**UNIVERSITY OF CALIFORNIA**
**College of Engineering**
**Department of Electrical Engineering and Computer Sciences**
*Last modified on Nov. 8th, 2010.*

*Elad Alon* **FALL 2010 PROJECT PHASE III** *EECS 141*

*Check-point due Sun. Nov 18th by email*
*Poster presentation due Wed. Nov.28th @ BWRC*
*Final report due Mon. Dec. 3rd @ 125 Cory*

## 1. Putting it all together

The principal goal of Phase III of the project is to assemble and optimize the entire SliceL. For convenience, the system block diagram is repeated below.

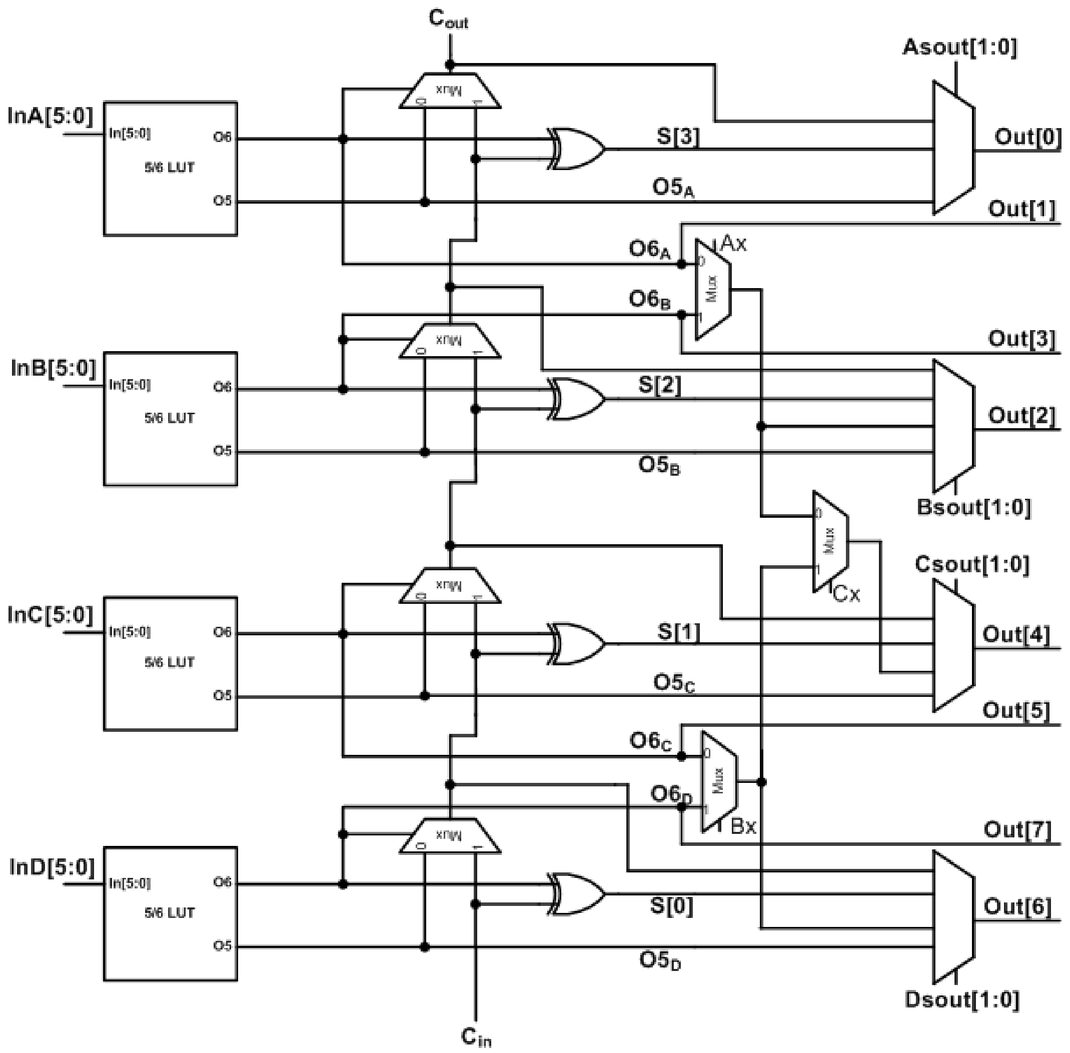

**Figure 1: Block Diagram of the slice**

In the previous two phases, explicit instructions were given on how to implement the address decoder and an SRAM row used within the LUTs of the slice, the MUX circuitry to propagate to the output the desired signal, and the vertical logic. In this third phase, you will design a read and write driver for the SRAM, optimize/assemble the entire system, and create input vectors to verify the functionality of the design.

