Text, logo

Description automatically generated **San Francisco Bay University**

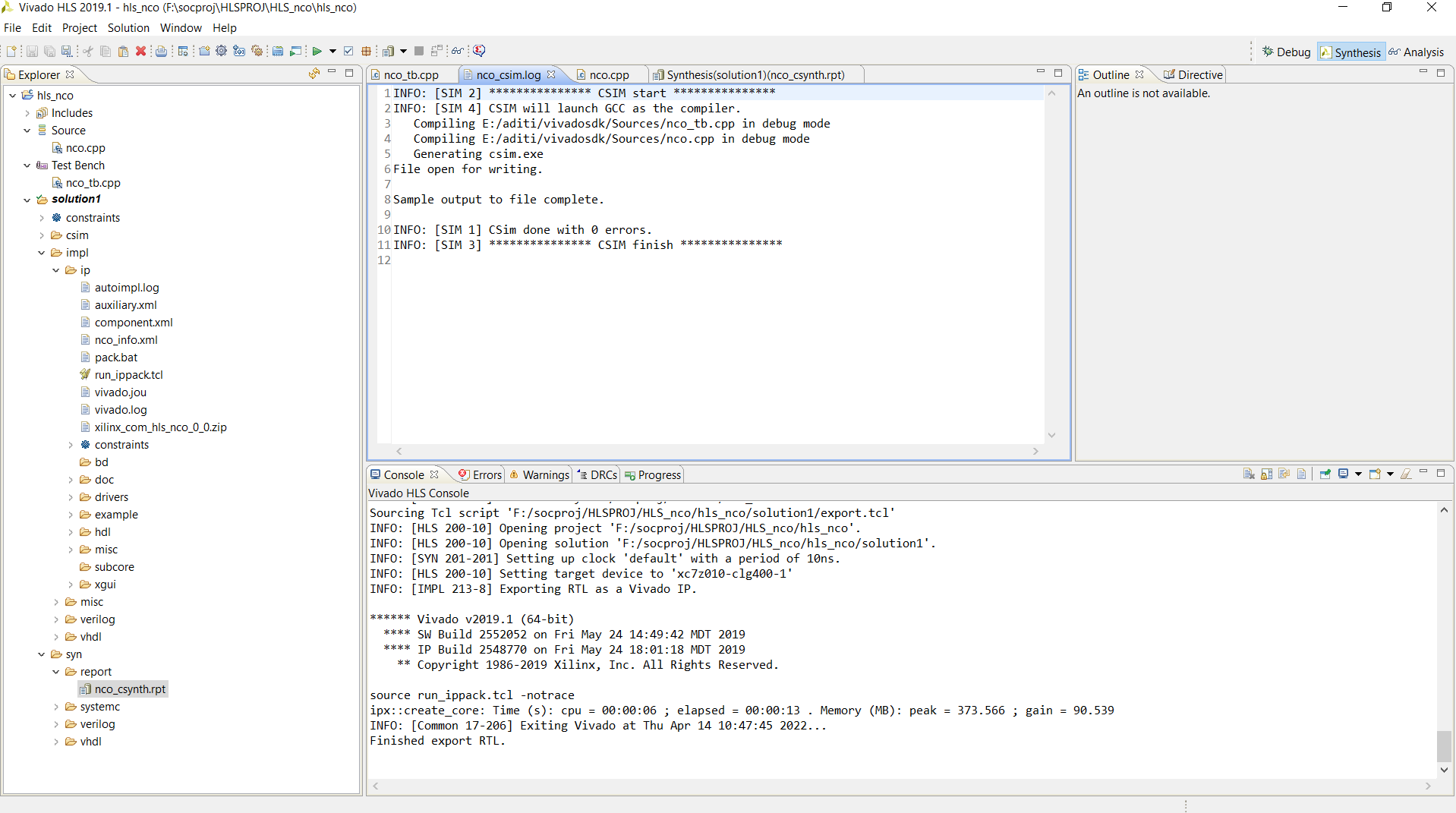
