DAEbot Reflective Operator Accelerator+

New trends in Research

Javier Reyes

Fachhochschule Dortmund
University of Applied Sciences and Arts

Master Embedded Systems for Mechatronics
Dortmund University of Applied Sciences and Arts
Germany

March 22, 2018

Contents

Chapter1

DAEbot Autonomous Robot

The DAEbot project stands for Distributed Architectures Evaluation robot, a modular system that allows different approaches for distributed hardware and software architectures to be tested [1].

The robot is designed and built with common hardware (evaluation boards and popular sensors-actuators), so that it can be easily replaced or tested with different devices, as well as test different software solutions or methodologies. Physically, the robot represents a layered structure, to resemble the software architecture approach that is used, which is based in the OCM software architecture.

1.1 OCM architecture

Typical mechatronic systems complexity grows exponentialy with every new technological step, which requieres a careful process of defining the right architecture for mechatronic systems. On this base concept, the goal for the DAEbot architecture is graphically represented in the figure 1.1.

The OCM structure is based on the work of the Mechatronics Laboratory Paderborn as a hierarchical structure for mechatronics systems [2]. This structure defines three different layers that allows the clear seaparation of controllers, considering the real-time characteristics, and therefore allowing a more comprehensive development process.

Controller Layer

This is the lowest level layer, where the traditional controller is defined as a traditional control loop, usually based on a mathematical model. This way the controller gains flexibility in terms of the configuration parameters, and can be individually simulated and tested (XiL), allowing different operation modes which will be *operated* from the next layer.

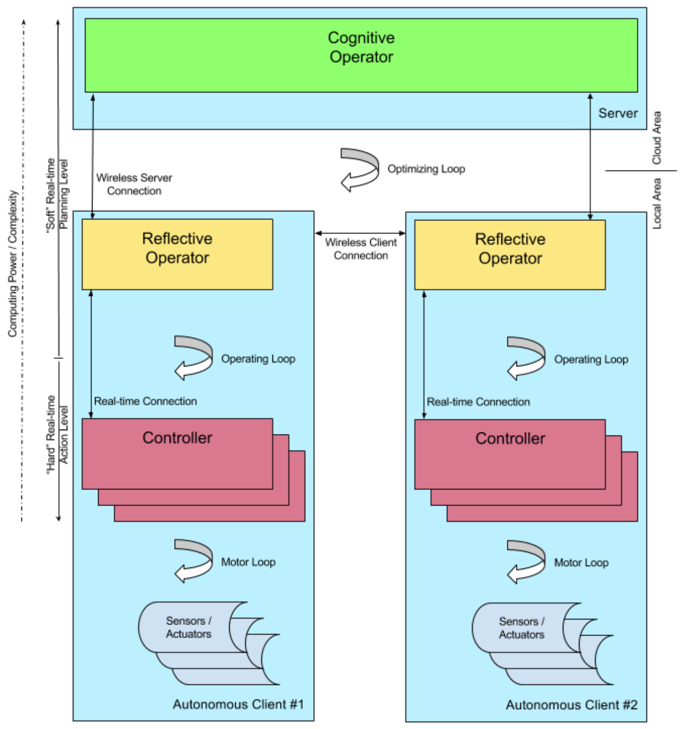


Figure 1.1: OCM software architecture model.

Reflective Operator
: On this layer, the monitoring and control is implemented, and even allow communication with other systems in a network.

Cognitive Operator
: For more complex systems, that need coordination/communication interactions or learning algorithms, the Cognitive layer expands the functionality.

The DAEbot defined structure related to the OCM architecture can be seen in figure 1.2.

Given the modular architecture, the design and develoment of each of the elements in the system can be modularized and worked independently, under the guidelines of the structure.

1.2 Refective Operator+ Zynqberry

The zynqberry board, with its FPGA capababilities, is considered a second operator with the expectation of evaluating a true parallel application performance inside the OCM architecture. The communication from and to the main Reflective Operator (Raspberry Pi) and the Controllers is provided by CAN bus, with an optional link via Ethernet for high data payloads.

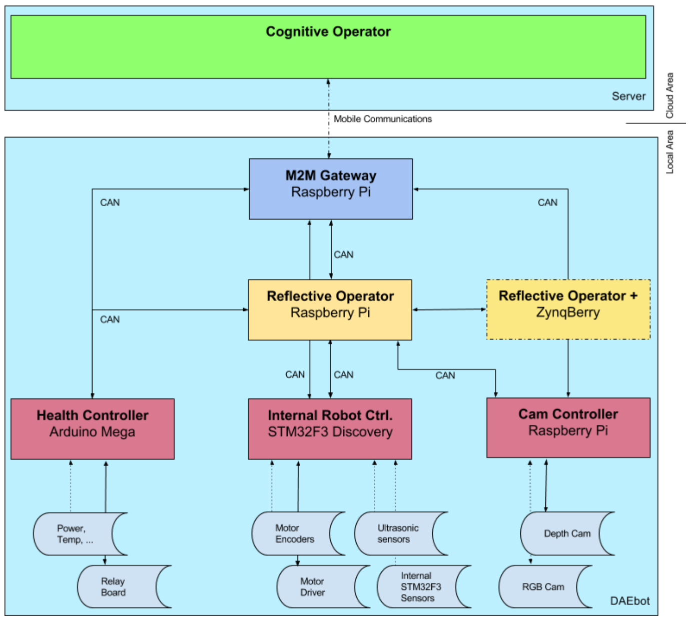


Figure 1.2: DAEbot architecture.