This phase will have an intermediate check-point to help you organize your work. The deliverables for the check-point are:

1. A full initial schematic implementation of the design.
2. A test file containing input vectors to verify the system functionality.
3. A detailed project plan, reporting which circuit optimizations you intend to make and how you intend to schedule your work to successfully fulfill all project requirements.

More details will be given in the following sections.

This phase of the project allows for the most creativity, but will also be the most time consuming of the three phases. We can not over-emphasize how important it is that you **START EARLY** in order to allow yourself plenty of time to design, optimize, and assemble the complete SLICEL.

## 2. Peripheral Circuits

Besides the SRAM cells and the decoder, additional peripheral circuits are required to allow writing and reading from an SRAM. Figure 2 shows the schematics of representative peripheral SRAM circuits.

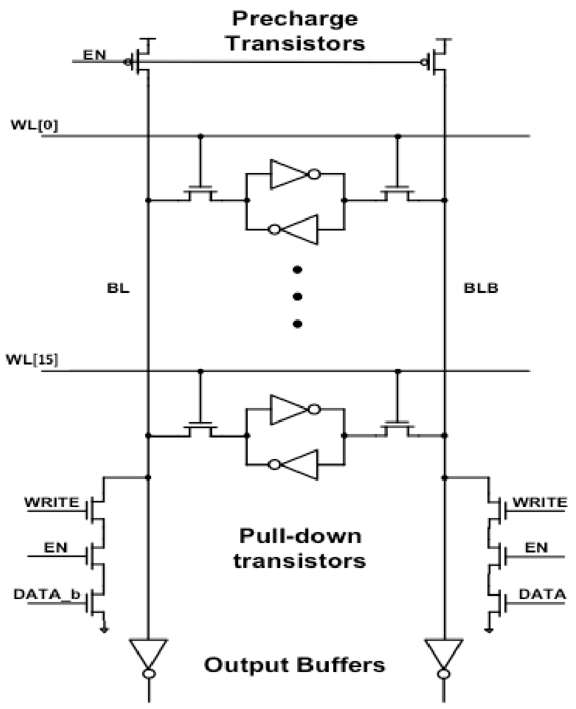

**Figure 2: Pre-charge, pull-down and output buffer schematics.**

Every Read/Write operation is enabled by an EN signal which is set to $V_{DD}$ during a read/write operation and to GND otherwise.

After every read/write operation, the bitlines need to be pre-charged back to $V_{DD}$ – this is achieved by connecting a PMOS pre-charge transistor to each of the bitlines. The gates of all pre-charge transistors can be connected to the EN signal. (Note that you can use a separate pre-charge signal as well, but this signal should only be low when the WL's are also low.)

For every write operation, one of the two bitlines of each SRAM column is driven to GND. There are many possible implementations of the pull-down network. For example, it can be implemented as a stack of three NMOS transistors whose gates are driven by the WRITE, EN, and DATA/DATA_b signals respectively. During a write operation, both the EN and WRITE signals are driven to $V_{DD}$. Depending on whether DATA or DATA_b is a "1", either BLB or BL is respectively driven to GND and DATA/DATA_b are written into the SRAM cell selected by the WL.

In order to drive the output capacitance of the LUT, you may want to use a buffer (inverter) at the output of your SRAM. You are free to size the output buffer as you like.

1. **Read Operation**
   Example waveforms associated with the SRAM read operation are shown below.

