Chapter 1

Timebox 1

1.0.1 Outline for this timebox

1.0.2 Development Plan

1.0.3 Results

1.0.3.1 Design VHDL for Slave 1 & 2 (Anders)

Requirements to fulfil

Theory

Implementation

verification of requirements

1.0.3.2 Design & implement code for Communicator (Henrik)

Requirements to fulfil

Theory

Implementation

verification of requirements

1.0.3.3 Implementation of the wishbone interface (Jacob)

Requirements to fulfil FR.3.b.i

Theory

Pin Budget There is of cause a limited amount of GPIO pins available at the FPGA board, so a detailed pin budget will be made, to clarify how many pins are needed and which function each pin will have. This is essential to design the modules inside the FPGA, or at least to make them work prober without much confusion.

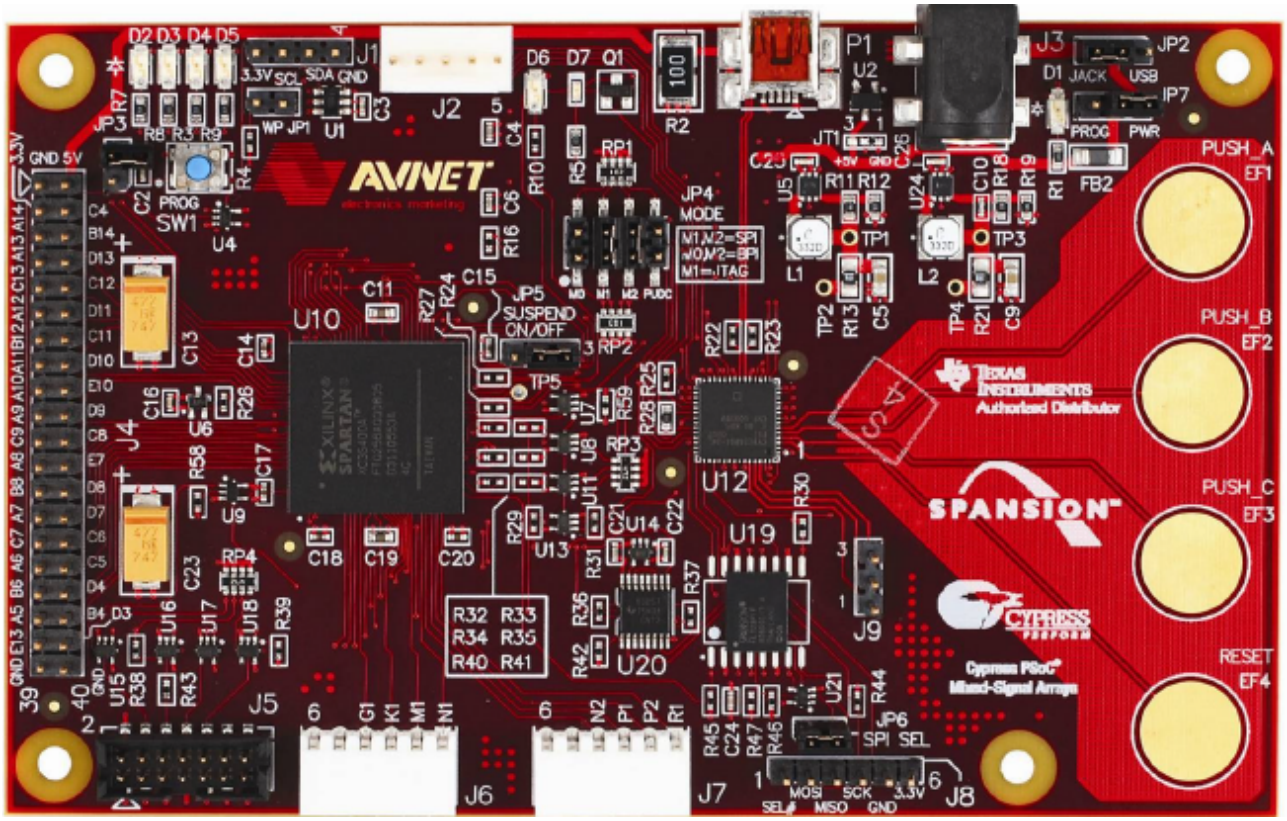

Figure 1.1: Spartan 3A board

Figure 1.1 shows the layout of the Spartan 3A board. The pin connectors J4, J6 and J7 will be used. For the CPU interface, only J4 will be used.