Chapter2

Zynqberry Evaluation Board

The Zynqberry board, developed by the company Trenz Electronic GmbH, is a SoM (System on Module) based on the Xilinx All Programmable Zynq-7010 SoC (XC7Z010) along with standar peripherials in an industrial-grade Raspberry Pi form factor. Some of the most relevant features are listed below [3]:

- Xilinx Zynq XC7Z010-1CLG225C
  - 512 MByte DDR3L SDRAM
  - 16 MByte Flash
- Raspberry Pi Model 2 form factor
- LAN9514 USB hub with Ethernet
  - 4 x USB with power switches
  - 100 MBit Ethernet RJ45
- Micro SD cad slot
- HAT header with 26 I/O's

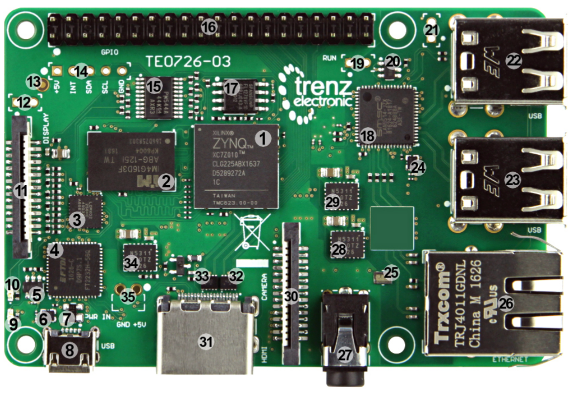


Figure 2.1: Top view of the Znyqberry board.

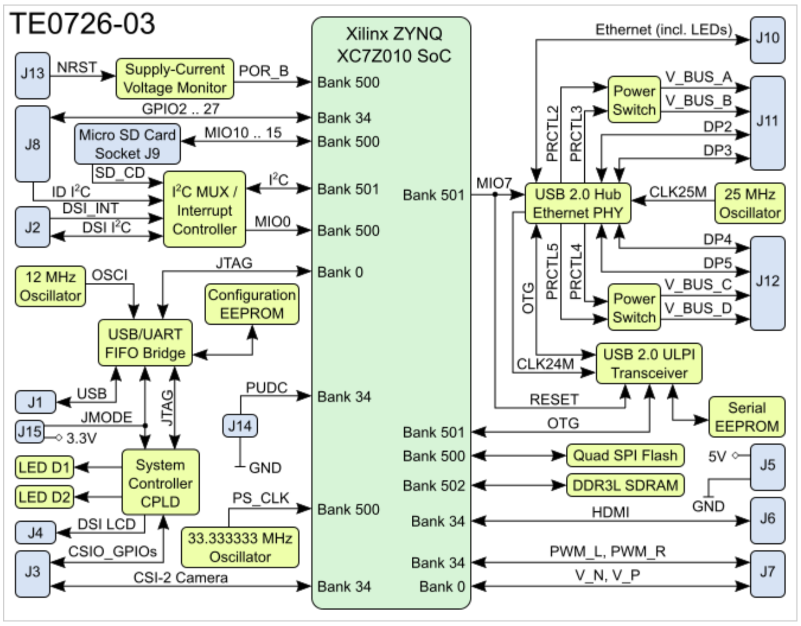


Figure 2.2: Block diagram of the Zynqberry 726.

- HDMI Type A
- DSI Connector (display)
- CSI-2 Connecto (camera)
- Micro USB
  - power input
  - USB UART
  - JTAG ARM- and FPGA-Debug
- 3.5 mm audio plug (PWM audio output only)

The physical device used in this project corresponds to the version 02 of the product, despite the fact that the manufacturer holds a newer 03 version. This needs to be considered during any reference consultation.

2.1 Xilinx All Programmable Zynq-XC7Z010 SoC

The Znyqberry TE-0726 02 board provides a SoC (System on Chip) device from the All Programmable Zynq-7000 family.

2.1.1 Device Achitecture

The Zynq-7000 SoC devices are part of the Xilinx All Programmable SoC architecture, integrating a dual-core ARM Cortex A9 processor (PS), and a 28 nm Xilinx programmable logic device (PL) into a single physical device.

Processing System PS

The Processing System (PS) is embedded into the FPGA device, providing and environment around it that makes possible more complex system configurations. The PS also includes on-chip memory and external memory interfaces. A more detailed characteristics description is available on [4] and [5].

ARM Cortex-A9 Application Processor Unit

- 2.5 DMIPS/MHz per CPU
- CPU frequency up to 1GHz
- Single and double precision Vector Floating Point Unit (VFPU)
- Timer and interrupts
  - Three watchdog timers
  - One global timer
  - Two triple-timer counters

Cache

- 32kB Level 1 4-way set-associative instruction and data caches (cpu independent)
- 512 kB 8-way set associative Level 2 cache (CPU shared)
- Byte-parity support

On-chip memory

- On-chip boot ROM
- 256 kB on-chip RAM (OCM)
- Byte-parity support

External Memory Interfaces

- Multiprotocol dynamic memory controller
- 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2 or LPDDR2 memories
- ECC support in 16-bit Mode
- Static memory interfaces
  - 8-bit SDRAM data bus qith up to 64 MB support
  - Parallel NOR flash support
  - ONFl1.0 NAND flash support (1-bit ECC)
  - 1-bit SPI, 2-bit SPI, 4-bit SPI, or two quad-SPI (8-bit) serial NOR flash

8-Channel DMA controller

- Memory-to-memory, memory-to-peripherial, peripherial-to-memory and scatter-gather transaction support

I/O Peripherials and Interfaces

- Two 10/100/1000 tri-speed Ethernet MAC peripherials with IEEE Std 802.3 and IEEE Std 1588 revision 2.0 support
- Two USB 2.0 OTG peripherials, each supporting up to 12 Endpoints
  - USB 2.0 compliant device IP core
  - Supports on-the-go, high-speed, full-speed, and low-speed modes
  - Intel EHCI compliant USB host
  - 8-bit ULPI external PHY interfaces
- Two full CAN 2.0B compliant CAN bus interfaces
  - CAN 2.0-A and CAN 2.0-B and ISO 118981-1 Standar compliant
  - External PHY interface
- Two SD/SDIO 2.0/MMC3.31 compliant controllers
- Two full-duplex SPI ports with three peripherials chip selects
- Two high-speed UARTs (up to 1 Mb/s)
- Two mastar and slave I2C interfaces
- GPIO with four 32-bit banks, of which up to 54 bits can be used with the PS I/O (one bank of 32b and one bank of 22b) and up to 64 bits (up to two banks of 32b) connected to the Programmable Logic
- Up to 54 flexible multiplexed I/O (MIO) for peripherial pin assignments

Interconnect

- High-bandwidth connectivity within PS and PL
- ARM AMBA AXI based
- QoS support on critical masters for latency and bandwidth control

Programmable Logic PL

Configurable Logic Blocks (CLB)

- Look-up tables (LUT)
- Flip-flops
- Cascadeable adders

36 kb Block RAM

- True Dual-Port
- Up to 72 bits wide
- Configurable as dual 18 kb block RAM

2.2 Board Peripherials

Based on the peripherials available on the Zynq XC7Z010, as well as the peripherials included in the board itself, a careful analysis needs to take place, given the fact that some of the peripherials in the SoC are not physicaly usable through the board, or that their functionality is already compromised for a specific purpose.

