Embedded Control Systems 5LIJ0 Assignment 2

Data-intensive Control

March, 2023

1. System description

We consider a Wafer-handling machine (Fig. 1), which is used for semiconductor assembly, die-bonding, and pick and place tasks. A Wafer is composed of several dies placed horizontally and vertically (Fig. 2).

Fig. 1: Die-bonding machine [www.itecequipment.com].

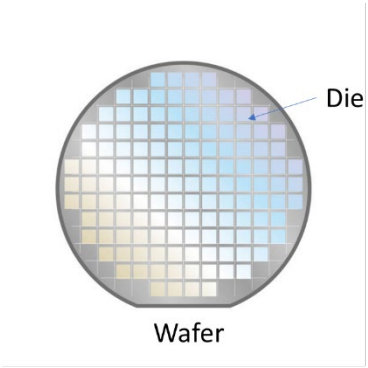

Fig. 2: Wafer and dies.

A die-bonding machine has a Wafer table and the wafers are placed on the table. The wafer table is a conveyer belt that linearly moves using a linear motor taking the Wafer from one location to another. When the Wafer comes below the vacuum nozzle, the dies (in Wafer) are picked up by the vacuum nozzle for further procedure such as die-bonding or packaging. As shown in the Fig. 3, the camera is mounted on top focusing on the Wafer containing an array of dies placed horizontally and vertically. The camera and an image processing algorithm are used to measure the position of the Wafer (or dies) and is used as feedback in the linear motor. The linear motor moves the Wafer table such that a die is placed precisely below the vacuum nozzle. This allows for further processing of the dies after they are picked up by the vacuum nozzle.

For simplicity, we assume that the camera focuses on the first die in this assignment (Fig. 4 (a)). We focus on the horizontal motion (of the dies) along x-axis. Fig. 4 (b) shows the top view of the die on the Wafer table with the current (first) die position and the reference position where the die needs to be positioned. A camera is used to measure the current position of the die. The control problem is to center the die to the

target position $r = 0.044m$ at the end of the iteration. The image resolution is 88×142 pixels. In the image plane, the reference of $0.044m$ translates to 44 pixel, i.e., the middle of the image.

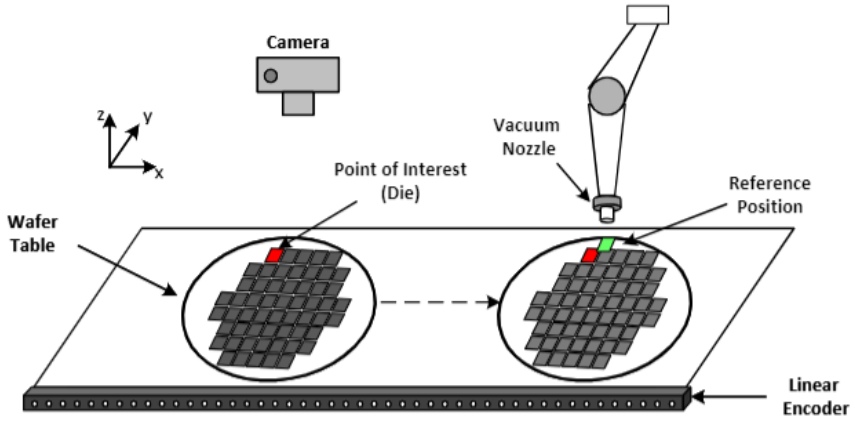

Fig. 3: Schematic of the Wafer table in a die-bonding machine.

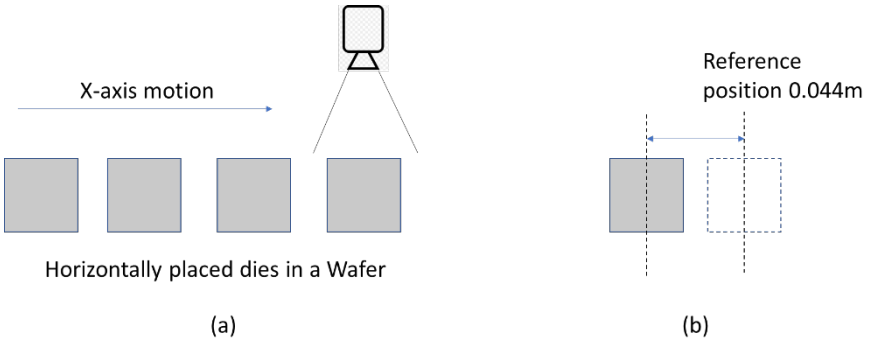

Fig. 4: (a) Wafer table motion along x-axis (b) Camera top-view and reference position.