Table 1.1: Data Pins

FPGA PIN #	FPGA PIN NAME	LPC PIN#	LPC PIN NAME	FUNCTION
J4 4	C4	J2 60	BD0	DATA_PIN0
J4 5	A14	J2 58	BD2	DATA_PIN2
J4 6	B14	J2 56	BD4	DATA_PIN4
J4 7	A13	J2 54	BD6	DATA_PIN6
J4 8	D13	J2 52	BD8	DATA_PIN8
J4 9	C13	J2 50	BD10	DATA_PIN10
J4 10	C12	J2 48	BD12	DATA_PIN12
J4 11	A12	J2 46	BD14	DATA_PIN14
J4 12	D11	J2 59	BD1	DATA_PIN1
J4 13	B12	J2 57	BD3	DATA_PIN3
J4 14	C11	J2 55	BD5	DATA_PIN5
J4 15	A11	J2 53	BD7	DATA_PIN7
J4 16	D10	J2 51	BD9	DATA_PIN9
J4 17	A10	J2 49	BD11	DATA_PIN11
J4 18	E10	J2 47	BD13	DATA_PIN13
J4 19	A9	J2 45	BD15	DATA_PIN15

Table 1.1 shows the data pins connections. Referring to FR 3.a.i & FR 3.b.i, 16 pins are reserved for the data transfer.

Table 1.2: Adress Pins

FPGA PIN #	FPGA PIN NAME	LPC PIN#	LPC PIN NAME	FUNCTION
J4 20	D9	J2 42	BA0	ADDR_PIN0
J4 21	C9	J2 40	BA2	ADDR_PIN2
J4 22	C8	J2 38	BA4	ADDR_PIN4
J4 23	A8	J2 36	BA6	ADDR_PIN6
J4 24	E7	J2 34	BA8	ADDR_PIN8
J4 25	B8	J2 41	BA1	ADDR_PIN1
J4 26	D8	J2 39	BA3	ADDR_PIN3
J4 27	A7	J2 37	BA5	ADDR_PIN5
J4 28	D7	J2 35	BA7	ADDR_PIN7
J4 29	C7	J2 33	BA9	ADDR_PIN9

Table 1.2 shows the address pins used. 10 pins are reserved for address.

Table 1.3: Motor Control Pins

FPGA PIN #	FPGA PIN NAME	FUNCTION
J4 31	A6	M_CTRL4
J4 32	C5	M_CTRL3
J4 33	B6	M_CTRL2
J4 34	D4	M_CTRL1

Table 1.3 shows the four pins reserved for motor control. 4 output signals to two H-bridges, to control the motors position. One pin for each direction the solar panel has to move; up, down, left right.

Table 1.4: LPC/FPGA Control Pins

FPGA PIN #	FPGA PIN NAME	LPC PIN#	LPC PIN NAME	FUNCTION
J4 35	A5	J1 45	2.14	Chip Select
J4 36	B4	J1 6	Reset_out	Reset
J4 37	E13	J1 35	BWE	Write Enable
J4 38	D3	J1 36	BOE	Output Enable

Table 1.4 shows the chip select, which has to be high for the FPGA to react on commands from the LPC, Reset for a synchronous reset of both boards. When reset is pressed on the LPC board, Reset_out will be set low, which the FPGA will react on. Also if the power is lost, the pin will go low, and secure a reset/idle state, until the power is turned on again. Also the Write and Read enable are described, for the LPC to control whether the FPGA has to send or receive data from the LPC.

Table 1.5: LPC Control Pins

Connection	LPC PIN#	LPC PIN NAME	FUNCTION
Chip Select	J2 43	DBUS_EN	Data Bus Enable
Ground	J2 44	ABUF_EN	Address Buffer Enable

Table 1.5 shows that DBUS_EN is connected to Chip Select. This is enabled when chip select is enabled. This data bus is controlled by the WE and OE signals, and can act both as input and output. This should not be left low, else it will collide with the boards internal data bus, therefore it is connected to Chip Select, and is only low when chip select is enabled. DBUS_EN is active low.