The case that is more relevant for this work, is the Ethernet controller available in the chip, that cannot be used due to the fact that the manufacturer uses a USB-to-Ethernet hub (See figure 2.2, module right to the MIO7 connection). This means that for the development point of view, any Ethernet programming need to be done for the USB module, not the Ethernet one. Other limitations appear for any development, that need to be considered for future work.

Chapter3

Vivado Design Suite

The Vivado Design Suite is the IDE provided by Xilinx Inc. for the design and development with their FPGA products and solutions. As part of the tool, a IDE for software development called Xilinx SDK is also provided. A third tool bundle for embedded linux image building is available independently in Xilinx web site, but that keeps strong relation to the Vivado program.

3.1 Installation

The main procedure for the installation of the Xilinx development platform is described in the UG973[1]. The tools are available for Windows, as well as most GNU-Linux OS. For the scope of this project, it is highly recommended to use Linux, considering that the hardware requirements are considerable high for a standar desktop PC, and also the image building tool runs only on Linux.

The Vivado installer can be downloaded directly from the Xilinx downloads page. This web installer includes the Vivado Design Suite for Hardware Description and the Xilinx Software Development Kit (SDK) for software (standalone and embedded) development. The installer will trigger the download of the necessary files on the web, which can vary from 5 to 7 GB depending on the installation options.

In order to install the Vivado Suite on a secure location (under linux file systems the folder `/opt` is recommended), the installer should be executed with root privileges, despite the fact that the official documentation claims that it can be run without them. The linux installer also requires that the USB cable drivers are installed after the Suite installation (not necessary in Wondows, as this installer is executed automatically). The following script can be used as a guide for the installation:

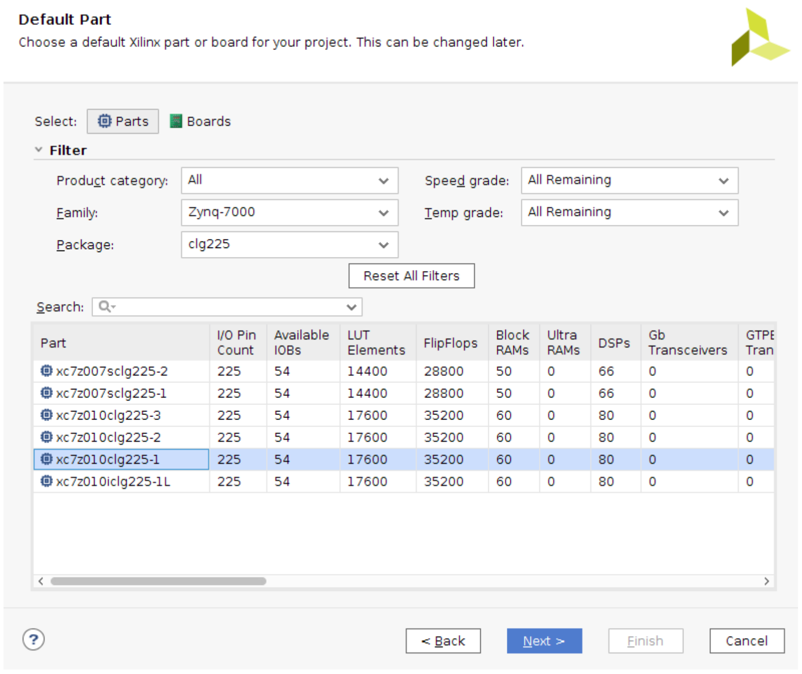
```
# Give executable permission to the installer file
chmod +x <installer_filename>.bin
# Executes the installer with root privileges
# Select the WebPack edition - no cost
```

[1]Vivado Design Suite User Guide, Release Notes, Installation and Licensing v2017.4

```
# Include at least the SDK and the device Zynq in the content selection
# Select the installation folder
sudo ./<installer_filename>.bin
# After install, change to the driver cable installer folder
cd <vivado_folder>/data/xicom/cable_drivers/lin64/install_script/install_drivers/
# Execute the driver cable installer with root privileges
sudo ./install_drivers
# Create the apropiate environment for the suite to be launched
# Needs to be run everytime before launching Vivado
# Alternatively, can be added to the .bashrc to be executed automatically
source _vivado_installed_folder_/settings64.sh
# Launch the Vivado GUI, blocks the terminal when launched
vivado &
```

3.2 Project Configuration

Once the Vivado Design Suite is installed and running, the hardware project can be created. The GUI launches a wizard that allows to select the folder for the project (this folder will also contain the software projects from SDK), the type of Xilinx project (RTL) and the device (as shown in the figure 3.1).



Default Part
Choose a default Xilinx part or board for your project. This can be changed later.

