The All Programmable SoC enters the maker arena

By Aaron Behmanand Adam Taylor, Xilinx

The maker sphere provides both a popular hobby for many and an inspiration for young people to follow careers in science, technology, engineering and math subjects. Many of the projects within this sphere contain an embedded processor, commonly a member of the Arduino or Raspberry PI families, to provide the intelligence required.

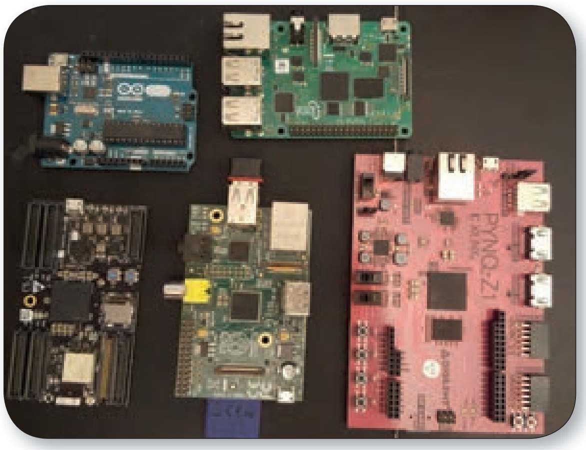
Figure 1. Maker development boards: left to right Arduino, ZynqBerry, Pynq, Raspberry and Snickerdoodle.

■ Both Arduino and Raspberry PI are supported by a development environment which provides a range of software libraries, modules and examples. These aid the developer to quickly and easily interface with a range of peripherals from cameras to accelerometers and motors. It is this ease of use which reinforces the popularity of these processors within the maker sphere.

Until recently makers have considered All Programmable SoCs to be outside the sphere and something for the more specialist engineer. This, however, is no longer the case with the introduction of Zynq-based boards like the ZynqBerry, Pynq and Snickerdoodle and software-based development techniques. These boards are fitted with devices from the Xilinx All Programmable Zynq-7000SoC family which combines dual ARM Cortex-A9 processors with programmable logic from the All Programmable Artix-7 FPGA range. This provides the ability to accelerate the functions within the programmable logic fabric to significantly increase system performance. Traditional development however has segmented the design of the programmable logic from the software development, requiring specialist development experience to implement programmable logic design. But this is no longer the case. When these boards are coupled with the latest development environments which allow the application to be defined entirely in software, they become very interesting for this sphere. Especially, as these development environments provide the ability to exploit the programmable logic without the need for the user to be a FPGA specialist, providing the best of both worlds.

There are two development environments which can be used to create applications for these Zynq-based development boards. The first of these is the SDSoC development environment which is eclipse based. This environment allows the development of the application in C or C++, and then to seamlessly move functions from running on the ARM Cortex-A9 processors for acceleration in the programmable logic. The SDSoC environment uses High Level Synthesis (HLS) to move the selected C function in to the programmable logic. Once the HLS has completed connectivity, framework is used to integrate the HLS module with the software application. Apart from the performance boost provided by the acceleration of the function now in the programmable logic, the process is transparent to the user. Moving functions between the processors and the programmable logic is extremely simple and controlled within SDSoC using the project overview. When it comes to operating systems, SDSoC supports Linux which is commonly used within the maker sphere along with a real-time operating system (FreeRTOS) and a bare metal approach.

The second approach is provided by Pynq, which provides a development framework based upon Python and Juypter notebooks. Both are executed on a Linux distribution running on the processors while the programmable logic has a defined overlay which provides connections to the peripherals provided on the Pynq. Within the Linux distribution to support the peripherals and hardware overlay, there is a defined Pynq package which allows us to directly interface with the peripherals using Python. As Pynq provides two PMOD interfaces this package delivers significant support for a range of PMOD such as ADC, DAC to ease the integration with the Python application. Within Pynq the programmable logic is loaded with one of many overlays to offer hardware acceleration, there are several open source overlays in addition to the provided basic overlay. The users can programme Pynq by connecting to the Juypter notebook server via a web browser. Once connected to this notebook, they can develop and document their Python applications to run on Pynq. The ability to use Python and directly interface to the PMODs using Python provides for a very powerful development platform.

