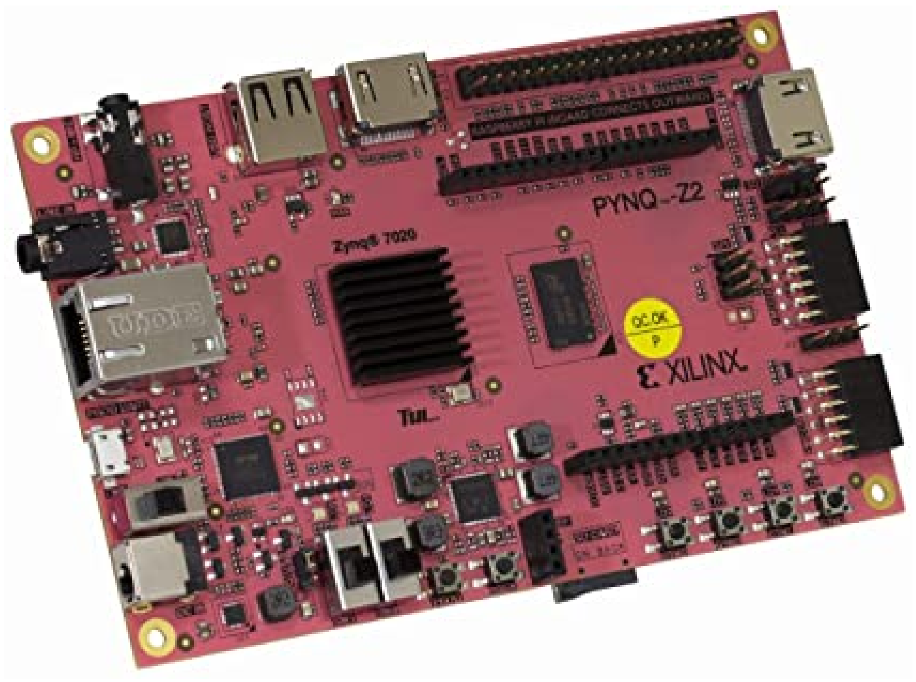
Assignment 2 - IMU data visualization through BRAM and CPU

Embedded Systems

Group 6
Karina Bay Bloch (kablo17) & Nicoline L. Thomsen (nthom18)

November 2021



Contents

1 Introduction

The purpose of this assignment is to build a digital circuit to interact with a given IMU and send the data to the CPU on the development board, which is the Pynq-Z2. Since it is built to work with the features of the FPGA board, where the CPU is included, this assignment will work with a Software-hardware interface, that works through memory. This can also be defined as PL-PS communication in this situation.

Gathering data from the IMU via SPI is discussed in a previous assignment, and will not be explored in detail here. The data from the IMU will be sent to a BRAM on the FPGA, where it will be fetched by the CPU. The CPU will process the data and show it to the monitor (Via. Terminal). This process should be continuous as long as it is enabled to do so. See figure 1 to see the plan of action for the layout of the project.

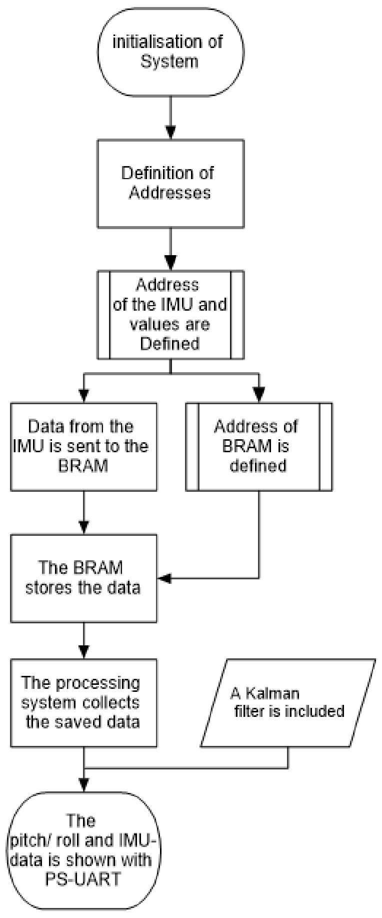

initialisation of System
Definition of Addresses
Address of the IMU and values are Defined
Data from the IMU is sent to the BRAM
Address of BRAM is defined
The BRAM stores the data
The processing system collects the saved data
A Kalman filter is included
The pitch/ roll and IMU-data is shown with PS-UART