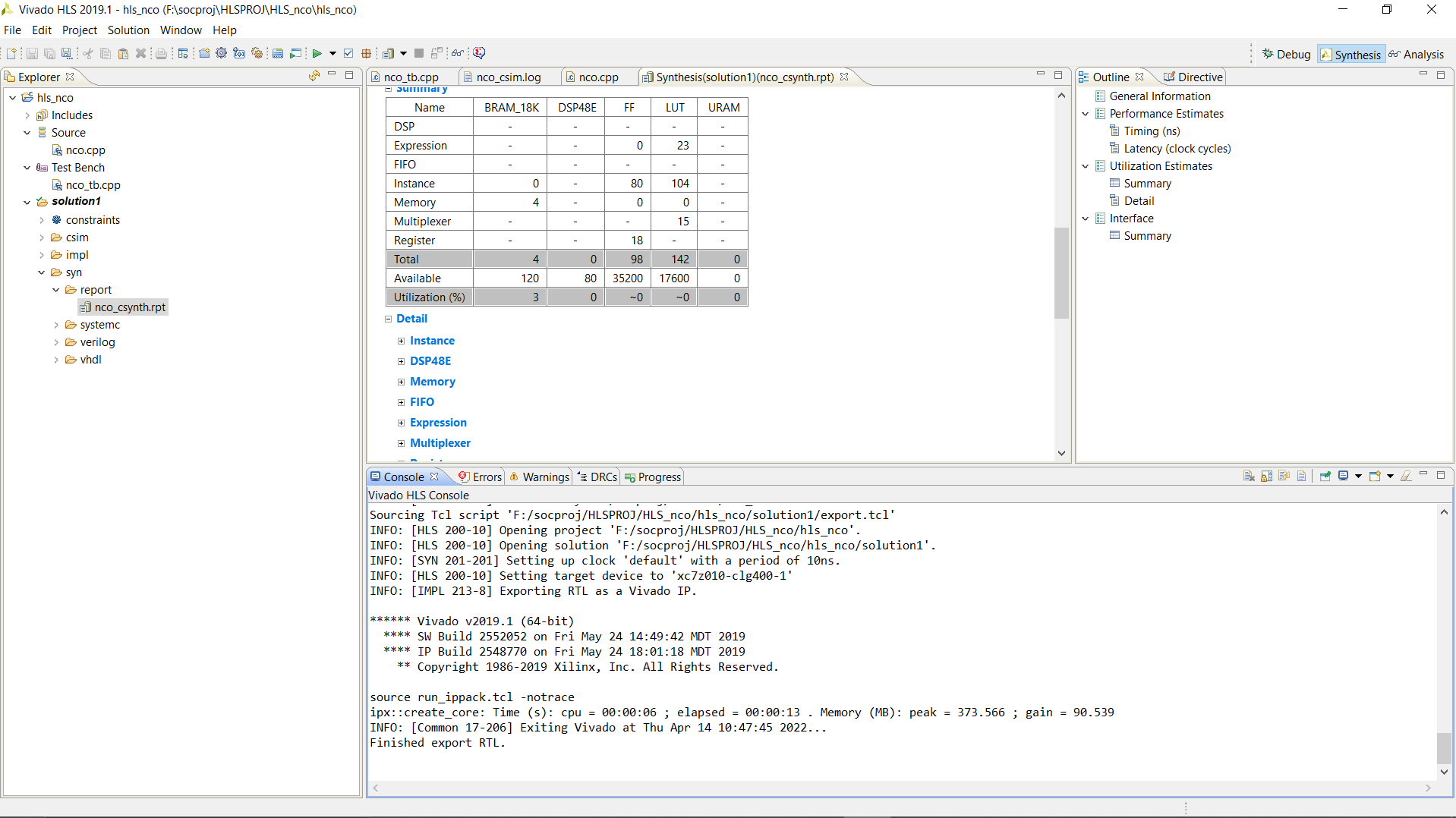
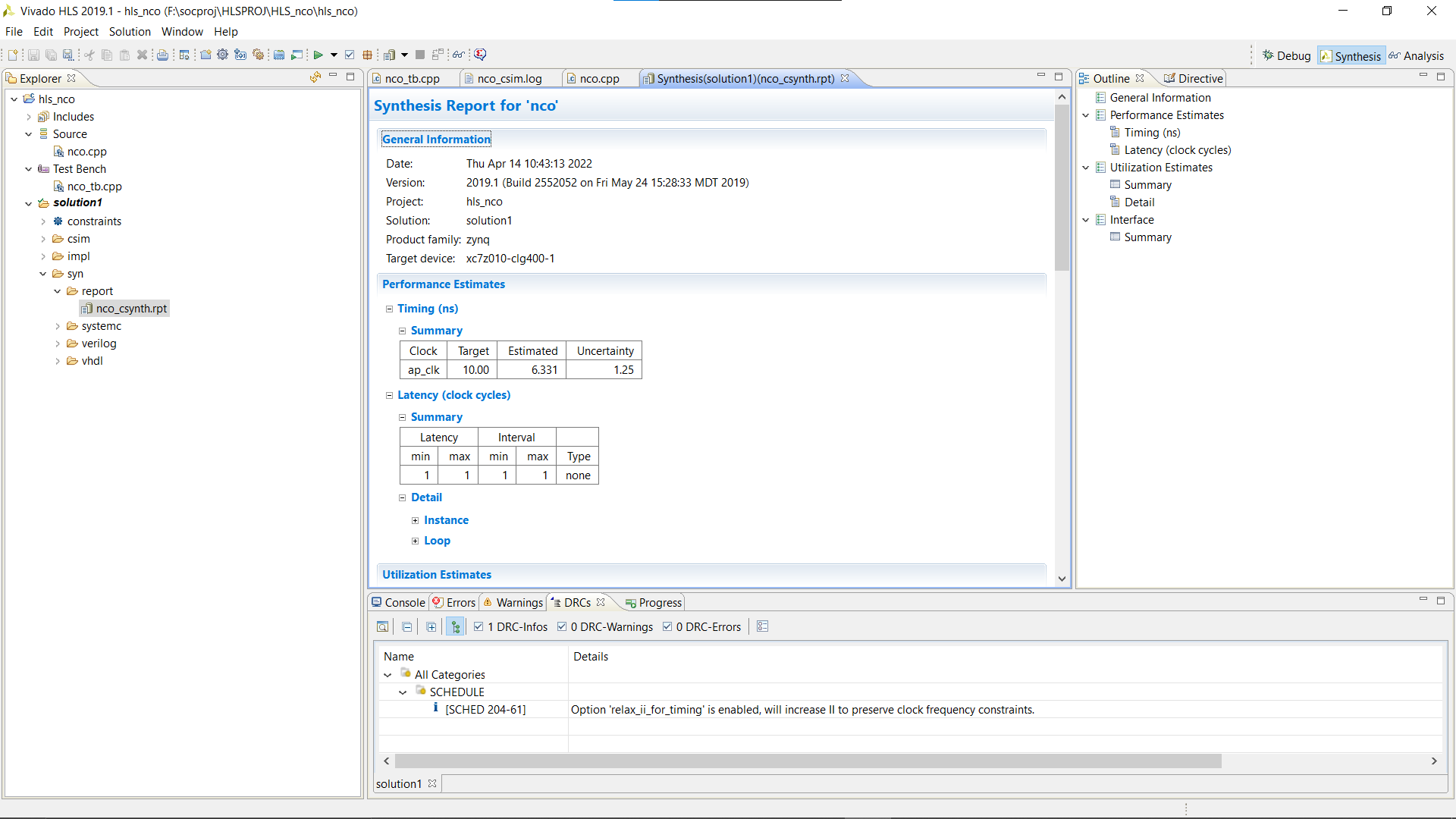
