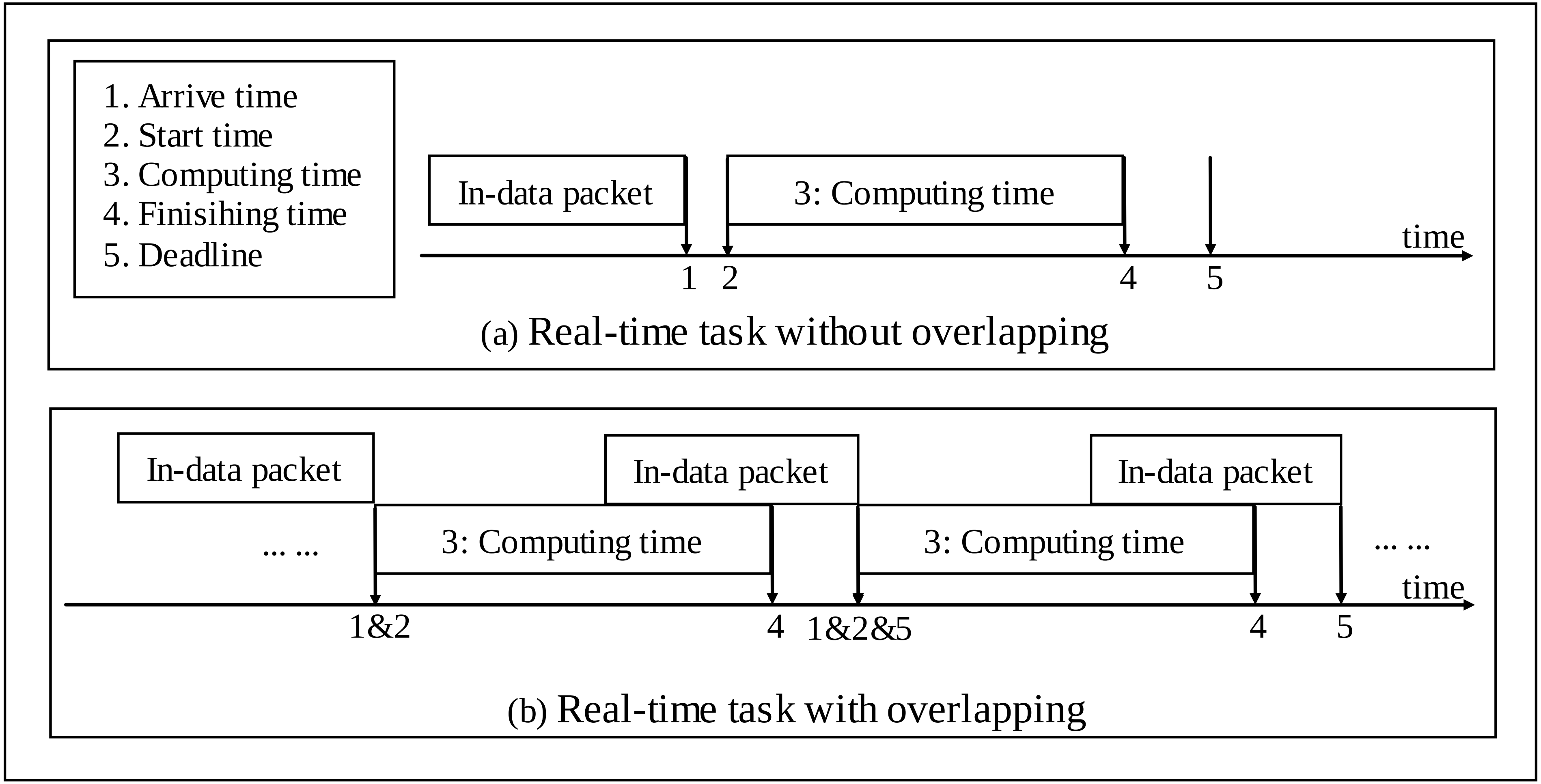
Added quality control codes



Memory access accurate FW

- Much memory accesses and address computing for the accesses are hidden in the C code
- A memory-accurate model is essential for parallel processing: parallel memory accesses
- Early expose the memory cost is essential for
  - Execution time estimation
  - Memory cost estimation (ASIP design)
- Design for memory subsystem will be discussed in chapter 16, 18, and 20.

Real time firmware parameters



(a) Real-time task without overlapping

(b) Real-time task with overlapping

Data streaming: (Input; Computation; Output)

How can we find a best instruction set?

- Evaluation of an instruction set
  - Cycle cost and memory usage
  - Suitability for specific applications
- How to evaluate a processor
  - Good assembly instruction set
  - Good (open and scalable) architecture
  - (Max clock frequency, low power, less area)
- Use benchmarking techniques!