Figure 1: Flowchart of the process that is sought after in this assignment

Figure 1 shows a block diagram of the overall system, which consists of an IMU and some features on this FPGA such as the BRAM and the CPU.

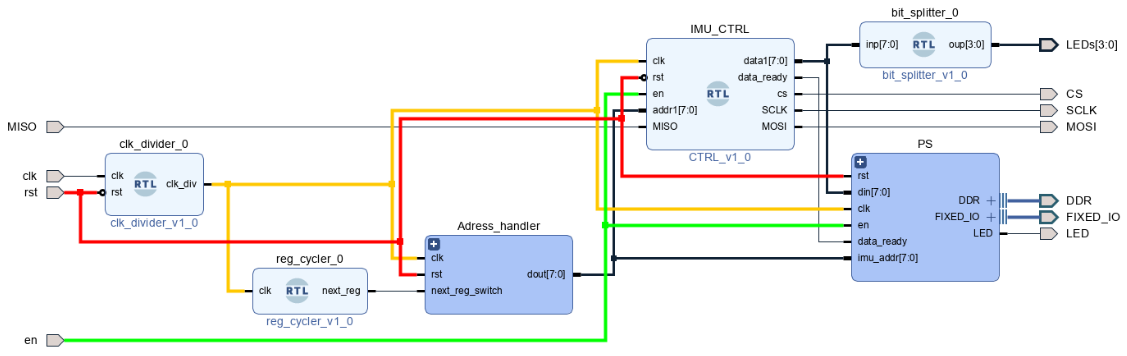


Figure 2: Block Diagram of system

2 Programmable Logic

The purpose of the SPI is the same as in the last assignment, as it is the protocol for communication the FPGA and IMU. This time, the objective is to acquire data from the gyroscope and accelerometer at the same time. The SPI works with a clk, an address handler and the control for the IMU. The setup, with focus on the IMU/SPI, is seen in figure 2. As a notice, the Adress_Handler and IMU_CTRL have been borrowed from the Embedded Systems GitHub. The values from the IMU, will be stored in the BRAM of the FPGA. The function of the BRAM (Block RAM) will be explained in the next section of this report.