Product category: All
Family: Zynq-7000
Package: clg225
Speed grade: All Remaining
Temp grade: All Remaining
Search:

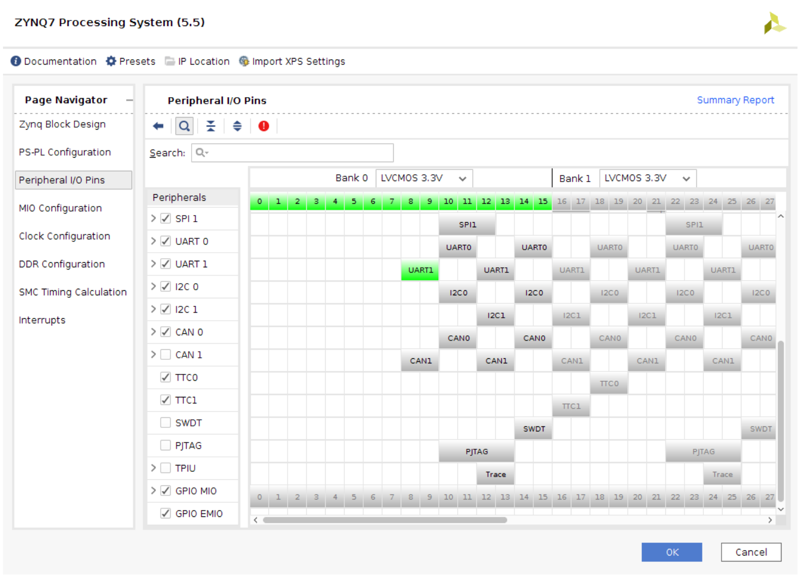
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE Tran
xc7z007sclg225-2	225	54	14400	28800	50	0	66	0	0
xc7z007sclg225-1	225	54	14400	28800	50	0	66	0	0
xc7z010clg225-3	225	54	17600	35200	60	0	80	0	0
xc7z010clg225-2	225	54	17600	35200	60	0	80	0	0
xc7z010clg225-1	225	54	17600	35200	60	0	80	0	0
xc7z010iclg225-1L	225	54	17600	35200	60	0	80	0	0

Figure 3.1: Device selection - Xilinx Hardware Project.

The hardware definition for the Zynq-7000 family implies the use of an IP Core design provided by Xilinx, in order to use the embedded ARM Cortex processor. Additional IP Cores can be added to the design in the Programmable Logic section, or user-defined hardware definition modules can be included.

The procedure for the hardware definition can vary depending on the specific hardware, and the requirements that the final application presents. A general procedure can be found in the User Guide UG1165[6].

1. Create a new Block Design.
2. Add the Zynq7 Processing System IP Core to the Block Design.
3. Add/create all the necessary HDL modules or IP Cores [Optional].
4. Configure the Zynq7 PS Core:
    (a) Apply the configuration in a TCL script file (provided by the manufacturer), or manually configure the Core in the GUI.
    (b) Ensure that the necessary peripherals are enabled (CAN, USB).



Bank 0 LVCMOS 3.3V
Bank 1 LVCMOS 3.3V

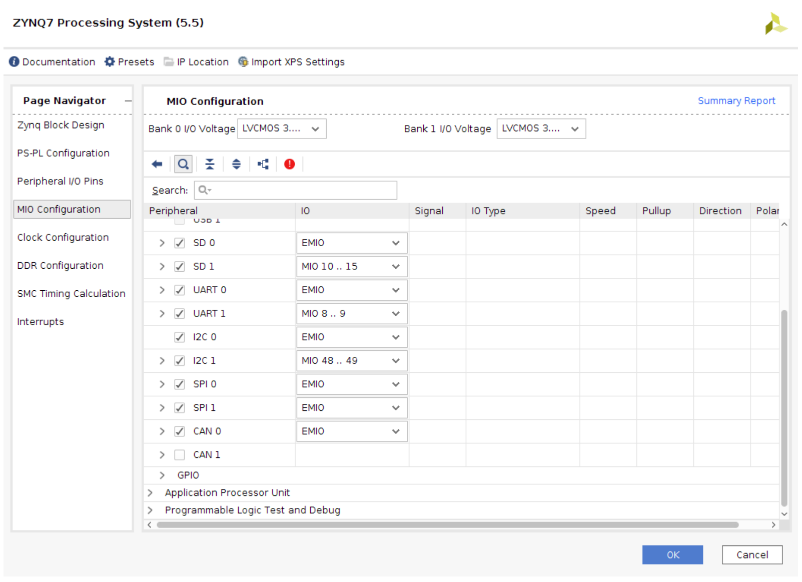
- ✓ SPI 1
- ✓ UART 0
- ✓ UART 1
- ✓ I2C 0
- ✓ I2C 1
- ✓ CAN 0
- ☐ CAN 1
- ✓ TTC0
- ✓ TTC1
- ☐ SWDT
- ☐ PJTAG
- ☐ TPIU
- ✓ GPIO MIO
- ✓ GPIO EMIO

Figure 3.2: CAN peripheral enable.

    (c) Ensure that the modules that are enabled (not by default) are routed to an available MIO pin (see Trenz TRM [7]), or via EMIO.
    (d) Configure the frequency for the CAN REF CLK signal (80MHz)[2].

5. Run the Block Automation wizard.
6. Run the Connection Automation wizard.
7. Config all the signals that will be routed to a physical pin external (Make external).
8. Generate a HDL wrapper for the Block Design.
9. Open the Elaborated Design (RTL Analysis), and set the I/O planning layout.

[2]The default value is 100Mhz, but in lab tests the CAN timing was imperfect with this setting, while 80Mhz produce an exact time quanta for the 500 kbps CAN network.