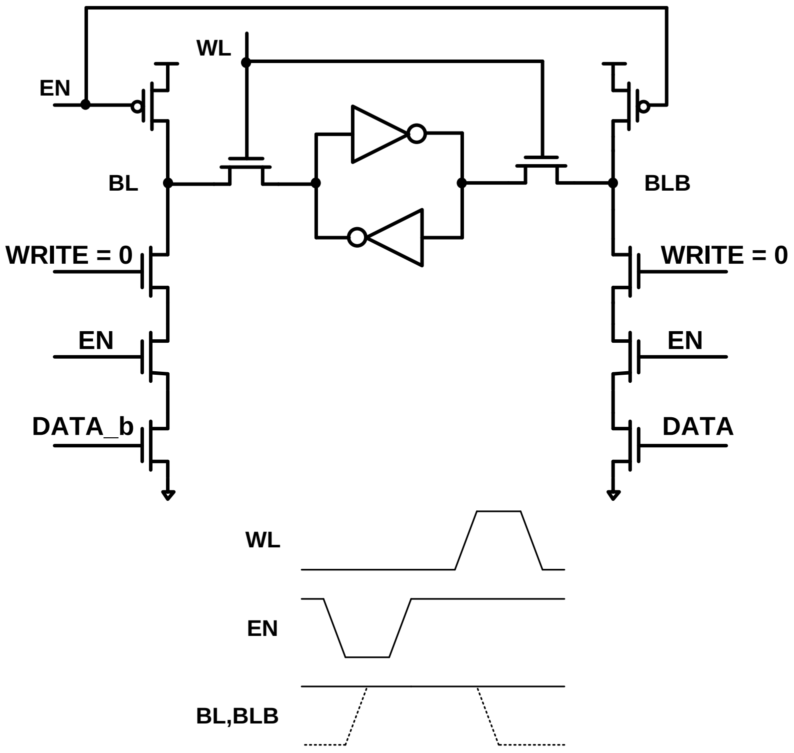

**Figure 3: SRAM Read**

2. **Write Operation**
   Representative waveforms for the SRAM write operation are shown below.

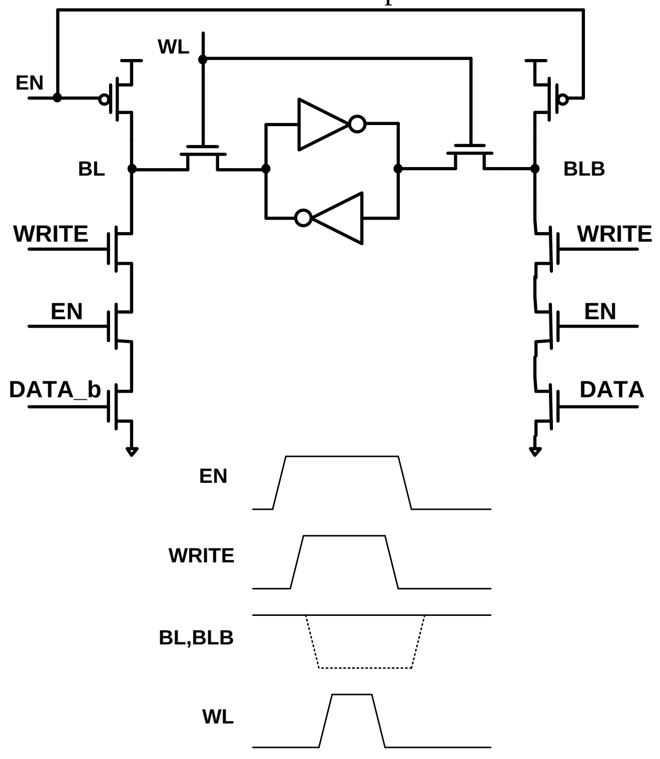

**Figure 4: SRAM Write**

Typically, the pre-charge and pull-down/buffer transistors would be physically placed at the top and bottom respectively of the SRAM array, so the dimensions of the cell containing these devices should match the width of the SRAM cell. You are free to size the pre-charge transistors as you like, but you must make sure that the time it takes for the bitlines to get pre-charged to within 10% of $V_{DD}$ is less than the total critical path delay through your decoder. Similarly, you can also re-size the pull-down transistors but you must make sure that the time it takes for the bitlines to get discharged to less than 10% of VDD is less than the total critical path delay through your decoder.

A schematic and layout implementation of the read/write peripheral circuit that creates appropriate read/write signals and includes the output buffers has been provided to you at ~/ee141/fall12/project/peripheralrw/.