ABUF_EN is grounded, to pull it low constantly. This will enable the two buffers for address. Also the control signals are enabled, and act as output. [1]

Table 1.6: Others

FPGA PIN #	FPGA PIN NAME	FUNCTION
J4 1	GND	GND
J4 2	+5V	+5V SUPPLY
J4 3	+3.3V	+3.3 SUPPLY
J4 30	C6	FREE
J4 39	GND	GND
J4 40	GND	GND

A lot of text, to describe the table of data!

State Machine Diagram (Jacob) Before implementing the VHDL code, a state machine diagram is made to clarify which states the module has to go through. When this is in place, the code is very easy to implement.

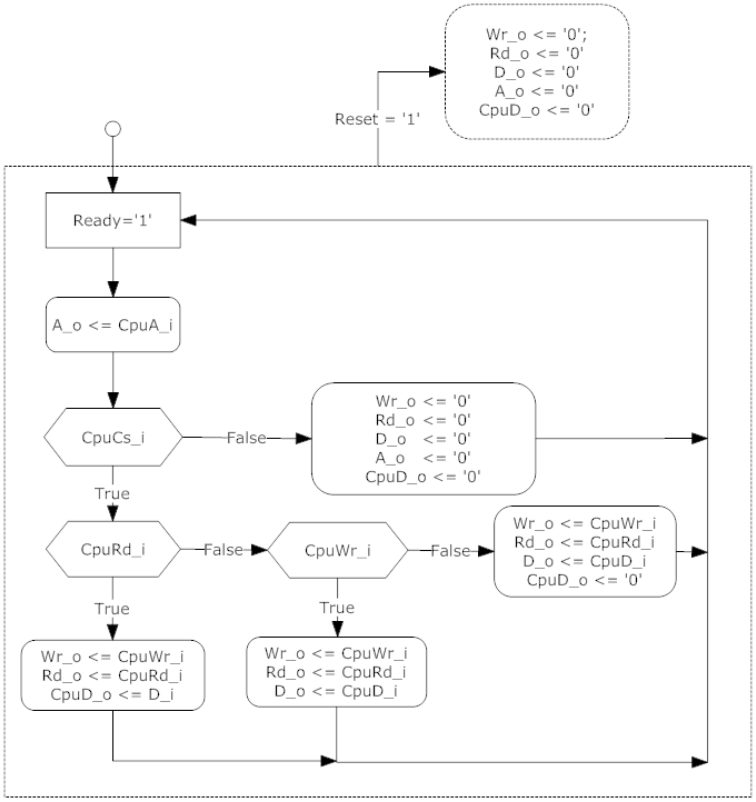

Figure 1.2: State Machine Diagram - CpuInterface

[1] LPC2478_OEM_Board_Users_Guide_Rev_H, page 10, chap 3.1.6

Figure 1.2 shows the State MAchine Diagram for the CpuInterface. The different states will be discussed in the "implementation" chapter below.

Implementation
Important thing first. The module has to check if the reset pin is enabled. If this is the case, it has to reset the board, no matter what. This is why it is set asynchronious. No clock has to be on its rising or falling edge. The reset values is all 0, to be sure no data is stored, no adress is stored and no write/read commands are given to the Wishbone Master.

```
-- Check if reset=high (Unsynchronized)

-- Reset all values to '0'

if(Rst='1') then

Wr_o <= '0';

Rd_o <= '0';

D_o <= (others => '0');

A_o <= (others => '0');

CpuD_o <= (others => '0');
```

If reset is not set, the module has to think syncronous. This is why we check the clock. Both the read and write state is inside the synchronous clock state, and also the chip select (CpuCs_i) has to be high, in order for the module to respond on read/write commands. When chip select is set high, the adress is sent from the Cpu to the Wishbone Master. The adress can always be set, without any failures occur.
Then it is time to check if the "Read" is set. If it is, then the CPU wants to read data from the FPGA, thats why the data from the Wishbone master (D_i) is sent to the output to the cpu (CpuD_o).
Also the Read and Write input from the cpu is sent to the wishbone master. Then the cpu will have to avoid both being high at the same time.

```
elsif(Clk'event and Clk = '1') then --(All sync)

-- Check if active read (CPU read ffrom FPGA)

if(CpuCs_i = '1') then

A_o <= CpuA_i; -- Common for all

if(CpuRd_i = '1') then

Wr_o <= CpuWr_i;  -- Enables/Disables master write
```

```
Rd_o <= CpuRd_i;  -- Enables/Disables master read

CpuD_o <= D_i; -- Data from master (D_i) to CPU (CpuD_o)
```