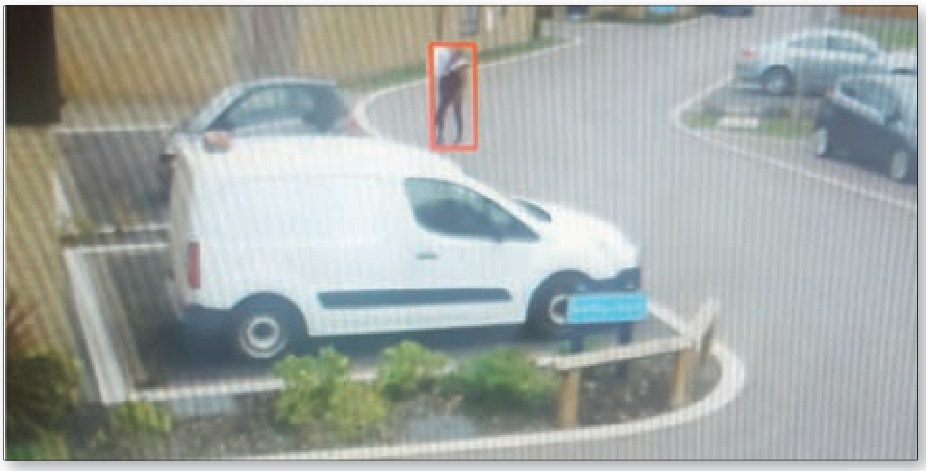

Figure 2. Result of the image tracking application

Both development methodologies provide the ability to use open source embedded vision frameworks like OpenCV to perform embedded vision applications. These applications can use web cameras when the Linux distribution supports the USB Video Class, or specific cameras like the Raspberry PI camera supported by the ZynqBerry. OpenCV allows them to develop in either C/C++ or Python, using this framework they can quickly and easily implement complex image processing algorithms using the acceleration of Zynq programmable logic to provide significantly higher performance. These applications can process images and detect objects or faces and more, using this framework.

When it comes to implementing a simple object detection algorithm they can run Linux, Python and OpenCV on Zynq-based platforms. Let us examine what is involved with implementing a simple object tracking system using OpenCV and a web camera. The algorithm they will implement is:

1) Capture the first frame from the web cam. This first frame serves as the reference background frame. They will detect any changes that occur in the scene from this first frame.

2) Convert the colour space from RGB to grey scale. This is a commonly used image-segmentation technique used to create binary images. Image segmentation covers several techniques that divide an image into multiple segments often called super pixels. Segmentation allows for easier analysis of the segment contents. In their application, they can use thresholding to segment the background from the foreground. This will produce a binary image.

3) Perform a Gaussian blur on the image. The performance of many image-processing applications that detect objects or edges can be negatively affected by noise present within the grabbed frame. Blurring the image prior to further processing reduces noise within the frame and this technique is often used for image processing, particularly for edge-detection algorithms (for example, the Laplacian of the Gaussian edge-detection algorithm). The result of this operation becomes the reference image against which to detect a change.

4) Repeat steps 1 to 3 again, that is capture another image from the web cam, convert the colour space to grey scale and perform a Gaussian blur on the image.

5) Calculate the absolute difference between the reference frame and the most recent captured image.

6) Perform a thresholding operation on the absolute difference to create the binary image.

7) Perform a morphological operation to dilate the image to enlarge any differences.

8) Find the contours remaining in the binary image and ignore any contours with too small an area.

9) Draw a box around each of the contours detected and display the original captured image over the HDMI output.

Developers can run the resultant Python code either directly on ZynqBerry or within a Juypter note book on Pynq. What they see is an image like the one below which identifies the changes from the reference image with boxes highlighting the differences. This example demonstrates both the power and ease with which they can use Zynq to perform their embedded vision application, using familiar open source frameworks. The presented development environment demonstrates the ease with which developers can exploit the capability of several Zynq- based development boards. These development environments enable the user to create applications which execute on the processors within the processor but accelerate their designs using programmable logic. ■