2. Software-in-the-loop (SIL) setup

The Wafer table and the camera are modelled in Coppeliasim (https://www.coppeliarobotics.com/) which is a free Physics simulation engine. See Fig. 5.

Plant/system model: We model the x-axis movement of a die on the Wafer table. As a control action, $u[k]$ applies force to the Wafer table to realize the motion along x-axis.

Vision sensor: Fig. 5 (a) shows the simplified schematic of a die-bonding machine with the camera and the linear motor. Fig. 5 (b) shows the image captured by the camera. The image processing algorithm uses the captured image to extract the position information. To model the camera, a CoppeliaSim vision sensor is used. We consider an image resolution of 142 × 88 pixels. One pixel is calibrated to dimension of 1 mm. A die has a dimension of 25 × 25 pixels (or 25mm × 25mm). The camera focuses on the first die. Initial position of the die is the left-most boundary of the image. The reference position r is 0.044m which 44 pixels, i.e., the center of the image.

Camera frame rate is configurable and measured by frames per second. For example, a frame rate of 100fps implies to generate one image in every 10ms. This is referred to as frame period f_h.

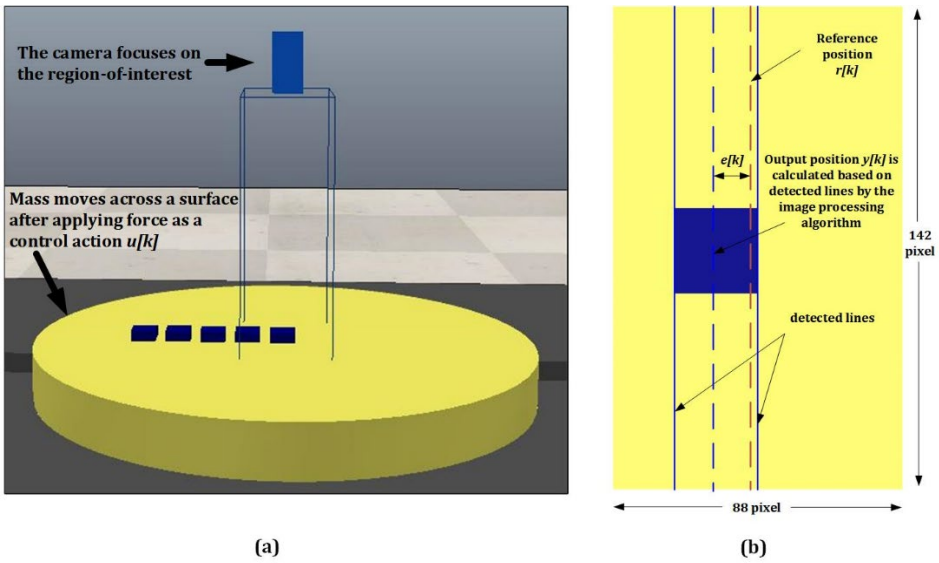

Fig. 5: (a) Wafer table and camera modeled in CoppeliaSim (b) Camera top view and reference position.

Vision algorithm: We use Hough transform algorithm to obtain the location of the die from the image. The algorithm extracts the line segments and returns the x-coordinate of the center point of a product. The ratio between the center point and full image width determines the absolute position of the center point to the axis in the simulation setup. Fig. 6 shows the algorithm steps. The input RGB image is converted to a grayscale image. Subsequently, we apply the Canny edge detection algorithm to detect the edges on the grayscale image. Once we get the binary image from edge detection, we pass it to the Hough transform algorithm which extracts the line segments on the image, which is used to compute the x-coordinate of the center point of a die.

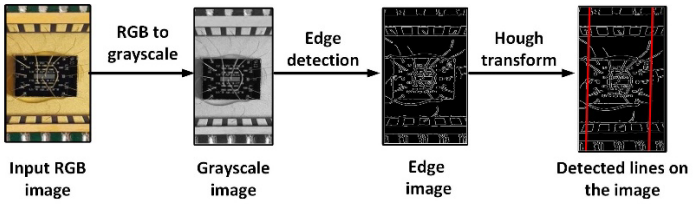

Fig. 6: Vision algorithm steps.

