			
Title:	Qualification Task	Revision 1.6	Page 1 of 2

Qualification Task for Applicants to NXP for C++/Java/Python programming

Language: ☐C++ ☐Java ☐Python

Objective

The document contains specification of the task that should be solved individually by the applicant. The task described below represents a common problem that is solved in our department.

1 Overview

The task is to design and implement an algorithm in C++, Java or Python language that finds the best clock frequency for a peripheral of a microcontroller.


1.1 Model of peripheral clocks

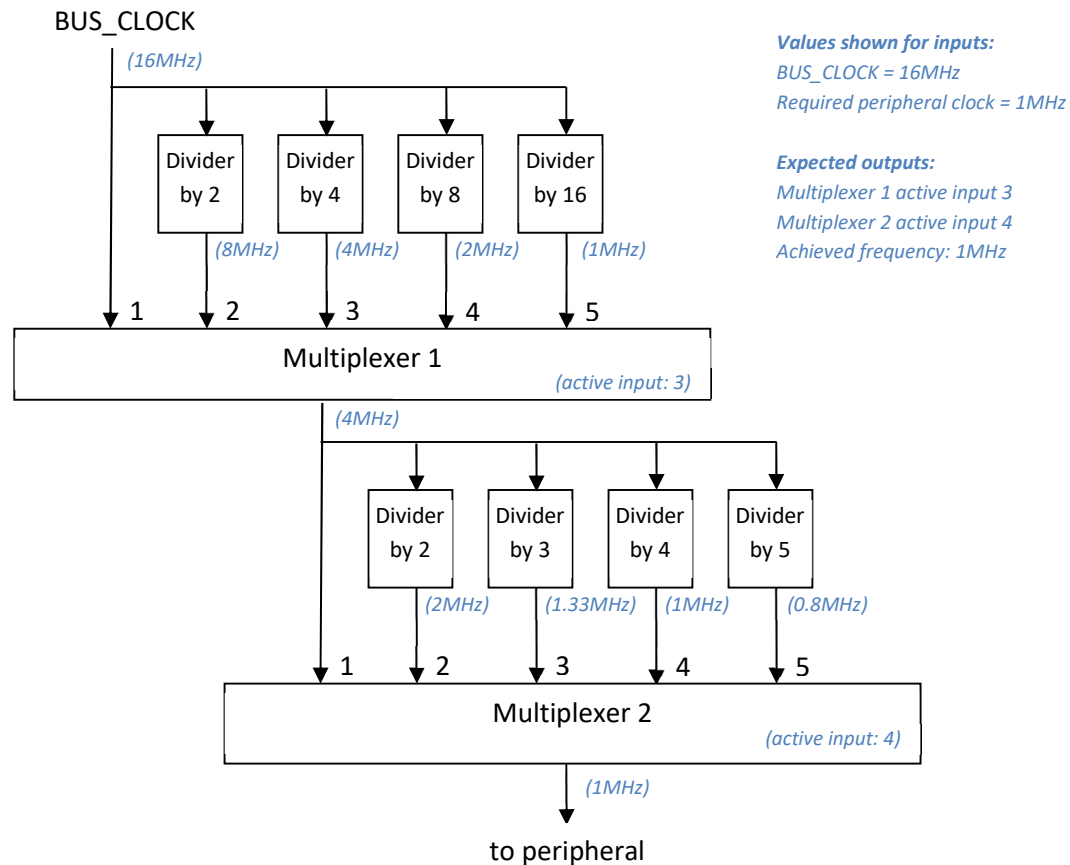
On-chip microcontroller peripherals need clock signals for their functionality. The source of that signal is usually microcontroller's bus-clock. Most of peripherals need to adjust (decrease) the frequency of bus clock for their correct function. This "adjustment" is done through a system of prescalers (dividers) and multiplexors. A prescaler divides an input frequency by a constant value (in this task). The output from a multiplexor is one of input signals.

2 Expected functionality

The algorithm should find the best configuration of multiplexors that produces the frequency closest to the required one. Inputs for the whole algorithm are: input clock frequency (BUS_CLOCK) and required frequency for the peripheral. The results of the algorithm are configurations of multiplexors and the really achieved frequency for a peripheral.

Figure below is a timing model that should be implemented (values in parentheses represent an example of execution).

		
Title: Qualification Task	Revision 1.6	Page 2 of 2



3 Requirements for the implementation

Applicants should demonstrate their knowledge and experience by doing good analysis, implementation and documentation of the problem. Applicants may freely extend the task if it presents better their capabilities.

3.1 Criteria for the evaluation

- The primary criterion for the evaluation of this task is an extensibility of the provided solution:
 - The algorithm should be capable to process also other timing models (that consist of the same timing entities).
 - The source code should be easily maintainable.
- Decomposition and modularity. The applicant should do a proper decomposition despite of small complexity of the task.
- Technologies used in the implementation.
- Readability - source code should be well readable and understandable.
- No user interface is required (inputs can be fixed in the source code).