



UNIVERSITY OF TEHRAN
Electrical and Computer Engineering Department
Object-Oriented Modeling of Electronic Circuits, Spring 1400
Computer Assignment 5, Week 14-16

System Modeling with CPU and Analog

Name:

Date:

The system you are going to describe, is a very simple model of an automotive sensor node. Sensor nodes integrate sensors, actuators, computing elements, e.g., microcontrollers, memory, and communication systems.

Figure 1 shows the overall picture of what you are going to implement in this homework. Two different sensors are used in this sensor node. The microcontroller reads the data from both sensors every other 1 millisecond and check if they sit within the specified range for correct performance. One example of this sensor node's application is in Electronic Stability Control systems that the microcontroller compares speed sensors' data and controls brake cylinder pressure. To provide the correct 1 ms interval of data acquisition a timer is needed to count the time and let the processor know for new reading time.

In the first part of this homework, you will design the sensor packages and the timer module and in the second part you will integrate this system to a processor and the corresponding bus interface.

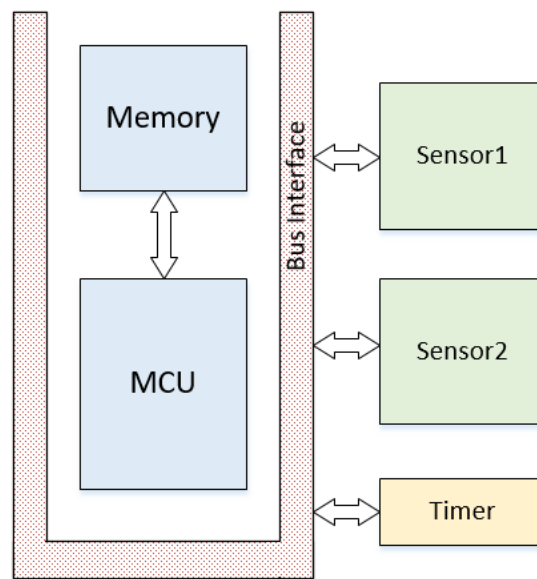


Figure 1: Overall system

Part 1:

Figure 2 shows the sensor packages. Each package includes an input source module, a sensor frontend, and a register to store the digital data.