PID controller: The discrete-time PID controller equation is given as follows –

$$u[k] = k_p e[k] + k_i \sum_{k=1}^{N} e[k] f_h + k_d \left(\frac{e[k] - e[k-1]}{f_h} \right)$$

where $u[k]$ is the feedback control action (force), k_p, k_i, and k_d are the proportional, integral, and derivative controller gains respectively, f_h is the frame period or sampling period, e_k is the error signal,

$$e[k] = x[k] - r$$

where, $x[k]$ is the actual position (obtained by the vision algorithm) of the die at k^{th} sampling instance. The controller sampling period is exactly identical to the camera frame period.

Overall closed-loop system is shown in Fig. 7.

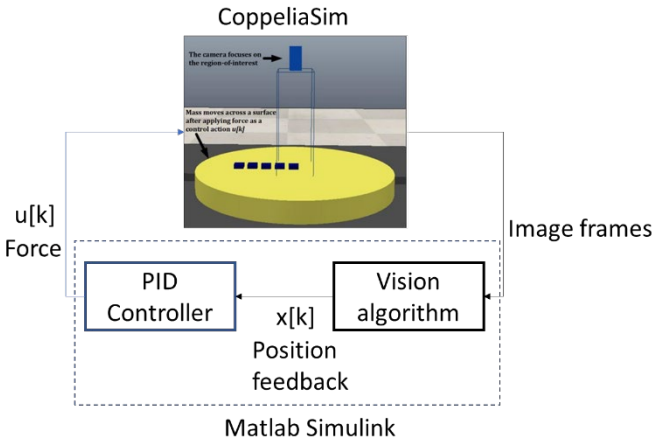

Fig. 7: Software-in-the-loop setup.

3. Multi-core platform model – CompSOC

The embedded platform targeted in this assignment is composable and predictable multi-core platform called CompSOC. Fig. 8 shows the hardware architecture of CompSOC platform. The system clock frequency is 40MHz. The CompSOC platform has three processor tiles running at 40MHz. The tiles are named as Tile 0, Tile 1 and Tile 2. These tiles can communicate through shared memories. For example, Tile 0 and 1 have a shared memory of Mem01, Tile 1 and 2 have a shared memory of Mem12, and Tile 0 and 2 have a shared memory of Mem02.

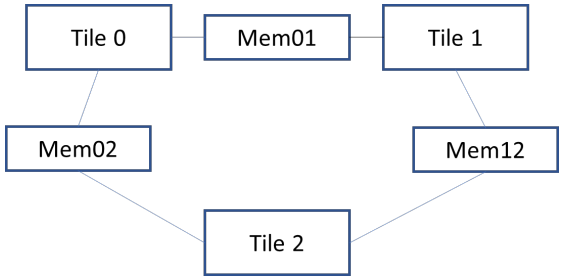

Fig. 8 : CompSOC hardware architecture – 3 tiles and 3 shared memories between the tiles.

A periodic time-division-multiplexing (TDM) policy is used on all processor tiles. All the tiles are synchronized. CompSOC uses a predictable and composable micro-kernel (CoMiK) to create a virtual execution platform (VEP) or a partition slot. The length of a CoMik slot is fixed and 2000 cycles. There can be maximum 4 partition slots in each tile. The partition slots are named as Par_x_y for the yth partition slot in tile x. For example, Par_0_1 refers to the second partition slot of Tile 0. The Par_x_0 (or the first slot of each tile) is of length 5000 cycles, allocated to the system application and this partition slot cannot be used for application development. The Par_0_1 is used for monitoring app (to be described later) and cannot be used by other application. Fig. 9 shows the overall platform architecture. In each tile, the user may choose to use 2 or 3 partition slots. Of course, the user may choose not to create/use any partition slot (which can be done by modifying vep_config.txt).