ZYNQ7 Processing System (5.5)

Documentation · Presets · IP Location · Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

MIO Configuration — Summary Report

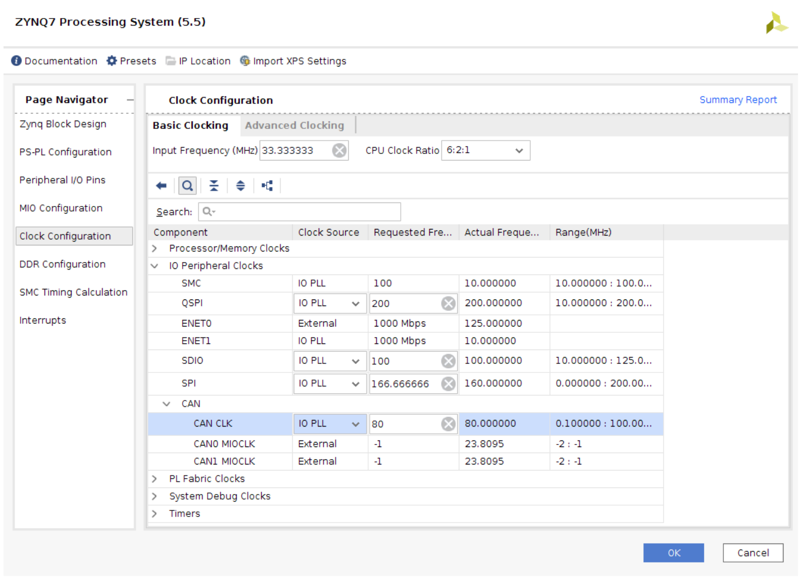
Bank 0 I/O Voltage: LVCMOS 3.... — Bank 1 I/O Voltage: LVCMOS 3....

Search:

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polar
USB 1							
✓ SD 0	EMIO						
✓ SD 1	MIO 10 .. 15						
✓ UART 0	EMIO						
✓ UART 1	MIO 8 .. 9						
✓ I2C 0	EMIO						
✓ I2C 1	MIO 48 .. 49						
✓ SPI 0	EMIO						
✓ SPI 1	EMIO						
✓ CAN 0	EMIO						
CAN 1							
GPIO							
Application Processor Unit							
Programmable Logic Test and Debug							

OK · Cancel

Figure 3.3: CAN signals routing through EMIO.



ZYNQ7 Processing System (5.5)

Documentation · Presets · IP Location · Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

Clock Configuration — Summary Report

Basic Clocking | Advanced Clocking

Input Frequency (MHz): 33.333333 — CPU Clock Ratio: 6:2:1

Search:

Component	Clock Source	Requested Fre...	Actual Freque...	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.0...
QSPI	IO PLL	200	200.000000	10.000000 : 200.0...
ENET0	External	1000 Mbps	125.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	100	100.000000	10.000000 : 125.0...
SPI	IO PLL	166.666666	160.000000	0.000000 : 200.00...
CAN				
CAN CLK	IO PLL	80	80.000000	0.100000 : 100.00...
CAN0 MIOCLK	External	-1	23.8095	-2 : -1
CAN1 MIOCLK	External	-1	23.8095	-2 : -1
PL Fabric Clocks				
System Debug Clocks				
Timers				

OK · Cancel

Figure 3.4: CAN module reference clock configuration.

10. Config the IO standard (LVCMOS33 for the Zynqberry board) and package pin for the signals made external (see Trenz TRM [7]).

11. Run Synthesis, Implementation and Generate Bitstream.

12. Export hardware, including bitstream file.

13. Launch SDK, selecting the local project environment.

At this point, the SDK will be started and automatically imported the hardware

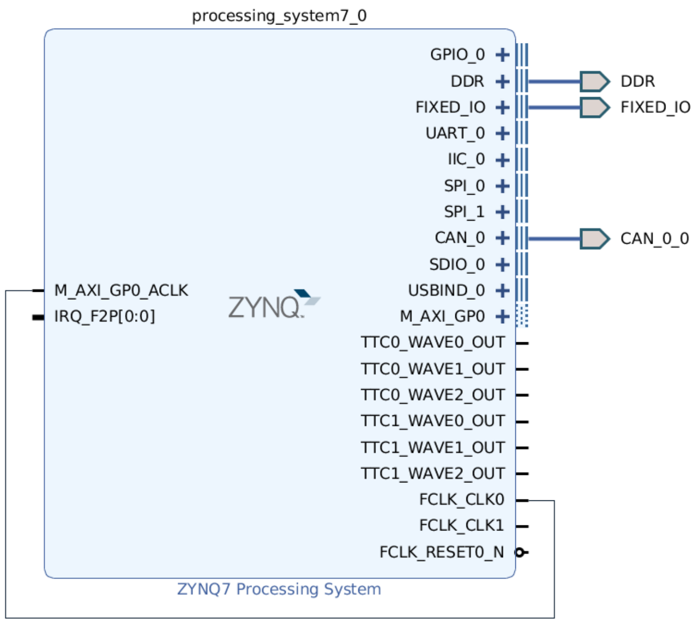


Figure 3.5: IP Core design block ready to sinthetize.

definition file. From this point, the software can be designed for this specific implementation.

The Vivado tool is also capable of command line scripts in TCL scripting language. This method can be a quite more complex to follow than the GUI, but it also increases the certainty in the procedure, as well as the ease of share, debug or collaborate (as a text file, can be easily pushed into a Source Control repository). The repository contains a file `create_project.tcl` (snippet shown partially), that can recreate the workflow above defined if run from the Xilinx Vivado TCL console [8].

```
create_project test /home/javier/Documents/fh/sem3/daebot/proj/test -part
    xc7z010clg225-1

create_bd_design "design_1"
update_compile_order -fileset sources_1

startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
    processing_system7_0
endgroup

apply_bd_automation -rule xilinx.com:bd_rule:processing_system7 -config {
    make_external "FIXED_IO,␣DDR" Master "Disable" Slave "Disable"} [get_bd_cells
    processing_system7_0]
```

3.3 IDiAL Server

In order to keep the environment available for further development, a Virtual Machine has been created in the IDiAL Server from FB4, with the following characteristics:

- VM: U16x64D_Vivado
- User: Javier
- Password: IDiAL
- OS: Ubuntu 16.04.3 Desktop
- IP-Address: 172.22.167.120