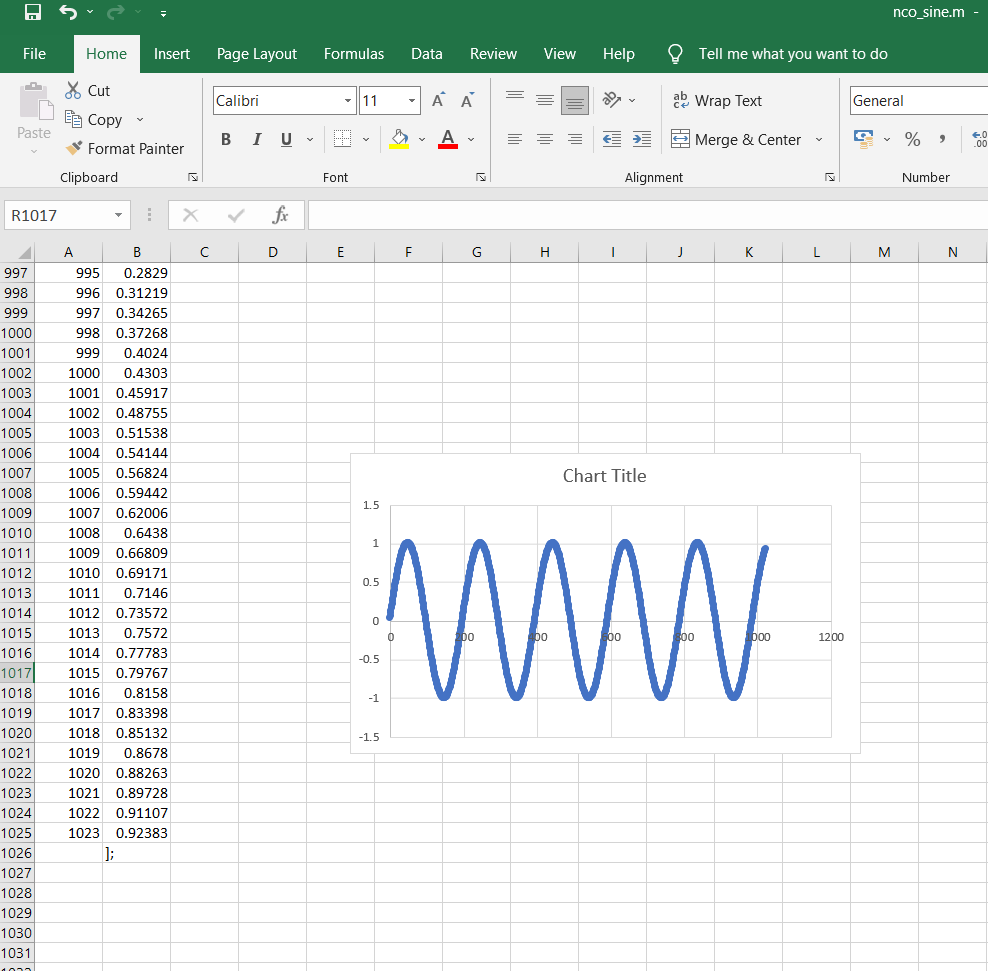
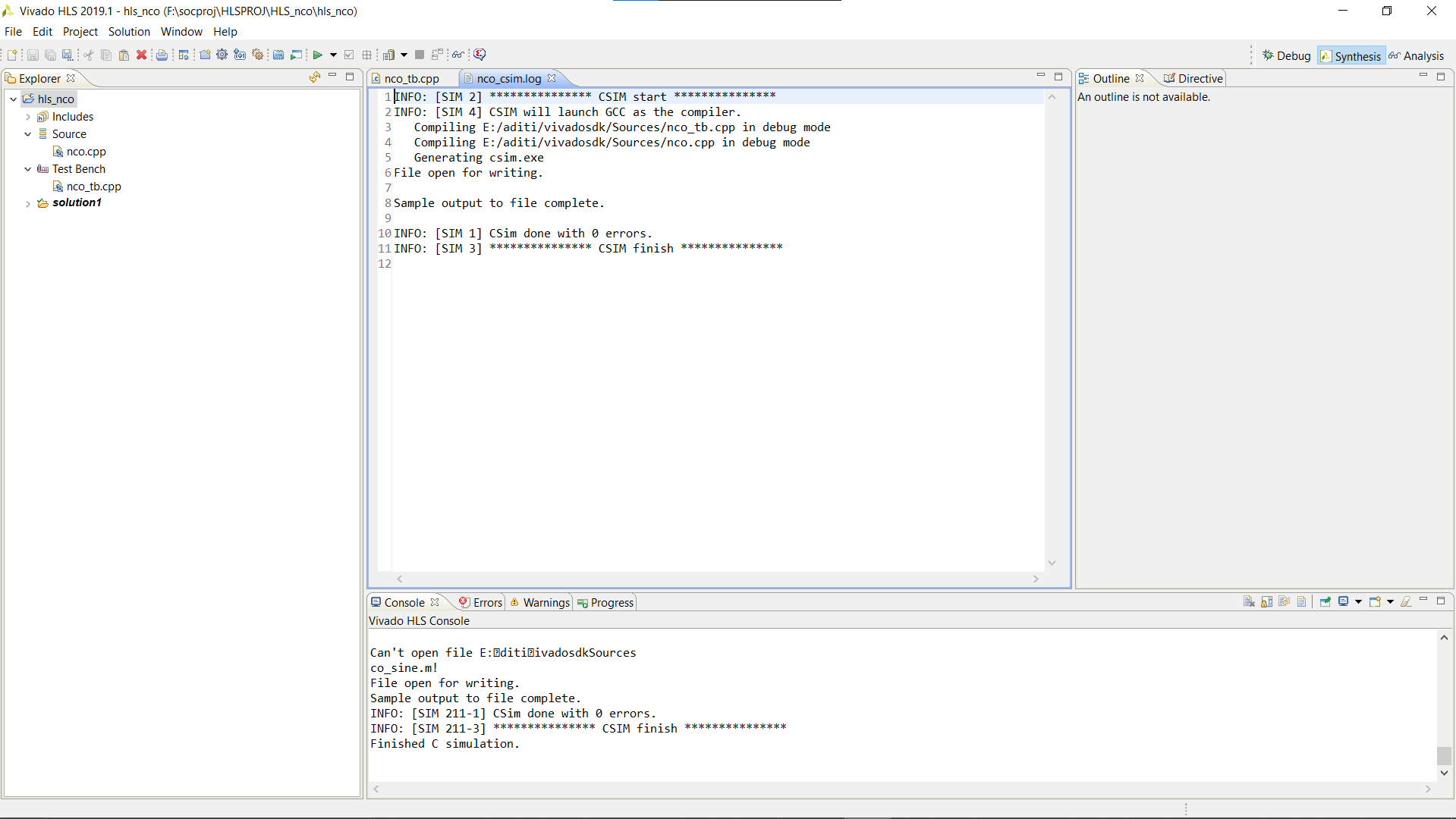