3 Processing System

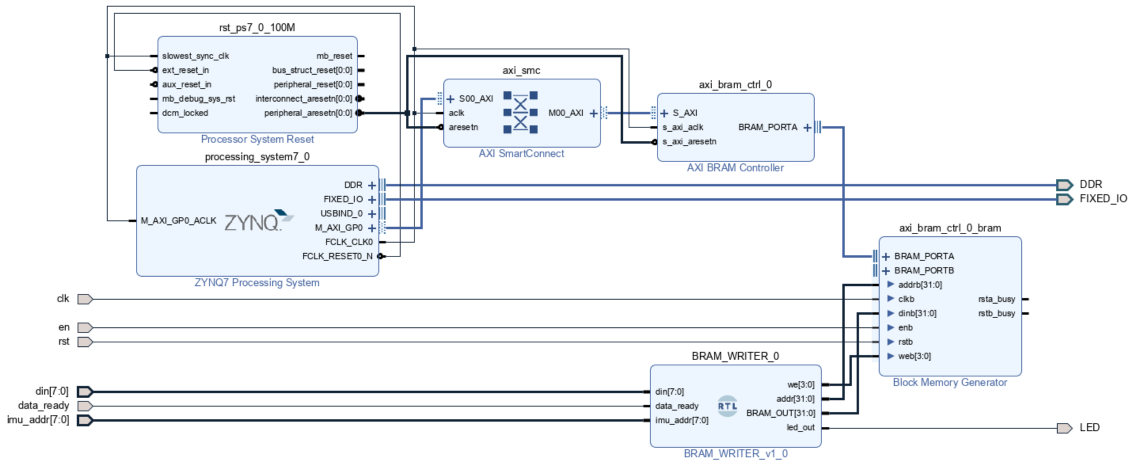


Figure 3: Caption

The new addition to the system is working with the BRAM and the CPU. What is seen in figure 3 is the addition of 2 IP's which is:

- `The ZYNQ7 Processing System IP`
- `AXI_BRAM_ctrl_0_BRAM`

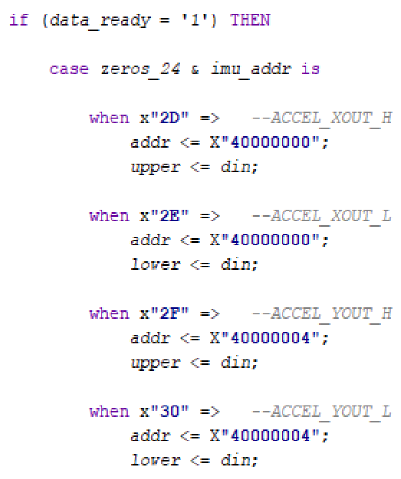
The `rst_ps7_0_100M`, `axi_smc` and `axi_bram_crtl_0` are auto generated blocks by Vivado, to implement the system with the specific development board Pynq-Z2.

3.1 ZYNQ7 Processing System

This IP is a way to communicate with the programmable system on the development board as software interface around the Zynq-7000 Processing System. It consists of an SoC style integrated PS and a PL unit. It basically acts as a logic connection between the PS and PL and can help with the integration of customised and embedded IP cores. To be able to communicate with the CPU itself, code was written via. Vitis IDE, and the way this works, is the CPU being programmed in C while still retaining the unaltered connectivity and other tings, such as logic functions. Here, it stitches the interfaces between the processing system and programmable logic together. In this assignment it retrieves the IMU-data from the BRAM, with the help from the additional AXI blocks to sort and define the different values of the IMU. When the values are then known, they will be filtered with a Kalman filter, and lastly they will be visualised in the PS-UART serial port.

3.2 BRAM

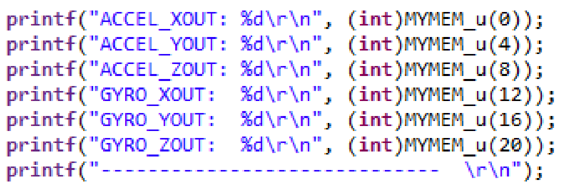
The BRAM is the common memory between the FPGA and CPU. `BRAM_WRITER` controls which address in the BRAM is written to. This will be dependent on the IMU register the sensor data is read from, to dedicate each sensor type data to its own register in the BRAM. On figure 4 is a code example of how the addresses are selected and written to. The addressing starts at `0x4000_0000` and span 32 bits. These registers however are 8-bit, so the next address would be offset by 4 and be `0x4000_0004`.



```
if (data_ready = '1') THEN

    case zeros_24 & imu_addr is

        when x"2D" =>    --ACCEL_XOUT_H
            addr <= X"40000000";
            upper <= din;

        when x"2E" =>    --ACCEL_XOUT_L
            addr <= X"40000000";
            lower <= din;

        when x"2F" =>    --ACCEL_YOUT_H
            addr <= X"40000004";
            upper <= din;

        when x"30" =>    --ACCEL_YOUT_L
            addr <= X"40000004";
            lower <= din;
```

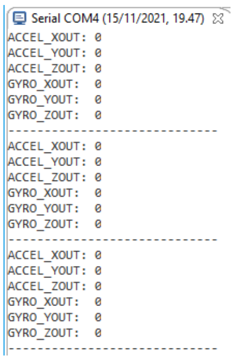
Figure 4: Code from `BRAM_WRITER`. This case continues with the rest of the sensor data, including the gyro.