If read is not high, the write pin has to be checked. If that is enabled, the CpuInterface will send the data recieved from the Cpu to the Wishbone master. Again both read and write is rooted directly to the Wishbone Master.

```
-- Check if active write (CPU write to FPGA)

elsif(CpuWr_i = '1') then

Wr_o <= CpuWr_i;

Rd_o <= CpuRd_i;

D_o <= CpuD_i; -- Data from CPU (CpuD_i) to Master (D_o)

else
```

If either read nor write is high, the Cpuinterface will reset the Cpu data out to all "0". Also the read and write are directly rooted to the wishbone master, which also is "0". This will keep the system from making any action when not supposed to.

```
-- Reset values to 0

Wr_o <= CpuWr_i;

Rd_o <= CpuRd_i;

D_o <= CpuD_i;

CpuD_o <= (others => '0');

end if;
```

Last but not least, if chip select is not enabled, all values will be reset, as iff the reset button was pressed. This is to avoid any failures and unwanted actions in the system.

```
-- If Chip not selected (CpuCs_i = '0'), reset all values to '0'.

else

Wr_o <= '0';

Rd_o  <= '0';
```

```
D_o <= (others => '0');

A_o <= (others => '0');

CpuD_o <= (others => '0');

end if;

end if;
end process Cpuinter;
```

verification of requirements

1.0.3.4 Module Design (Jacob)

Requirements to fulfil
NR.4.b

Theory
The module design shall help us clarify which blocks shall be implemented where. All the hardware, the Spartan 3A board, the LPC2478 and other modules, shall be drawn an connected. This clarifies what the different hardware contains and how they interface with each other. The module design is shown in Figure 1.3