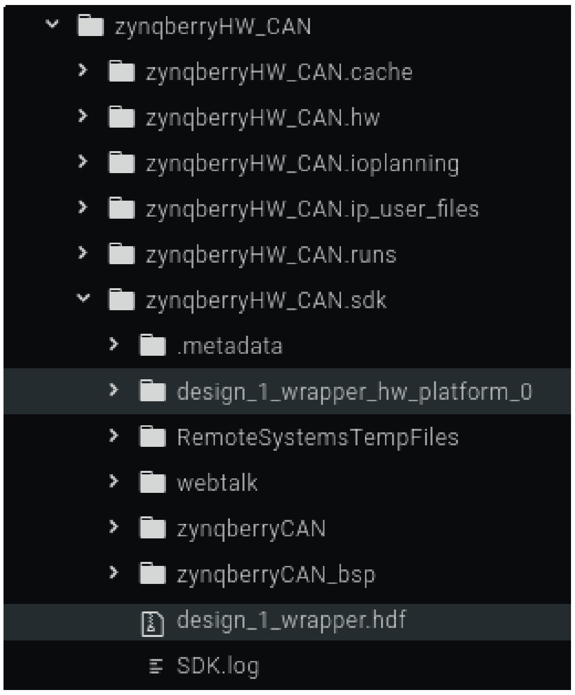
On this machine, the Vivado Design Suite was installed, as well as the Petalinux tool, in order fully handle the Zynqberry device. In order to access it, the VMware client can be used in Windows, or using the `vpnc` package in Linux. It can only be accessed from the FB4 network directly, or via the FB4 VPN[3]

[3]Connection details under https://www.fh-dortmund.de/de/fb/4/103020100000253634.php

Chapter4

Application

Once the hardware has been defined and exported in Vivado Design Suite, the provided Xilinx SDK environment can be launched from the Vivado GUI itself, or via the binary installed during Vivado installation. The SDK is based on the open-source Eclipse software core, which is well known for its workspace focus. This way, when SDK is launched from Vivado, it is automatically associated with the hardware project exported, which is part of the global project that Vivado created in the first steps of the hardware definition.



```
zynqberryHW_CAN
  zynqberryHW_CAN.cache
  zynqberryHW_CAN.hw
  zynqberryHW_CAN.ioplanning
  zynqberryHW_CAN.ip_user_files
  zynqberryHW_CAN.runs
  zynqberryHW_CAN.sdk
    .metadata
    design_1_wrapper_hw_platform_0
    RemoteSystemsTempFiles
    webtalk
    zynqberryCAN
    zynqberryCAN_bsp
    design_1_wrapper.hdf
    SDK.log
```

Figure 4.1: Xilinx project structure - Hardware definition exported to SDK folder.

As shown in picture 4.1, the SDK folder resides inside the global project created by Vivado, and the hardware export process defines both a file `.hdf` and a folder with all the correspondent information into the SDK folder (which is to be recognized as the workspace for the SDK software). Special care need to be taken with this project structure, to avoid folder or file duplicates when a change is made in the hardware definition after the software development has been started in SDK (it is possible that the hardware gets duplicated into the SDK folder, and the software definitions are not correctly updated into the source files).

4.1 Hello World application

In order to create a software project in the Xilinx SDK environment,

Once the SDK is ready (hardware is correctly imported), two different items need to be created in the SDK:

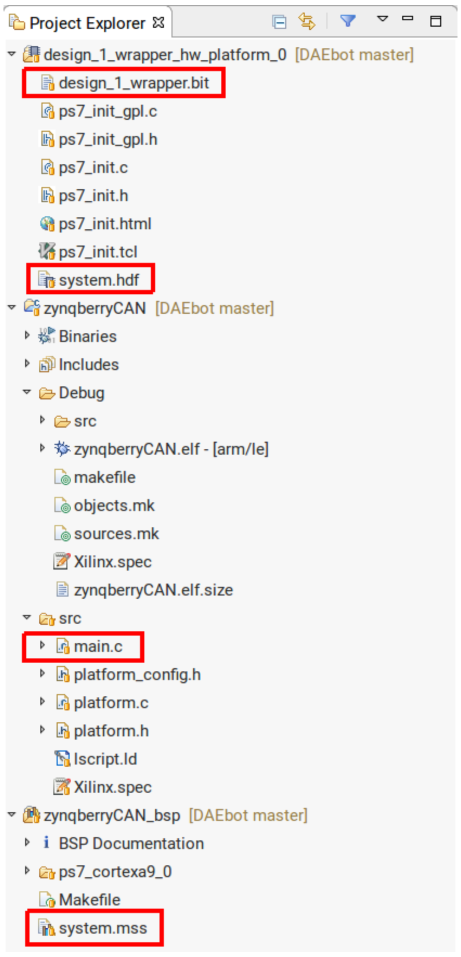
- A Board Support Package (BSP), which contains the libraries and source files for the different modules defined in the hardware definition, correctly compiled and ready to use in a software project.
- A software project itself, that will use the BSP and the hardware definitions in order to create an application (either standalone, or in an embedded linux OS).

The application can be a C or C++ language-coded program, through the estandar GNU compiler colection. The SDK creates all the necessary source files (including headers) for the selected template. For a first test of the platform, a Hello World template[**wiki**] is listed among the options. This template will provide a `helloworld.c` file, which can be renamed to a more convenient name (`main.c`).

The BSP need to be configured for the specific hardware platform (Zynberry board 0726), which provides a USB-to-serial converter, connected to the Zynq processor on the UART1. As the default UART defined in SDK is UART0, this setting needs to be changed. With this, the Zynq board will allow a serial communication with the host computer (via a serial console terminal program), with a baud speed of 115200 bps / 8N1 characteristics.