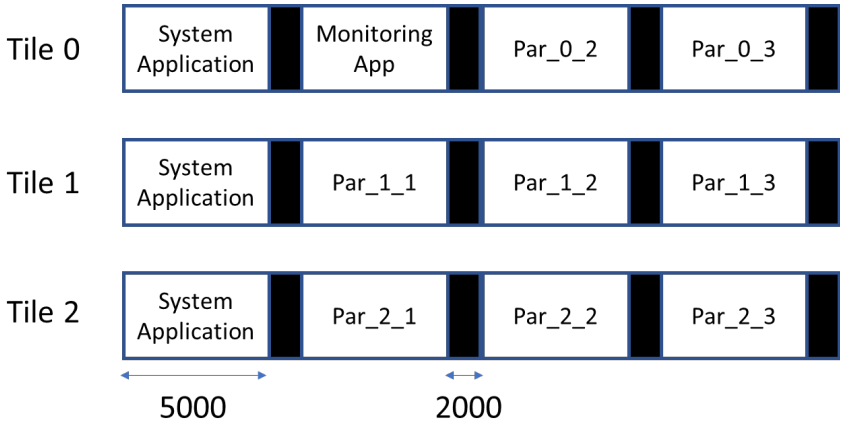

Fig. 9 : CompSOC platform model.

Memory constraints: Each tile may use maximum 96KB data memory which is further divided between the partition slots. A partition slot (to be used for application development) may either have 32KB or 64KB data memory. Therefore, if one partition slot is allocated 64KB, only one more partition slot can be used with 32KB data memory. If three partitions slots are used, 32KB data memory should be allocated to each partition slot.

Monitoring App: Monitoring App reads the image frames from the CoppeliaSim and stores in the shared memory for further processing. It further stores the control input/force (computed by the controller) in the shared memory to be applied to the Wafer table in the CoppeliaSim environment for x-axis motion.

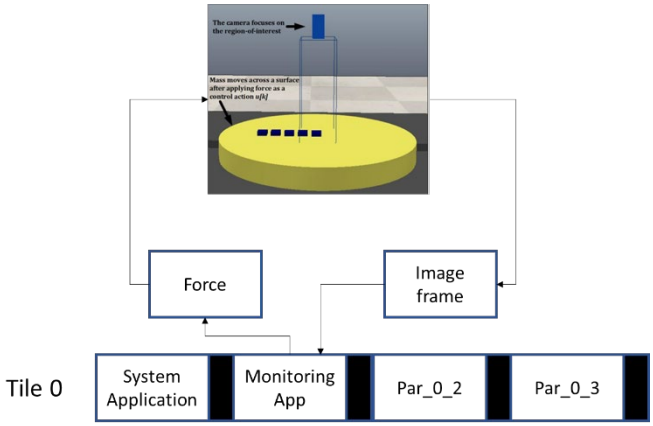

Fig. 10 : Monitoring App and interfacing with CoppeliaSim using shared memory.

4. Application model

The application is composed of sensing and controller tasks. The application model has to be considered for designing the schedule on the CompSOC platform.

Sensing task is responsible for obtaining the location of the die from the image captured in the CoppeliaSim environment. It is done in two stages – Canny edge detection and Hough transform.

Canny edge detection: Canny edge detection takes RGB image as an input and provides a binary edge matrix of the size of the image. That is, for an image of size 88×142, the Canny edge detection algorithm gives a matrix of 88×142 where 0 implies no edge and 1 implies edge. For reference, you have a look at the following reference: https://indiantechwarrior.com/canny-edge-detection-for-image-processing/. Under

the scope of this assignment, the execution time of Canny edge detection algorithm is executed in Matlab, is not modelled and the sensing task execution is assumed to be contributed by only the Hough transform algorithm.

Hough transform: Hough transform takes the binary edge image and gives the location of the center of the die. See reference https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549. The Hough transform searches to fit the line over a given range of angle denoted as θ. We consider the range to be -5^0 to 5^0. The Hough transform algorithm is compute-heavy and memory intensive. For the range and the image size, minimum 54Kb memory is required on the CompSOC platform to execute Hough transform algorithm. Furthermore, the computation of Hough transform can be parallelized over the range of θ. For example, the computation can be split into two tasks – one for the range -5^0 to 0 and 1 to 5^0 which can be performed independently. Internally, the Hough transform algorithm computes 4 data-peaks and the merging task uses those data peaks to obtain the location of the center of the die. The model of two different deployments are shown in Fig. 11.

Controller: Controller uses the measured location of the center of the die and compute the error with respect to the reference r = 44 pixels. PID control algorithm described previously is used for computing for the force to be applied to the Wafer table using the error signal. Overall application model is shown in Fig. 11 for the two cases.