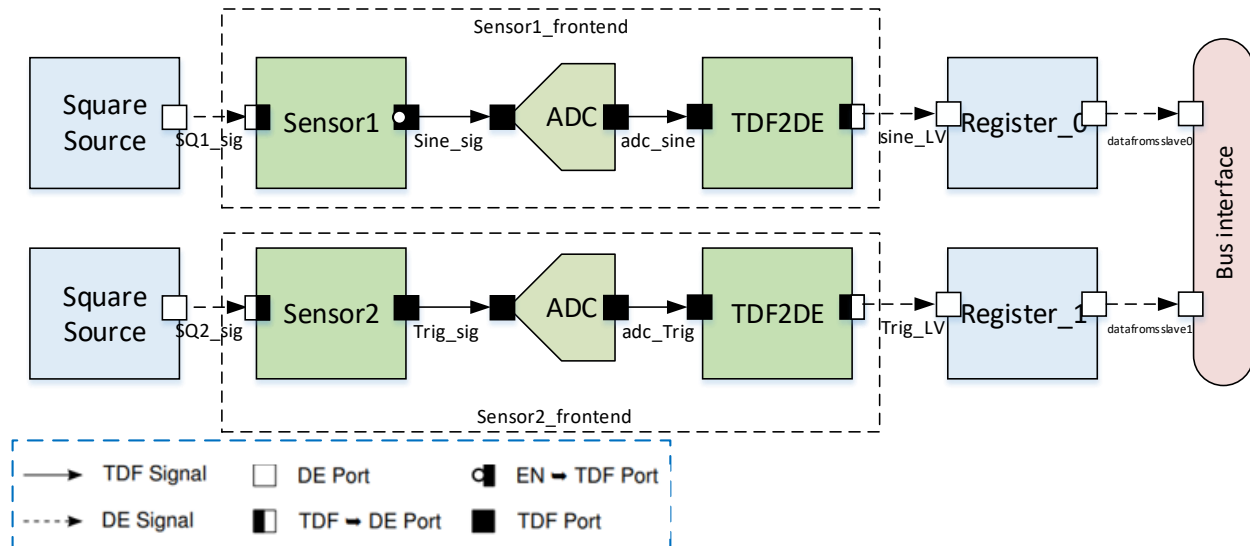


Figure 2: Sensor packages

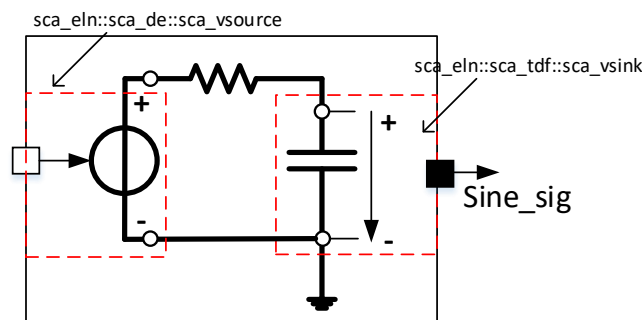
A. Sensor 1 structures and interfaces

a. Square Wave Source:

The input source module is a square wave generator with a specified time period. This module represents the change in environmental parameters like speed, distance, or pressure. Design this module as explained in the lectures. The output of this module is a discrete event signal changing between zero to one with the frequency of 100 kHz.

b. Sensor1 Frontend:

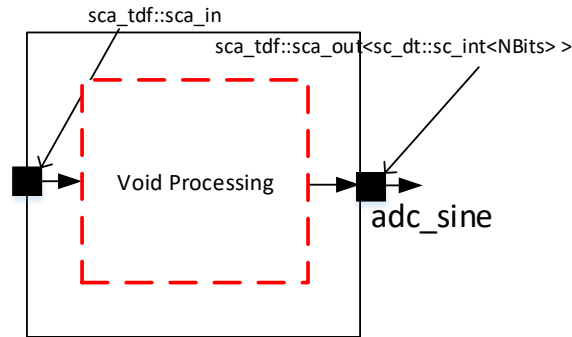
- Sensor 1:** This sensor is a lowpass filter that generates a sine wave from its input square wave. As shown in Figure 2, this filter receives a DE input, and the output is a TDF signal. Use the necessary input and output data types. For the output you need an eln:tdf voltage sink.



- c. **ADC:** Analog to Digital Converter converts the TDF signal into an N-bit digital output. For this purpose, you need to write a TDF module that implements the following equation:

$$\begin{cases} V_{max} & V_{in} > V_{max} \\ \text{round}\left(\frac{V_{in}}{V_{max}} \times (2^{N-1} - 1)\right) & -V_{max} < V_{in} < V_{max} \\ -V_{max} & V_{in} < -V_{max} \end{cases}$$

In this homework, N=16.



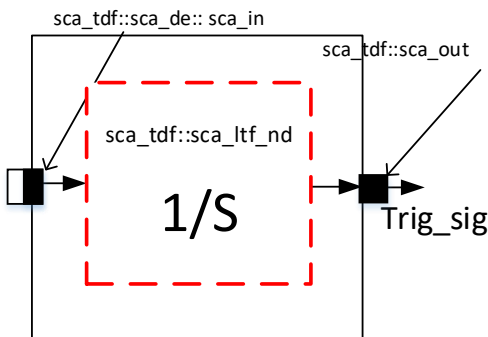
- d. **TDF2DE:** The output of the ADC is of type $\{sc_dt::sc_int\}$. Since this data will be used in a complete digital part (de) that starts with a register, passes through the bus interface, and is finally processed in the processor, it should be converted to an $sc_lv <16>$ data. Write a TDF module that receives a tdf $sc_dt::sc_int$ signal and turns it into an $sc_lv <16>$ output.
- e. **Register:** Before sending data to the processor, the sensor clocks the generated digital into its buffer register. the data with a specified clock. This clock should be at least twice as fast as the square wave frequency or faster. Both the input and output of this module are of $sc_lv <16>$ data type. Use an internal clock of 1 MHz for registering sine wave into the buffer register. Clocking data into the register happens at all times. Use a tristate structure at the output of the register that becomes active with an *outEnable* signal. This signal will be controlled by the processor when the registered sensor output is to be placed on the databus.