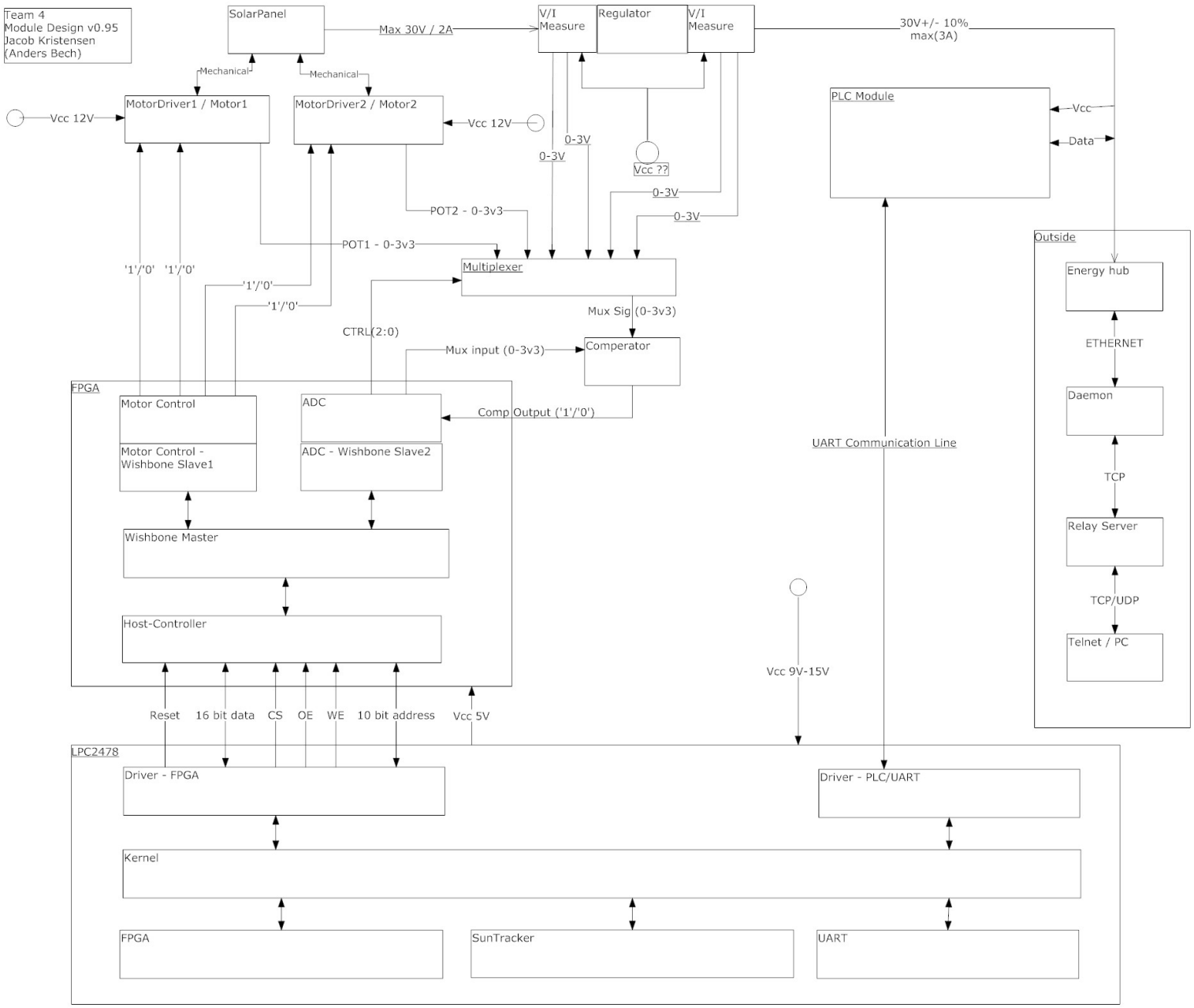

Figure 1.3: Module Design of "system-to-be"

The **solar panel** is physically connected to the two motors, which will turn the panel towards the sun. **The two motors** are each supplied by 12 V DC (Source unknown). Four signals are received from the **FPGA**, that controls which way the motors have to turn. Those signals are sent to two H-bridges, which will invert the voltage, and make the motors turn in different directions. The four signals are controlled by a **PWM** block in the FPGA, to make speed control of the motors possible.
The potentiometers of the motors will return a value to the **ADC** implemented in the FPGA, which then can compare their position, to the position of the sun, received from the LPC2478. The ADC input limit, on the FPGA board is 3v3[2]
The ADC and the PWM are implemented in a wishbone structure, which makes it possible to use it at different platforms, if the wishbone interfaces are obeyed. The two wishbone slaved (PWM, ADC), are interfaced to the LPC2478 through a wishbone master, controlling the communication, and then through a host-controller, onto the FPGA-driver implemented on the LPC2478 board.
The SPARTAN 3A board and the LPC2478 developers kit is a requirement to this project, given by the teachers. The **FPGA** board is software coded hardware, which means it makes most sense to replace some actual hardware by this development board, or maybe make a module more efficient, by replacing some of the software by this "hardware" board. In this case it is replaced by both. The suntracker was meant as pure software, but now a formula, calculating the position of the sun, is implemented as software, and the position of the sun is sent to the FPGA; which compares it to the position of the motors, received by the potentiometers, and then repositioning the motors, according to the sun.
The **LPC2478** board is where the software is placed, and are an essential part of the system. This is used for communication, logging and other things, which will be described later. The board is controlled by a small embedded Linux system. Three drivers are implemented, and they are controlled by the kernel, on behalf of three user applications. This board is supplied with a voltage between 9-15V DC[3].
The first interface was the interface for communication with the FPGA, shortly described above. This is a 16 bit communication line, which is also a requirement for the system. Further there is a chip select, read and write line and finally a supply line, supplying the FPGA with power from the LPC2478 board[4]. The FPGA user application will handle the information that is to be sent and received.
The power generated from the solar panel, will be sent through a **regulator**, which is pure hardware, where the power will be converted from approx 30V, into 30V+/-10%. This output is a common requirement, to make the system function with the other systems.
Two galvanic isolated **sensors** are applied to the input and the output of the regulator, to keep track of how much energy is produced and how efficient the regulator is. Those sensors sends a PWM signal (0-3v)[5] down to the ADC, in the sensor driver, in the LPC2478 board. The user application to the sensor driver will then pack the data and send them to the energy hub, which updates the web-page.
The last user application/driver at the LPC board is the **PLC driver**, which makes the communication to the energy hub possible. This communication is established through UART,

[2]FPGA datasheet page 3.
[3]LPC2478 print
[4]LPC2475 user manual, chap. 5 - External Memory Controller
[5]LPC2478 user manual, chap 28.2

which sends and receives messages between the system and the energy-hub. A **PLC module** is implemented too, which is supplied by the powerline and speaks through it too.

verification of requirements

content...