Assignment Homework 5

Due date: Check Canvas

(*Preliminary*) Student Version – "Finite State Machines"

Please team up with a fellow student and start working on the following tasks (100 pts total). In addition, you *must* consider the following:

- Vivado creates a default template including a comment block with:
  - "Company" – insert team name/number
  - "Engineer" – put team member names here
  - "Design Name" – based on task description (*do not leave empty*)
  - "Project Name" – Assignment 5
  - "Description" – based on task description (*do not leave empty*)
- To prepare your assignment for submission through Canvas, you must use (in Vivado): `File > Project > Archive`
- Preparing your HDL project through this process is especially important when working with IP cores such as DSPs and BRAMs!
- You work as a team. Please alternate between one student creating the testbench, the other student creating the implementation, or other aspects of the implementation.

We will start this homework assignment together through Zoom as a lab. Please try to follow along and ask questions if you get stuck somewhere. Your homework assignment includes all tasks mentioned here (even those, that you manage to complete during our joint lab time).

1 Task

All previous assignments could be solved with simple control logic that was regular. While your implementations will operate in a certain state, they can be (from a practitioner's point of view) hardly be called a state machine, instead, we only called it control logic. In contrast, there are many practical use-cases where a more complex ("true") state machine is needed. For the following assignment, please check the documentation and example by Xilinx*.

State machines is a topic of academic debate (Mealy vs. Moore) and you most likely discussed it in another class – at least, the ECE 272 materials prominently include this topic. For industry-related scenarios, you will most likely never consider these academic differences. Instead, you will need to distinguish the following scenarios in your design:

* `https://docs.amd.com/r/en-US/ug901-vivado-synthesis/FSM-Example-Verilog`

- One-process FSM:
  - Pro: Only one process (VHDL) or sequential block (always @(posedge CLK))
  - Pro: cannot generate latches and is shorter
  - Con: complex FSM might lead to chaotic code
  - Con: might result in incomplete FSM more easily
- Three-process FSM (but also Two-process):
  - Processes: "change state" (clocked), next state (unclocked), signals (unclocked)
  - Pro: separates control logic and signal changes
  - Con: complex FSM, modifying multiple blocks needed
  - Con: can result in latches

  As a general rule of thumb: smaller FSMs should always be implemented as a single process (block). For larger and sufficiently complex FSMs, a three-process FSM might be the better choice. A lot depends on "style" and you will need to find your own personal preference. Finite state machines is also a topic where you will see the advantages of VHDL and SystemVerilog, as they know the data type "enum", whereas Verilog needs to use a somewhat limiting workaround of using a parameter. Still, the focus of this assignment should be implementing the FSM in Verilog and the example provided by Xilinx is only in Verilog, too.

1.1 Track 1 (474): Bus Arbiter State Machine (VLSI 2023 edition – week 3)

Please implement the bus arbiter with the state machine provided below. Save the file as arbiter.v.

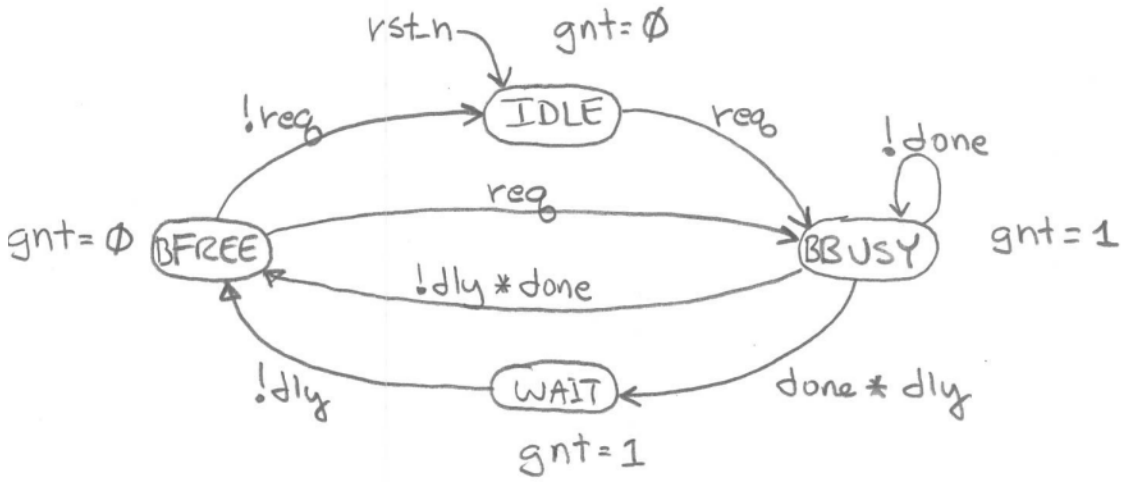

Figure 1: FSM before the enhancement

Here are some reminders:

- The signal *rst_n* is an active-low reset
- *!req* indicates a transition occurs when *req* is low.