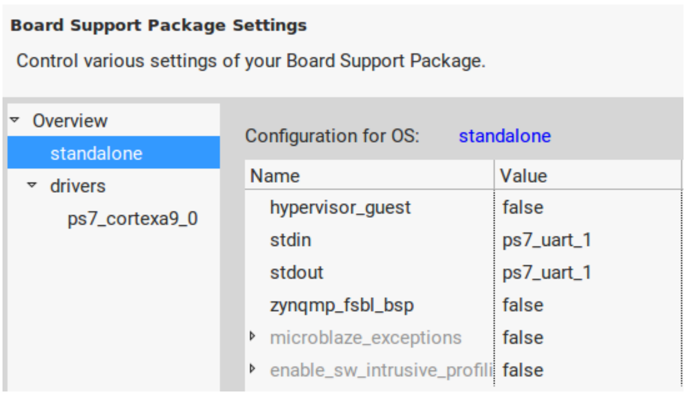
4.2 Sensor Communication

In order to provide communication with the devices in the DAEbot (via CAN bus), the application requires an appropiate driver for the CAN peripheral module in the processor. This driver is created by the SDK tool when the Board Suport Package is created from the hardware project. This created driver contains all the definitions, macros, data types and function prototypes for the correct handling of the CAN



```
Project Explorer
design_1_wrapper_hw_platform_0 [DAEbot master]
    design_1_wrapper.bit
    ps7_init_gpl.c
    ps7_init_gpl.h
    ps7_init.c
    ps7_init.h
    ps7_init.html
    ps7_init.tcl
    system.hdf
zynqberryCAN [DAEbot master]
    Binaries
    Includes
    Debug
        src
        zynqberryCAN.elf - [arm/le]
        makefile
        objects.mk
        sources.mk
        Xilinx.spec
        zynqberryCAN.elf.size
    src
        main.c
        platform_config.h
        platform.c
        platform.h
        lscript.ld
        Xilinx.spec
zynqberryCAN_bsp [DAEbot master]
    BSP Documentation
    ps7_cortexa9_0
    Makefile
    system.mss
```

Figure 4.2: Xilinx SDK project structure - Relevant files.



Board Support Package Settings

Control various settings of your Board Support Package.

```
Overview
    standalone
    drivers
        ps7_cortexa9_0
```

Configuration for OS: standalone

Name	Value
hypervisor_guest	false
stdin	ps7_uart_1
stdout	ps7_uart_1
zynqmp_fsbl_bsp	false
microblaze_exceptions	false
enable_sw_intrusive_profili	false

Figure 4.3: BSP configuration for STDIN and STDOUT.

module. The documentation of this peripheral driver (as well as all the other PS peripherals found in the hardware definition) are linked in the `system.mss` file, inside the BSP folder.

The CAN functionality provided by the module is quite standard (defined in [9]), so that it can be used with a rather simple code. In essence, there are 3 relevant functions coded for the CAN driver:

- canConfig - A subroutine that configures the functionality and baud rate of the module.
- sendFrame - A subroutine that triggers a CAN frame sending from a data buffer.
- recvFrame - A subroutine that triggers the receiving of a CAN frame into a buffer.

This functions contain the simple, register level functions provided by the BSP CAN driver, correctly adapted to the required functionality. Additional functionality can be added to the driver in the application.

The baud rate configuration for the Xilinx peripheral module works (in essence) similarly to the oficial specification. The baud rate configuration needed to achieve the established 500 kbps speed on the robot network, requires calculation an adjustment of the Time Quanta factor, and subsequently the Baud Rate Prescaler BRP, Synchronization Jump Width SJW, Time Segment 1 and Time Segment 2 (TS1-TS2). For this purposes, the equation for the registers are provided [5, p. 580]:

$$freqBit\_Rate = \frac{freqCAN\_REF\_CLK}{(BAUD\_RATE\_PRESCALER + 1) \cdot (3 + TS1 + TS2)} \tag{4.1}$$

In order to ease the configuration, a convenient tool is provided online[1], that helps with the calculation for the register values of different CAN modules available on the market. This was used given the fact that the recomended flow with the equations supose to asume a value and calculate the rest of them, which led to malfunctioning during the tests in the lab. With the tool, relevant design criteria were noted:

- Sample Point is 87.5% for CANopen and DeviceNet.
- SJW is tipically 1 for CANopen and DeviceNet.
- Bit time consisting of 16 Time Quanta is recommended.

The values proposed by the tool were tested, and proved succesful CAN communication, after several calculated combinations. For further changes in the DAEbot network speed, it is recomended to update the register values with help of this tool as well.

[1]See http://www.bittiming.can-wiki.info/#XCAN

```
 * Timing parameters to be set in the Bit Timing Register (BTR).
 * These values are for a 500 Kbps baudrate assuming the CAN input clock
 * frequency is 80 MHz.
 */
#define TEST_BTR_SYNCJUMPWIDTH    1
#define TEST_BTR_SECOND_TIMESEGMENT 1
#define TEST_BTR_FIRST_TIMESEGMENT  12

/*
 * The Baud rate Prescalar value in the Baud Rate Prescaler Register (BRPR)
 * needs to be set based on the input clock  frequency to the CAN core and
 * the desired CAN baud rate.
 * This value is for a 500 Kbps baudrate assuming the CAN input clock frequency
 * is 80 MHz.
 */
#define TEST_BRPR_BAUD_PRESCALAR  9
```

4.3 Final Application

The basic application can be represented with the activity diagram in figure 4.4:

For this application to work correctly, the three functions mentioned in section 4.2, which will use the low level driver functions generated by the BSP.

The communication between the host computer, the Zynberry device, and any of the devices in the CAN network, can be seen in the figure 4.5:

At the moment of the development, only a group of devices where enabled, but further devices can be easily included in the application, by verifying that the identifier (8-bit, defined in the table available in the Wiki) is correctly located in the data array `devices`, and adding the option in the menu.

```
/*
 * Make sure to align the position of the identifier with the menu option
 * (minus 1).
 */
static u8 devices[] = {0x1E,  // Gyroscope X
            0x1F, // Gyroscope Y
            0x20, // Gyroscope Z
            0x21, // Accelerometer X
            0x22, // Accelerometer Y
            0x23, // Accelerometer Z
            0x24, // Magnetometer X
            0x25, // Magnetometer Y
            0x26};  // Magnetometer Z
```