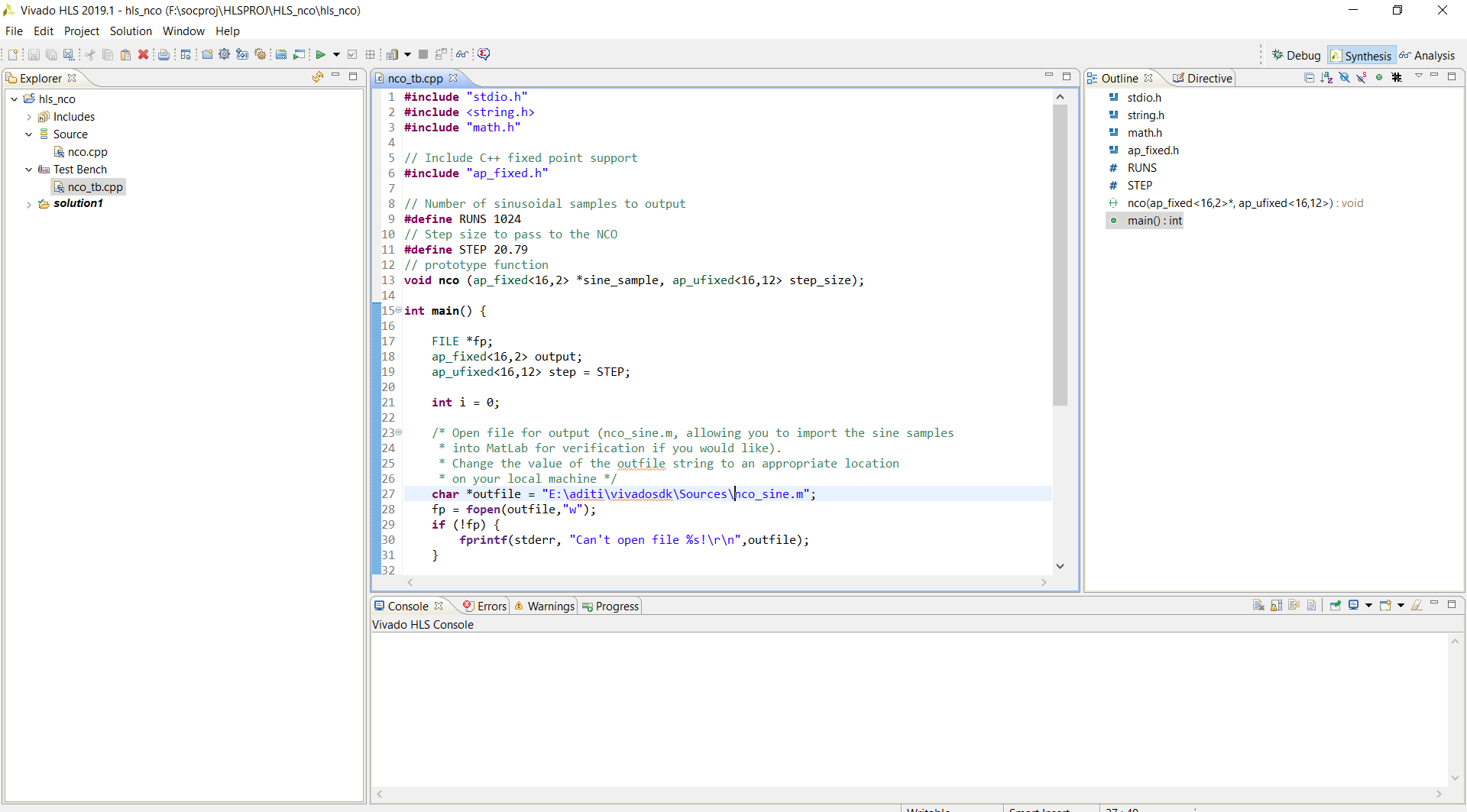