## 3. Design Verification

Verification of the correct functionality of a circuit is a critical step in any design. Besides creating a full schematic level implementation of the system, you are asked to create a test input vector to verify the functionality of your design as a deliverable for the check-point.
To limit verification run-time and make sure that you will all test the circuit under the same conditions, the test input vector should perform the following tasks in this order:

1. Program the slice with the truth table of the 8-input function
$$Y = (A + B + C + D)(\bar{E} + \bar{F} + \bar{G} + \bar{H})$$
In particular, you should map the 8 inputs in the following way: slice inputs InA/InB/InC/InD[0:5] to function inputs A-F, with InA/InB/InC/InD[0]-> A, InA/InB/InC/InD[1]-> B, etc., slice inputs Ax/Bx to function input G, and slice input Cx to function input H. The mapping is also reported in the table below.

| Slice Input | Function Input |
|---|---|
| InA/InB/InC/InD[0:5] | A-F |
| Ax/Bx | G |
| Cx | H |

2. For all $2^8 = 256$ input combinations:
   a. Read the slice output corresponding to the output Y of an 8-input function
   b. Read the slice outputs corresponding to the eight 5-input functions (these outputs can be interpreted as the eight cofactors of Y corresponding to all value combinations of inputs F,G and H)
3. Re-program the slice to perform the addition of two 4-bit vectors X[3:0] and W[3:0]. where these two vectors are defined as:
$$X[3:0] = (A[3:0] \,\&\, B[3:0] \,\&\, {\sim}C[3:0] \,\&\, D[3:0] \,\&\, E[3:0])$$
$$W[3:0] = \big((A[3:0] \mid B[3:0]\big) \,\&\, (C[3:0] \mid D[3:0] \mid E[3:0]))$$
In the equations above, each LUT input receives one bit each of the A, B, C, D, and E input vectors, with A representing position [0] and E representing position [4], and you should assume that the bottommost LUT represents the least significant bit. Finally, note that & above indicates a bit-wise AND, | indicates a bit-wise OR, and ~ indicates a bit-wise NOT. For example, all of this implies that:
$$X[0] = \big(InD[0] \cdot InD[1] \cdot \overline{InD[2]} \cdot InD[3] \cdot InD[4]\big)$$
$$W[1] = ((InC[0] + InC[1]) \cdot (InC[2] + InC[3] + InC[4]))$$
Please don't forget that you will also need to use the LUTs to compute the propagate and generate signals to support adding the two vectors X[3:0] and W[3:0].

4. Read the four summation bits S[3:0] and the Cout bit, corresponding to all $2^8 = 256$ input combinations of vectors X and W.

Additional inputs to the slice are required in order to allow writing into the slice, as explained in Section 2. You should use the following naming convention. Have a single EN signal and WRITE signal for the whole slice. Have a DATA and a DATA_b signal for each column of each LUT. You should thus have two sets of 16 signals, named DATA[15:0] and DATA_b[15:0]. In particular, DATA[0] and DATA_b[0] should be used to write the data addressed by inputs InA[4]=0 and InA[5]=0, DATA[1] and DATA_b[1] for input InA[4]=1 and InA[5]=0, DATA[2] and DATA_b[2] for input InA[4]=0 and InA[5]=1, etc. Note that for the sake of simplicity, when you complete the final layout, you do not need to route the DATA/DATA_b inputs all the way to the edge of your block – you can simply place pins on an appropriate metal layer. Note however that these are the only interface signals that you are allowed to do this with – all others must be routed to the edge of your block.

You are free to choose any method to generate the test input vector file, but we strongly recommend looking at the func_check.vec file that we provided during Phase 2 of the project as a starting point, and that you write a simple script (e.g., in Matlab or Python) to automatically generate the input vectors to perform the 4 steps listed above. More tips will be given during discussion session.

## 4. Design Optimization

As with the previous phases, the primary constraint on your design is that it must function correctly. However, beyond building a functional design, in this final phase of the project it will be up to you to decide which design metrics you'd like to focus on optimizing. For example, you can try to build the fastest LUT/MUX/adder possible, minimize the system power consumption, pack the design into the smallest area, or shoot for a balance of two or three of these metrics. No matter which of these goals you pick, your final design should include substantial optimizations/modifications to at least one out of the three principal blocks: the LUT, the MUX logic, and the vertical logic.

As a first step, we will provide you with a more compact layout implementation of the SRAM cell. You should use this new implementation in your final design. We will post an announcement on Piazza when the layout files will be available in the folder ~/ee141/fall12/project/newSRAM/. We strongly encourage you to make use of this SRAM cell layout since coming up with optimized SRAM cell layouts can be a highly time consuming process.

