# Controls and Menus

## Inputs

This subsection discusses the implementation of the PS/2 keyboard, pushbutton and slider switch inputs.

## Menus & Screen Navigation

In this subsection, the menu and screen driver is discussed, along with how they are interfaced with from the main program loop.

There are 3 main types of screen to be reached within the pong game: a start-up screen, loaded once the game is switched on; a menu system, which needed to contain important settings for the game and be reachable from the game; and a game-play screen, where the pong engine is utilised.

The start-up screen was simplest to implement, as it is non-interactive and runs on a timed basis; this meant that it could utilise purely blocking functions, including sound generation. At this point, the inputs are not enabled, so as not to produce unwanted effects for keypresses. A sound is played and a greeting displayed over a colour fill; this gave the added benefit at the time of debugging of being a useful tester for display and sound functions. This screen displays for a total of 4 seconds – 2 for the sounds to play in addition to an extra 2 second sleep.

\begin{figure}[H]

  \centering

  \lstinputlisting[language=C, firstline=145, lastline=158, frame=bt]{../DE1\_SOC\_PONG/pongScreens/pongScreens.c}

    \caption{Code used to run start-up screen.}

\end{figure}

\- \\

\begin{figure}[H]

  \centering

  \fbox{\includegraphics[width=0.4\textwidth]{./images/StartScreen.png}}

  \caption{Start-up screen}

\end{figure}

\- \\

The menu system contained both blocking and non-blocking elements, including input and timer ISRs, sounds and animation, making it more complex.

# Audio Output

This section explains the method used to implement sounds in the final design. In order to include sounds, the DE1-SoC WM8731 and I2C drivers were utilised in combination with an example code. [Reference T. Carpenter’s work from lab 4??]. It was noted initially that the code was slow to perform due to the float arithmetic involved in calculating the sinusoid of an angle, which caused the FIFO buffer of the audio device to overflow. We realised therefore that there were 2 real ways of approaching the problem: either, to utilise the onboard FPU of the DE1-SoC, or to create a faster approximation of the \[\sin\theta \] function.

\\ \- \\

The code below demonstrates the fast sin function that was devised. The values of \[sin\theta \] from 0\degree to 90\degree were first calculated in Matlab, multiplied by 10^5 and rounded to whole numbers. These results were then copied into a lookup table within the ‘lookupSin(unsigned int degree)’ macro. The storing of the numbers as fixed integers as opposed to floating point numbers served to save on memory usage.

\begin{figure}[H]

  \centering

  \lstinputlisting[language=C, firstline=141, lastline=167, frame=bt]{../DE1\_SOC\_PONG/pongSound/pongSound.c}

    \caption{Code used produce fast sin lookup results for whole degrees.}

\end{figure}

\- \\

By utilising the characteristics of the \[y=\sin\theta \] function over the period 0\degree to 180\degree, where [y] is mirrored about [x=90\degree], and the characteristic that the function is mirrored about [y=0] at [x=180\degree], it was then possible to find an approximation of \[\sin\theta \] for any whole positive integer \theta up to 360\degree. Within the Sound macro, the float input to ‘lookupSin’, indicating the phase, was wrapped to a maximum of 360\degree and cast to an integer type. The maximum error in this lookup function at whole integers was around 4.8\times10^-6.

\\ \- \\

Problems encountered – sound generation stopped temporarily if a key is pressed, solution, disable sounds temporarily during output