B. Sensor 2 structures and interfaces

The same procedure is almost true for the second sensor package with the following differences:

- Square wave Source:** The second square source module works with the frequency of 200 kHz.
- The internal clock of the second sensor is 2 MHz.
- Sensor2 Frontend:**

- **Sensor 2:** This sensor is an integrator circuit implemented with a Laplace equation as explained in the lectures. It converts the square wave to a triangular wave. As shown in Figure 2, this filter receives a DE input, and the output is a TDF signal. Use the appropriate input and output data types.



d. ADC, TDF2DE and Register: These modules are the same as those of the first sensor package.

C. Timer: The timer module is used for synchronizing the times that the processor reads data from the sensors. The processor reads the data of the sensors every millisecond. The timer is responsible for signaling to the processor that the 1ms time interval has arrived and data should be read from the buffer registers of the sensors. Write an SC_MODULE with an input for starting the timer and one for issuing the timeout. The input is *startTimer* and requires a synchronous pulse from the processor to start the timer. This signal can be generated by the processor simply by addressing the appropriate location for a write. When the timer starts, its *timeOut* flag becomes 0 and after its programmed time interval (i.e., 1 ms) it asserts the *timeOut* flag. After issuing the *startTimer* and after the clock edge, the processor continuously reads the *timeOut* flag waiting for the elapse of 1 ms time interval. You can use the same IO address for *startTimer* and *timeOut*, one using write and the other using read of the same location. Note that the timer *timeOut* flag must have a tristate structure in order to be read by the processor on the databus.

Part 2:

In this part you are going to integrate the modules of part 1 to the bus interface and processor.

A. Bus Interface: Write an SC_MODULE that uses the memory maps below for decoding the peripherals.

- **Sensor1:** address_bus [15:0] = Fx10 to Fx13
- **Sensor2:** address_bus [15:0] = Fx14 to Fx17
- **Timer:** address_bus [15:0] = Fx18 to Fx1B

Based on the address on the address bus one of these peripherals will be activated and based on the *writeIO* or *readIO* signals, the processor writes or reads the appropriate source. Use any and as many addresses as you need from the above. The bus interface puts the data from the peripherals on the databus of the processor when *writeIO* is 1 and puts the databus on the peripheral inputs when *readIO* =1.

B. Processor SystemC Bracketing. For the processor use the SystemC Bracketing explained in the class. The processor does the followings:

- First it puts the address of timer on the address bus and issues *write_IO*. It can configure the timer with the time for waiting. If you want, you can dedicate an address location for this.
- The timer starts waiting and the processor checks the flag *timeOut* repeatedly.
- When the *timeOut* flag becomes 1, the processor reads Sensor 1 by placing its address on the address bus while enabling the *readIO*.
- The processor stores this value in *sensor1Data* variable.
- Next, the processor reads Sensor 2 by placing its address on the address bus while enabling the *readIO*.
- The processor stores this value in *sensor2Data* variable.
- The processor compares these two values with two thresholds, *sensor1Th* and *sensor2Th*, and reports the differences.
- For displaying the results, cout if the sensors Pass or Fail. You can cout other information as you see helpful.
 - Pass if sensor data is greater than the sensor threshold.
 - Fail if sensor data is less than or equal to the sensor threshold.

C. Processor ISS. Repeat the same thing you did with the bracketing, now using SAYAC instructions. All peripheral readings and writings should be done by SAYAC instructions. Obviously, you will not be able to use cout and you are limited by using SAYAC instructions. For outputting the results, choose a port to write to similar to what was done in the display module discussed in class.

Historical note: In the 8-bit processor era, Intel provided 8253 for a timer, 8255 for peripheral interfaces, 8251 for UART, and 8259 for interrupt handling, just to name a few. In this assignment, we are doing what 8253 and 8255 were designed to do.