Several examples of potential optimizations are described below; note however that these are only a few of the possible optimizations that can be made to your design. The optimizations you decide to apply are limited only by your own creativity, but you do need to adhere to the constraints on input capacitance, rise/fall times, margins, etc. provided in the project phase I handout.

1. **Re-organize the SRAM array**
   The SRAMs used in the functional unit are logically 32x2 and we already implemented them as 16x4 to get a better layout aspect ratio. Nevertheless, this may still not be the optimal physical organization for the array. For example, the SRAM could be re-organized into a more square configuration (8x8 in the example shown below) by introducing a further column MUX. This obviously has implications on both the lengths of the bitline and wordline wires as well as the decoder implementation, so if you try this optimization you should be sure to analyze the impact of the array organization on the metrics you care about. Note that if you choose to re-organize the SRAM array, you will need to change also your verification deck, to take into consideration the modified signal connectivity.

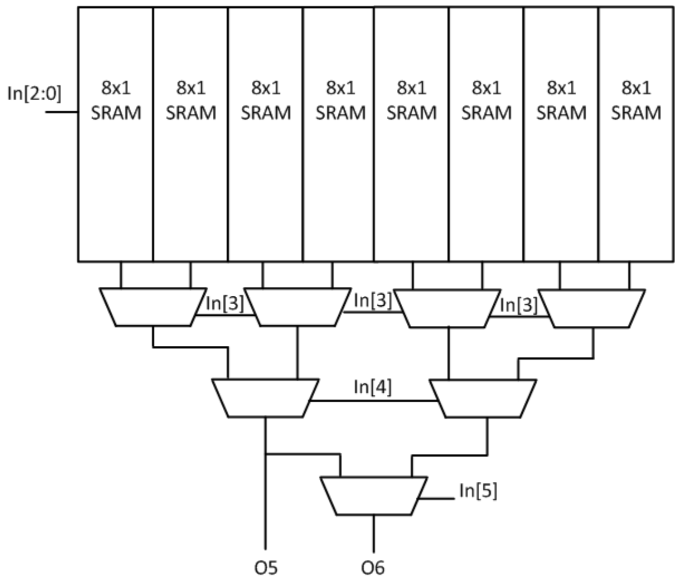

**Figure 5. Re-organizing the SRAM array.**

2. **Use more advanced logic styles (domino, pass-transistor, pseudo-NMOS)**
   Instead of using static CMOS logic to implement the decoder(s), vertical logic, and MUXes, you can explore the use of alternate logic styles. For example, you might try to use domino logic. Remember that there are almost always tradeoffs when choosing between logic styles, so it will be up to you to determine which logic style best fits with your original optimization goals.

3. **Use a lower $V_{DD}$ and/or add one or more extra power supplies to reduce power**
   Reducing the supply voltage of a digital circuit is one of the most effective means for trading off performance for reduced power consumption, and thus one option you can explore is to lower the $V_{DD}$ for the entire functional unit. If you are shooting for a balance of speed and power, you can also explore using a separate supply voltage for various pieces of the design. For example, you might choose to use a lower voltage in the final decode drivers in order to reduce their switching power without significantly impacting the overall delay. Note that if you use a lower supply voltage somewhere in your design, you should re-characterize the transistor parameters ($C_G$, $C_D$, $R_{sq}$) at this new voltage.

In your final report, you should state what your objectives were for the optimizations you chose to apply. Whether you optimized for speed, power, layout area, or some combination of the three, please be very clear about the methods you used to improve your design. A big part of the judging of your project will be based on how well the optimizations you made actually coincide with your stated goal. Whichever optimizations you choose to do, remember that you do need to complete an LVS and DRC clean layout of the entire design – including the four 5/6 LUT, the MUX logic, the vertical logic and the write and read peripheral circuits.

## 4. Analysis and Simulation

1. **Functionality**
   In order to ensure that your design functions properly, you will need to simulate the entire operation of your slice (including your optimizations) using the input patterns described in the earlier section.

2. **Performance**
   The latency of the slice depends on the selected configuration. To ease your calculations, you should characterize the latency of only two configurations:

   (a) Worst-case delay from the transition of one of the inputs InA[5:0] / InB[5:0] / InC[5:0] / InD[5:0] to the data appearing at the output, assuming that the slice is configured to implement one 8-input function, and
   (b) Worst-case delay from the transition of one of the inputs InA[5:0] / InB[5:0] / InC[5:0] / InD[5:0] to the completion of the addition, i.e., all sum signals and Cout bit (including propagating those signals through any downstream further MUXes).