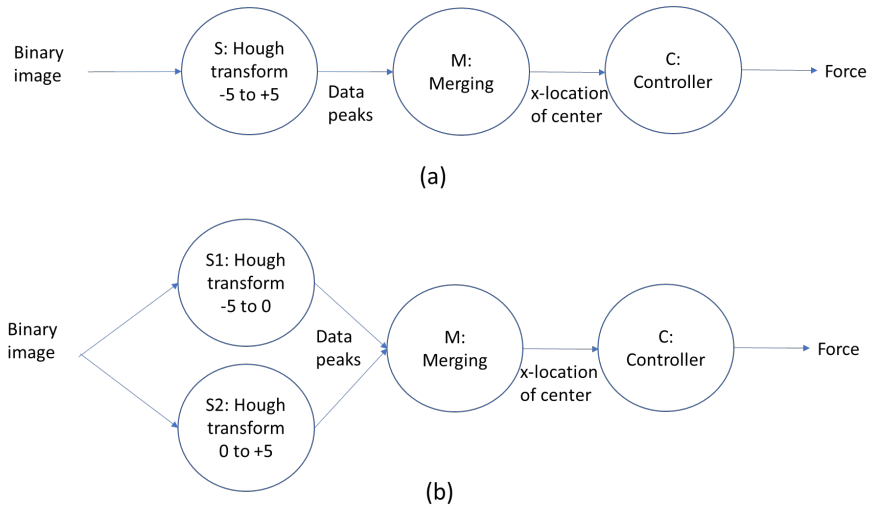

Fig. 11 : Sensing task (Hough transform) model (a) sequential (b) parallel.

5. Design problem

5.1 Sequential vision-in-the-loop control (10 points): Consider the sequential application model shown in Fig. 11(a). Please consider the platform model shown in Fig. 12.

(a) Measure and report the execution time of each computation block in Fig. 11(a).

(b) Considering the execution time measured in part (a), design schedule for the computation blocks for the platform shown in Fig. 12. Please report the resulting schedule diagram with the chosen lengths in cycles and the memory allocated to each partitions.

(c) Compute the sampling period for the application model under consideration (i.e., without considering execution time for Canny edge detection and actuation) for the schedule designed in part (b).

(d) Tune the controller gains $k_p, k_d, and\ k_i$ ensuring the system stability. Report the settling time and observed system behavior.

(e) Report the output plot for 5 seconds.

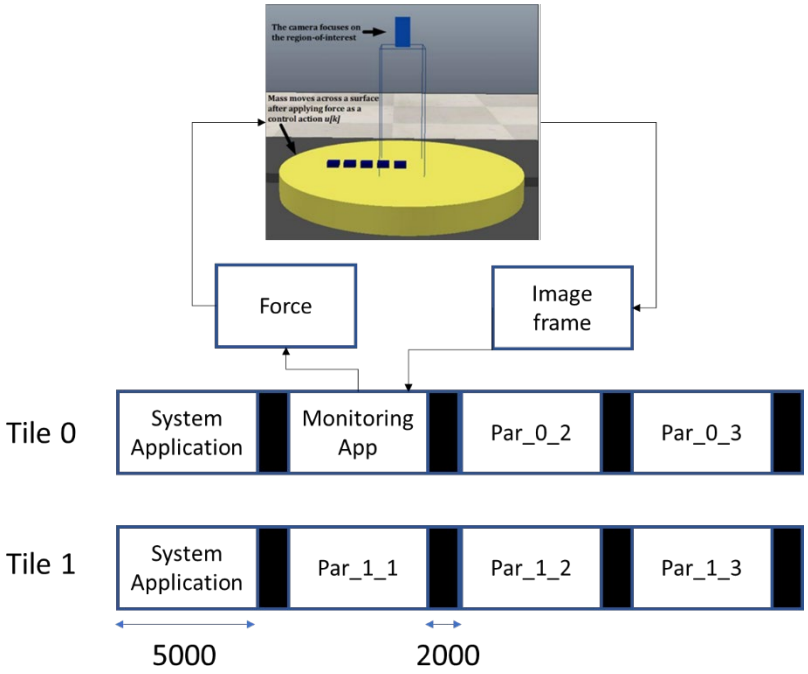

Fig. 12 : Sequential implementation platform model

5.2 Parallelized vision-in-the-loop control (15 points): Consider the parallel application model shown in Fig. 11(b). Please consider the platform model shown in Fig. 13.

(a) Measure and report the execution time of each computation block in Fig. 11(b).