- *done*dly* signifies that both *done* and *dly* are asserted (active high).

Additionally, enhance the above state machine by introducing a new 1-bit output register named *tout*, representing a timeout. Modify the logic to allow the machine to remain in the BBUSY state for a maximum of 2 clock cycles. Upon reaching the 3rd cycle, the machine transitions to a new state, TIMEOUT. In TIMEOUT, it sets the *tout* bit to 1 and deasserts *gnt*. The TIMEOUT state persists until *rst_n* is de-asserted. Then, the machine transitions to IDLE and de-asserts *tout*.

Finally, create a testbench file named tb_arbiter. The testbench should guide the new FSM through all states. Specifically, we aim to observe the FSM going through the following sequences:

- IDLE, BBUSY, WAIT, BFREE
- IDLE, BBUSY, TIMEOUT, IDLE
- IDLE, BBUSY, BFREE

1.2 Track 2 (574): Next Generation Assignment

Students enrolled in 574 are tasked to "*to lead, mentor, and otherwise contribute to the development of future engineers*". As discussed before, you will create a new assignment that will be added to the collection of assignments that future students may have to work on. In short: make this such that it is interesting and fun, with sufficient complexity – but do not make it too difficult either. For both variants of this assignment – "regular 574 assignment" and "portfolio assignment", you are kindly asked to do the following:

1. Create a task assignment in LaTeX, including figures, tables, etc. as needed
   - Preferably, use an open-source program to draw figures (or simply draw by hand)
   - Only use CircuiTikZ or WaveDrom for timing diagrams (or simply draw by hand)
2. Implement the module(s) and testbench as needed
   - Assume everything up to this point is known (including DSPs, BRAM, and FSMs)
   - Include a brief background section in your assignment or suitable references if domain-specific knowledge is needed (e.g., for an AI-related implementation problem)
   - Optional: include aspects of the synthesis process (what we will cover next week)
3. Prepare presentation slides
   - As a minimum a 5 minute lightning talk with the following slides: problem statement, approach, design, implementation, results (i.e., one minute per slide)
   - Submission of your slides as `.pdf` as all your slides will be concatenated for the lightning talk to minimize gaps between talks
4. If you want to do this as a portfolio assignment, please check with your instructor the specifics of your topic.

Please choose from the following list of topics. They are somewhat ordered based on my perceived level of difficulty (possibly the easiest topics first, most difficult topics last).

- Regular 574 assignment (given topic; TAs/instructor may help you):
  - Dongjun: Comparison of DSP-based combinational multiplier vs. sequential multiplier
    * Combinational multiplier: using a DSP (as demonstrated by Dongjun)
    * Sequential multiplier: see lecture slides
  - Gabriel: Protocol implementation (e.g., modified UART called "OSU-ART")
    * This will be an "FSM-heavy" assignment and should be fun
    * Possibly gamified such that an "attack" test-bench is written to enter illegal state?
    * Goal is to replace the bus arbiter for the current 474 students
  - Vincent: CRC32 checksum computation (similar to long division)
    * Includes computation of parity and verification
    * Based on Linear Feedback Shift Register (LFSR) and GF arithmetic
    * Additional materials can be provided to you
  - Vincent: Error-Correcting Code (Encoder/Decoder, e.g., BCH or RS)
    * Includes encoding and decoding
    * Decoding should be kept simple (e.g., $t = 1$) and as LUT
    * Additional materials can be provided to you
  - Vincent: Filter implementation (advanced)
    * Test vector generation using Matlab/Python
    * Possibly using `cocotb` and/or `fxpmath`
    * Filter module and test bench using test vector
- Portfolio assignment (your own topic; TAs/instructor cannot help you):
  - A topic of your choosing (please check with me, requires my approval)
  - Assignment document will be created under CC BY SA licensing conditions
  - You are the author of this document and can include it in your portfolio (public)
  - Solution *cannot* be made public; must be kept private
  - Examples include but not limited to: AI-related, image/video processing, etc.

Note, the due date for the 574-based assignment is June 2, such that we can have your presentations on June 5. We should be able to have up to 18 presentations with 5 minutes each. If we have less presentations, the remaining time will be used for Q&A.

2 Output

Submit all code files and a lab report including the enhanced FSM diagram and test documentation, and ideally, the wave form configuration in the archive.

3 Hints

- In many practical scenarios, your states will be internal and not in the top module
- Please check the translate on/off option of the Xilinx Vivado synthesis*
- This can help you to create signals in your port description for simulation purposes only
- For the actual implementation (synthesis) that we will do next week, these additional signals will be ignored. This approach enables more complete testbenches while also having the opportunity to automatically verify signals that would remain "hidden" in sub-modules otherwise.

* https://docs.amd.com/r/en-US/ug901-vivado-synthesis/TRANSLATE_OFF/TRANSLATE_ON-OFF/ON