The CPU can then read from the BRAM addresses and access this data. However, the implementation of this did not yield the expected result. The implementation and serial output is shown in figure 7. It is unknown why this occurs.



```
printf("ACCEL_XOUT: %d\r\n", (int)MYMEM_u(0));
printf("ACCEL_YOUT: %d\r\n", (int)MYMEM_u(4));
printf("ACCEL_ZOUT: %d\r\n", (int)MYMEM_u(8));
printf("GYRO_XOUT:  %d\r\n", (int)MYMEM_u(12));
printf("GYRO_YOUT:  %d\r\n", (int)MYMEM_u(16));
printf("GYRO_ZOUT:  %d\r\n", (int)MYMEM_u(20));
printf("----------------------------  \r\n");
```

Figure 5: C code accessing BRAM data and printing it to serial port via UART

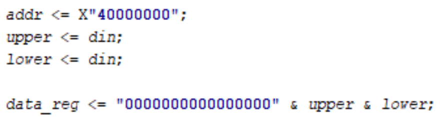


```
Serial COM4 (15/11/2021, 19.47)
ACCEL_XOUT: 0
ACCEL_YOUT: 0
ACCEL_ZOUT: 0
GYRO_XOUT:  0
GYRO_YOUT:  0
GYRO_ZOUT:  0
----------------------------
ACCEL_XOUT: 0
ACCEL_YOUT: 0
ACCEL_ZOUT: 0
GYRO_XOUT:  0
GYRO_YOUT:  0
GYRO_ZOUT:  0
----------------------------
ACCEL_XOUT: 0
ACCEL_YOUT: 0
ACCEL_ZOUT: 0
GYRO_XOUT:  0
GYRO_YOUT:  0
GYRO_ZOUT:  0
----------------------------
```

Figure 6: UART output

Figure 7: C code showing all sensor data from BRAM

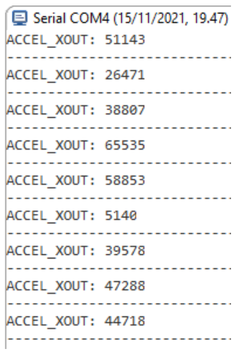
It was a success to write and read from only one BRAM register, show with the code in figure 8.



```
addr <= X"40000000";
upper <= din;
lower <= din;

data_reg <= "0000000000000000" & upper & lower;
```

Figure 8: Code from `BRAM_WRITER` that writes to only one address

This is the solution implemented in the submitted code. For this to function only one address should be read from the IMU, and therefore the `reg_cycler` seen in figure 2 has been removed, and replaced with a physical switch on the board. The serial output for this implementation is seen in figure 9



```
Serial COM4 (15/11/2021, 19.47)
ACCEL_XOUT: 51143
----------------------------
ACCEL_XOUT: 26471
----------------------------
ACCEL_XOUT: 38807
----------------------------
ACCEL_XOUT: 65535
----------------------------
ACCEL_XOUT: 58853
----------------------------
ACCEL_XOUT: 5140
----------------------------
ACCEL_XOUT: 39578
----------------------------
ACCEL_XOUT: 47288
----------------------------
ACCEL_XOUT: 44718
----------------------------
```

Figure 9: UART output of single address value

4 Conclusion

The assignment regarded the collaboration between the PL part and PS part of the development board. The objective were to be able to extract data from the IMU and store this data in the BRAM, so that the CPU would have access to the data. With this data and the ability to program the CPU of the development board, it could be visualised in the terminal of the PS-UART. By defining the addresses of the IMU, and the address of the BRAM, the values would be retrieved by the CPU via. AXI to define and later be shown in the terminal. With some issues regarding splitting the IMU data into its corresponding values, by reading from multiple addresses, the Kalmin filter was never implemented.