(b) Considering the execution time measured in part (a), design schedule for the computation blocks for the platform shown in Fig. 13. Please report the resulting schedule diagram with the chosen lengths in cycles and the memory allocated to each partitions.

(c) Compute the sampling period for the application model under consideration (i.e., without considering execution time for Canny edge detection and actuation) for the schedule designed in part (b).

(d) Tune the controller gains $k_p, k_d, and\ k_i$ ensuring the system stability. Report the settling time and observed system behavior.

(e) Report the output plot for 5 seconds.

5.3 Pipelined vision-in-the-loop control (10 points): Consider the sequential application model shown in Fig. 11(a). Please consider the platform model shown in Fig. 13.

(a) Design schedule for the computation blocks for the platform shown in Fig. 13 for a pipelined implementation with two pipes. Please report the resulting schedule diagram with the chosen lengths in cycles.

(b) Compute the sampling period for the application model under consideration (i.e., without considering execution time for Canny edge detection and actuation) for the schedule designed in part (a).

(c) Tune the controller gains $k_p, k_d, and\ k_i$ ensuring the system stability. Report the settling time and observed system behavior.

(d) Report the output plot for 5 seconds.

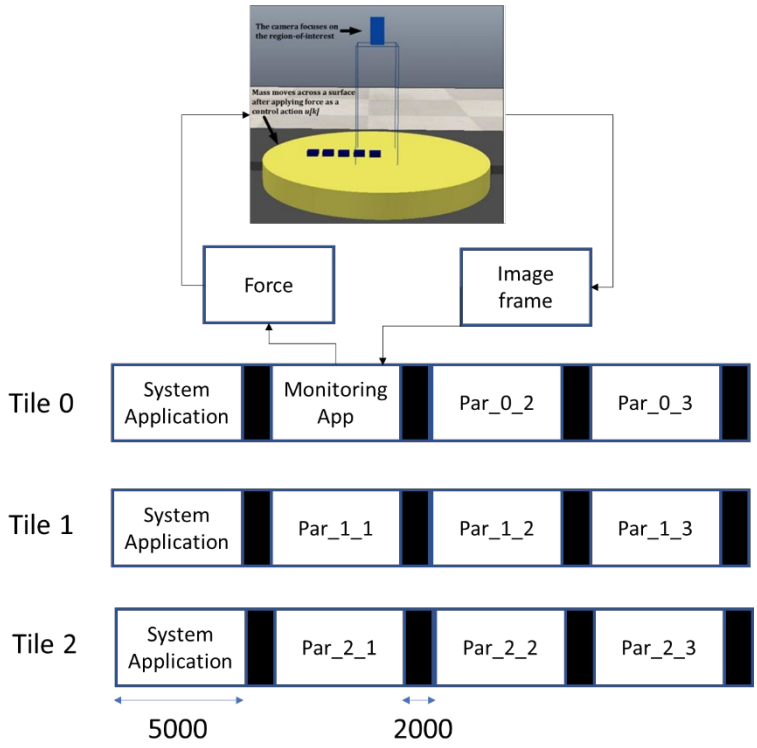

Fig. 13 : Paralellel and pipelined implementation platform model

6. Deliverables

- Report with the answers to the design questions 5.1 (a)-(e), 5.2 (a)-(e) and 5.3 (a)-(d) with **Assignment2_group_no.pdf.**
- Sequential in 5.1 – archive of the two folders including all the files with the following names

  (a) SIL_Sequential_Implementation_group_no.zip (b) PIL_sequential_impl_group_no.zip
- Parallel in 5.2 – archive of the two folders including all the files with the following names

  (a) SIL_Parallel_Implementation_group_no.zip (b) PIL_parallel_impl_group_no.zip
- Pipelined in 5.3 – archive of the folder including all the files with the following name

(a) PIL_pipeline_impl_group_no.zip

7. Provided materials

- **The_virtual_machine_tutorial.pdf –** follow the instructions to install the virtual machine.
- **PYNQ_board_tutorial.pdf** – to setup the PYNQ board
- **Assignment_2_SIL_Tutorial.pdf** – follow the instructions to perform SIL simulation and execution time profiling
- **Assignment_2_PIL_Tutorial.pdf** – follow the instructions to perform PIL simulation
- **Synchronous_simulation_VIL_systems.pdf** – explains the concepts of SIL and PIL simulation with CompSOC