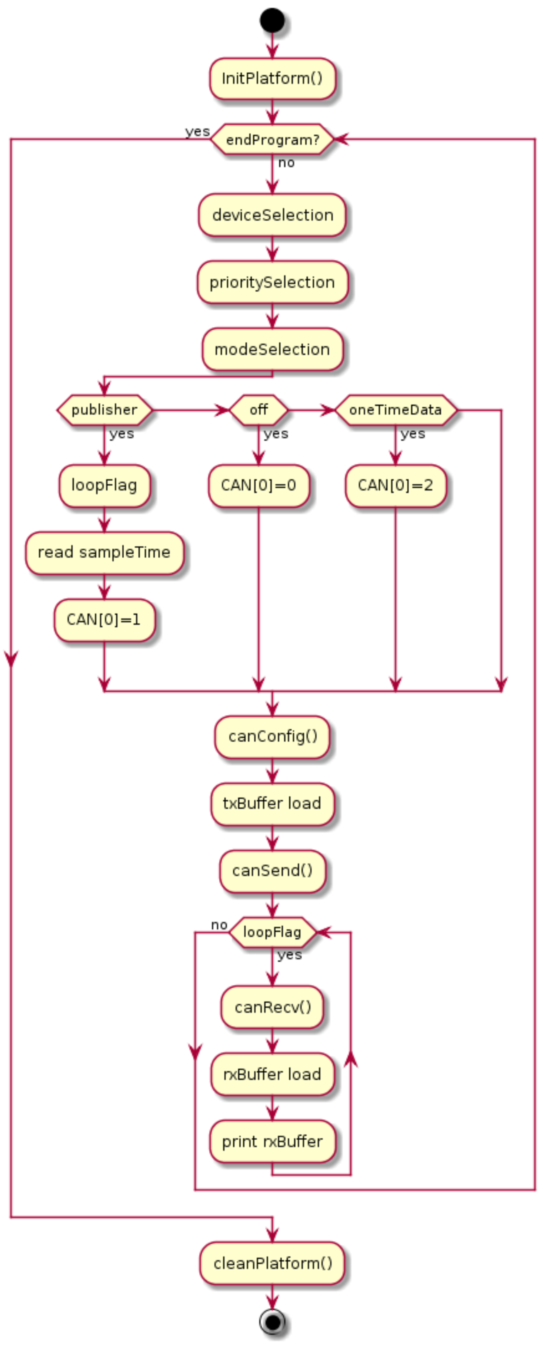


Figure 4.4: Activity diagram for the main application for CAN connectivity.

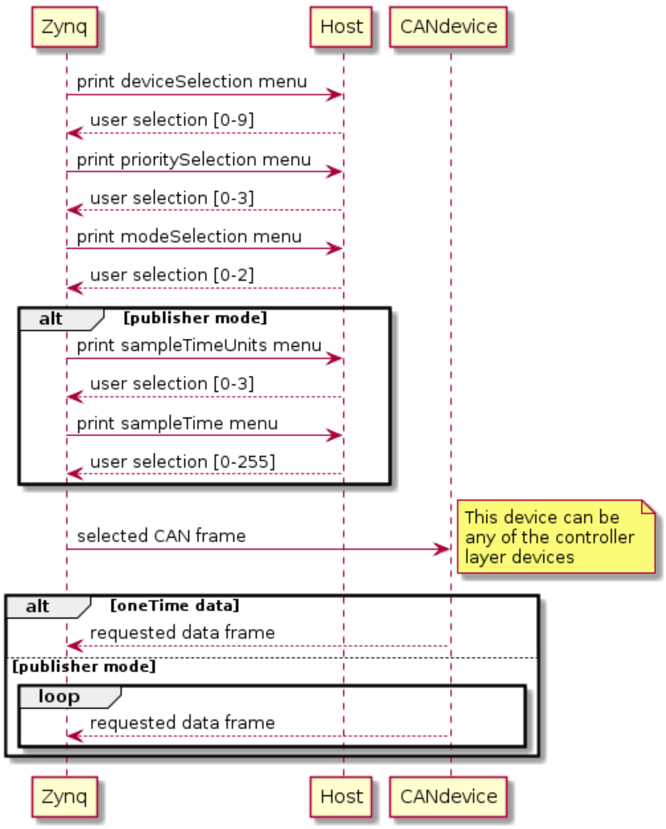


Figure 4.5: Sequence diagram for the main application.

Bibliography

[1] U. Lauschner. (2017). Daebot wiki, [Online]. Available: https://gitlab.idial.institute/DAEbot/DAEbot/wikis/home.

[2] J. Lückel, T. Hestermeyer, and X. Liu-Henke, "Generalization of the cascade principle in view of a structured form of mechatronic systems," *IEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings*, 2001.

[3] T. Electronic. (2017). Zynqberry - zynq-7010 in raspberry pi form factor, [Online]. Available: https://shop.trenz-electronic.de/en/TE0726-03M-ZynqBerry-Zynq-7010-in-Raspberry-Pi-form-factor?c=350.

[4] Xilinx, *Zynq-7000 all programmable soc data sheet: Overview*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[5] ——, *Zynq-7000 all programmable soc: Technical reference manual*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

[6] ——, *Zynq-7000 all programmable soc: Embedded design tutorial. a hands-on guide to effective embedded system design*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1165-zynq-embedded-design-tutorial.pdf.

[7] T. Electronic. (2017). Te0726 technical reference manual, [Online]. Available: https://wiki.trenz-electronic.de/display/PD/TE0726+TRM.

[8] Xilinx, *Vivado design suite tcl: Command reference guide*, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug835-vivado-tcl-commands.pdf.

[9] R. B. GmbH, *Can specification 2.0*, 1991. [Online]. Available: http://esd.cs.ucr.edu/webres/can20.pdf.

[10] T. Roorda. (2017). Getting started with the zynqberry, [Online]. Available: https://eewiki.net/display/Motley/Getting+Started+with+the+ZynqBerry.

[11] U. Lauchner, B. Igel, L. Krawczyk, and C. Wolff, "Applying model-based principles on a distributed robotic system appplication," *The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 2015.

[12] C. Wolff, P. Schulz, M. Knirr, K.-P. Priebe, and J. V. J. Strumberg, “Flexible and controllable orc turbine for smart energy systems,” *IEEE International Energy Conference (ENERGYCON)*, 2016.

[13] Xilinx, *Vivado design suite user guide, release notes, installation and licensing*, 2017. [Online]. Available: `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug973-vivado-release-notes-install-license.pdf`.