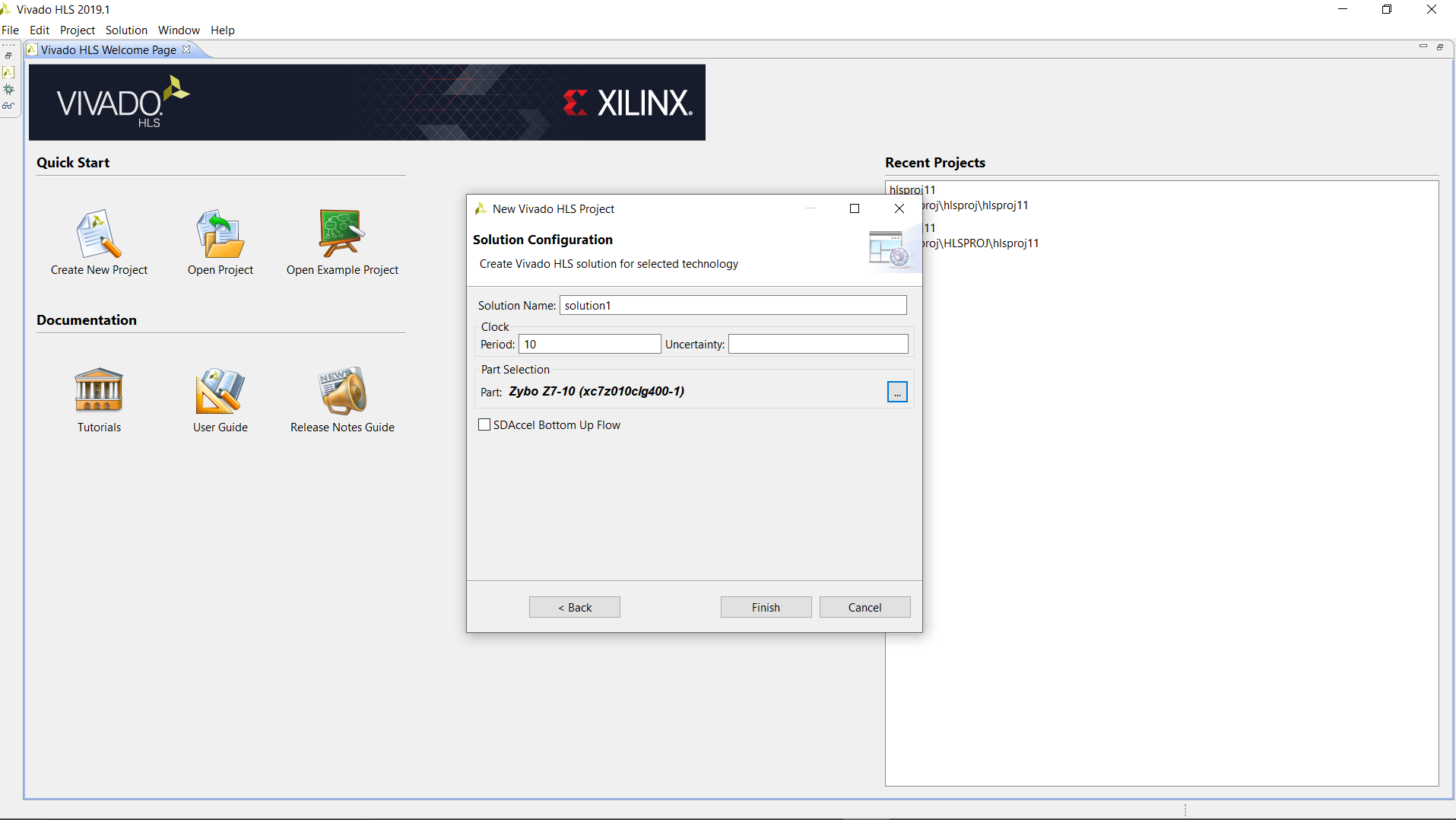
**EE553 - System on Chip (SoC) Design**