   You are also asked to provide the critical path delay of the individual components in your design (i.e., the decoders, the SRAM array, MUX logic and vertical logic).

3. **Area**
   There is one main area measurement that needs to be made for the final phase –the area of your complete design. In order to measure the area, draw (with rulers) the smallest rectangular box that fits around your entire design. The area is simply the length times the width of the drawn rectangle. Remember that all of your input/output pins must be at the edges of this rectangle.

4. **Power**
   You will also need to hand estimate as well as simulate the power consumption of your circuit. Since the power consumption of the design depends upon the frequency it operates, to compare all of the designs equally you will measure power with a fixed 100MHz clock frequency in HSPICE. We will provide you with an HSPICE deck that extracts the power consumption of your design (please watch for an announcement on the availability of this deck too) – further details on using this deck will be provided in the discussion sessions.

5. **Noise Margins**
   If you have not changed the supply voltage of the SRAM cell (or the cell itself), then you already have the noise margins of your cell and no further work is necessary. If you have altered the cell or use a lower power supply voltage, you will need to re-simulate the read and write static noise margins (the ones based on the butterfly curves) as instructed in phase I. Please note that your altered cell must meet the following requirements: the read static noise margin must be at least 100mV, and the write static noise margin must be at least 300mV.

## 5. Report

The quality of your report is as important as the quality of your design. Be sure to provide all relevant information and eliminate unnecessary material. Organization, conciseness, and completeness are of paramount importance. Do not repeat information we already know. Use the templates provided on the web page (Word and PDF formats). Make sure to fill in the cover page and use the correct units. Turn in the reports for each phase in the homework drop box. In addition, mail an electronic version of your final report and the poster as a Word or PDF file to ee141-project@bwrc.eecs.berkeley.edu. You should also provide an electronic version of your final netlist, HSPICE decks, and your LVS/DRC reports showing that you passed both of these checks.

### 5.1. Check-point

**Full initial schematic netlist, verification test-bench, and project plan.**

For the checkpoint, you are required to provide us with the following two files:

- An updated version of the file checkpoint.sp (provided in ~/ee141/fall12/project/checkpoint/), containing a full schematic level HSPICE netlist of a preliminary slice design, wrapped into a .subckt definition.
- An input test vector called input_checkpoint.vec capable of performing all the steps described in Section 3, to check the functionality of your circuit. If you used a script to generate this file, please submit this script as well.

Note that this preliminary complete slice does not need to be optimized. In fact, beyond helping you to develop the verification deck, the main purpose of this preliminary design is to help make it more clear which pieces of the design are the most attractive targets for further optimization.

In order to make sure that you are going in the right direction with your project and provide an opportunity for us to give you early feedback, the second deliverable for this intermediate checkpoint is a short project plan that includes the following information:

- The primary objective of your optimizations (i.e., highest speed, lowest area, etc.).
- A list of the design optimizations you plan to attempt.
- A schedule outlining when you plan to complete (i.e., have a DRC/LVS clean layout) and characterize each component of your design as well as the slice as a whole. You should make your plan as realistic as possible, and be sure to budget in time for unexpected challenges as well as design iterations.

**The deliverables for the check-point are due at 5PM Sun. Nov 18th** by email. You should send us three unzipped files to ee141-project@bwrc.eecs.berkeley.edu. We will provide you with our feedback shortly after you submit the report, so early submissions are strongly encouraged Moreover, we will upload our verification test-bench shortly after the deadline, so that you can use it in case you had problems in this phase. For this reason, no late submissions will be granted. The deliverables for the check-point account for 30% of the total score of the project.

As mentioned above, we strongly encourage you to complete the full functional schematic and verification test-bench ***before*** writing-up the project plan. This will allow you to gain a better understanding of the system before proposing any optimizations.

## 5.3. Final Report for Phase III

The format of the final report and instructions on printing your poster will be provided on the web – please stay tuned for an announcement.

## 5.4 Note on Due Dates

The poster session has intentionally been scheduled several days before the final report is due so that you have an opportunity to get more detailed feedback on your design and make adjustments based on this feedback before submitting your final report. This does not however mean that you should delay completing the design until the final report is due – every team **must** present a poster, and the quality of the poster (and your presentation) will be included as a part of the final project score.