General benchmarks

- Algorithm benchmarks/kernel benchmarks
- Normal precision and native word length
- What to check:
  - Cycle costs of kernels, prologs, and epilogs
  - Program/data memory costs
- Algorithms including
  - FIR, IIR, LMS, FFT, DCT, FSM

Third Party Benchmarks

- BDTI: Berkeley Design Tech Incorporation
  - Professional hand written assembly
  - http://www.bdti.com
- EEMBC (the EDN Embedded Microprocessor Benchmark Consortium), fall into five classes:
  - automotive/industrial, consumer, networking, office automation, and telecommunication
  - http://www.eembc.org

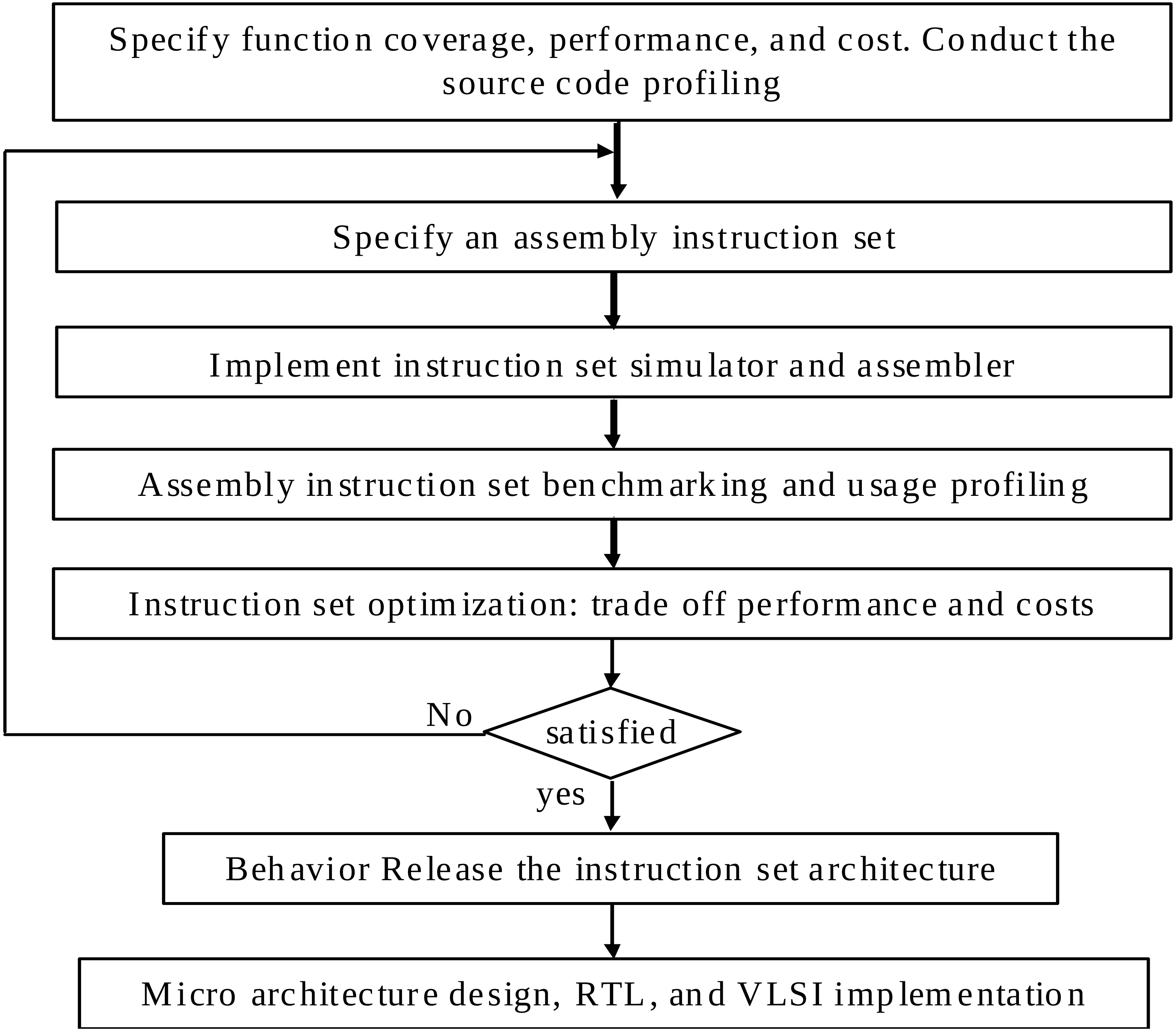
Microarchitecture design

- The microarchitecture design of an ASIP is to specify the hardware implementation of the assembly instruction set into core functional modules.
- The input of the microarchitecture design
  - ASIP architecture specification and
  - Assembly instruction set manual.
- The output of the microarchitecture design
  - Microarchitecture specification for RTL coding.

Microarchitecture design

- **Step 1:** Partition each assembly instruction into microoperations, allocate each microoperation into corresponding hardware modules
- **Step 2:** Collect all microoperations allocated in a module and specify hardware multiplexing for RTL coding of the module
- **Step 3:** Fine-tune intermodule specifications of the ASIP architecture specification and finalize the top-level connections and pipeline

Review of ASIP design flow



Review

- **ASIP design flow in general**
- **Profiling and architecture selection**
- **Instruction set design**
- **Toolchain design**
- **Microarchitecture design**
- **Firmware design and benchmark**

Understand Applications