**Homework Assignment #6**

**Due day: 4/16/2022**

**Instruction:**

1. **Push the answer sheet to Github**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. Implement IP core generation in HLS following the instructions in Lab11 and submit the program running video recording



1. Answer the following questions regarding the logic synthesis in Verilog
   1. In combinational logic, the statements can contain combinational feedback loops. If it is not true, explain why
      * Combinatorial feedback loops are usually undesirable because the output will oscillate and the output is unpredictable. However, sometimes that's exactly what you want for example- A hardware random number generator (RNG). The most common approach for true random number generation on FPGAs is ring oscillator based.
   2. Can we assign "x" value to the signals? Why
      * Some designers assign X to signals to show that they are “don’t care” values, as a hint to the synthesis tool so it can assign either 1 or 0 to the signals during logic optimization. Although, simulations may exhibit indeterministic behaviour even when inputs to logic are X and have indeterminate value. It can be dangerous and can mask real RTL bugs.
   3. Use parameters for state-encoders in FSM (Finite State Machine)?
      * Yes, use of parameter (local parameter) can be made to design FSM, to define or assign states values.
   4. The design coding should generate tri-state logic. If it is not true, explain why
      * Tri state logic points to indeterministic behaviour with values X(unknown) or Z(high impedance), When these value are obtained, designer should have a logic to handle these situations.
   5. Tie the un-used bits to a known value, and why
      * We can set bits to be X in situations where we don't care or don’t use, the value. This can help catch bugs and improve synthesis quality.
   6. Minimize top-level glue logic coding style, and explain why
      * You can minimize this logic sometime by merging in inside another module.

It is recommended to avoid glue logic as much as possible, because it might cause problems and design complexity later in the chip design flow

* 1. Prefer using case statements for truth-table like structure, instead of

using long if-else statements, and explain why

* + - For just a few items, the difference is small. If you have many items you should definitely use a switch case. If a switch case contains more than five items, it's implemented using a lookup table or a hash list. This means that all items get the same access time, compared to a list, where the last item takes much more time to reach as it has to evaluate every previous condition first.
  1. Ensure that default statement is used in the case statements, and explain why
     + The default statement is executed if no case constant-expression value is equal to the value of expression. If there's no default statement, and no case match is found, none of the statements in the switch body get executed if the expression doesnot have any matched value.
  2. Ensure that the unused module inputs are driven and not floating, and explain why
     + When unused digital inputs are left unconnected then they will float, which can cause both unexpected logic behavior and excess current draw.
  3. Ensure that the design does not contain initial blocks and delay elements, and explain why
     + Delays declared in RTL code can never be synthesized. They are meant only for simulation purpose and modern synthesis tools will just ignore delays declarations in the code.
  4. Ensure that the whole design is resetable to a known state. No internal

logic generated asynchronous resets, and explain why

* + - For individual ASICs, the primary purpose of a reset is to force the ASIC design (either behavioral, RTL, or structural) into a known state for simulation. many data path communication ASICs are designed to synchronize to an input data stream, process the data, and then output it. If sync is ever lost, the ASIC goes through a routine to re-acquire sync. Also, a synchronous reset is recommended for these types of designs because it will filter the logic equation glitches between clocks. By using synchronous resets and a pre-determined number of clocks as part of the reset process, flip-flops can be used within the reset buffer tree to help the timing of the buffer tree keep within a clock period.
  1. Ensure that the resets are not used as data or clock signals in the design, and explain why
     + No reset glitches, resets only occur at an active clock edge. Any Reset signal which is generated from some logic in the design will be synchronized and the logic glitches are filtered.
  2. Never use both "posedge clock" and "negedge clock" to trigger in one

system, and explain why

* + - Mostly all standard flip-flops have not more than two inputs with single edge events. So, the solution must use standard flip-flops with additional combinational circuits. “always” blocks with same trigger executes concurrently. Hence, using of both edges creates variation resulting into loss or indeterministic result in control over time.