Connecting a DE2 board with a 5-6k interface board containing an ADC for digital data transmission

Bachelor thesis performed in Electronics Systems
by
Markus Keller
LiTH-ITY-EX-ET--11/0382--SE

Linköping 2011
Connecting a DE2 board with a 5-6k interface board containing an ADC for digital data transmission

Bachelor thesis in Electronics Systems at Linköping Institute of Technology
by

Markus Keller

LiTH-ISY-EX-ET--11/0382--SE

Supervisor: Kent Palmkvist
Examiner: Kent Palmkvist

Linköping 2011
Connecting a DE2 board with a 5-6k interface board containing an ADC for digital data transmission

Markus Keller

The goal of this bachelor thesis work was to establish a cable connection between an analogue interface board, containing a 16 bit analogue to digital converter, and a DE2 board in order to allow for digital data transmission between the two boards.

The DE2 board includes an FPGA which was configured to contain a Nios II softcore microprocessor for handling the tasks of reading and saving the 16 bit digital words transmitted over the cable as well as controlling the analogue to digital converter on the interface board.

During the project work various tasks had to be fulfilled which included soldering the cable for parallel transmission of the 16 bit digital data words and the control signals between the boards as well as adjusting the analogue interface board with the correct voltage supplies and jumper settings. Furthermore the hardware circuit inside the FPGA had to be configured and the program running on the Nios II processor had to be written in C language.

Nios II processor, DE2, Altera monitor program, ADC, 5-6k interface board
Abstract

The goal of this bachelor thesis work was to establish a cable connection between an analogue interface board, containing a 16 bit analogue to digital converter, and a DE2 board in order to allow for digital data transmission between the two boards.

The DE2 board includes an FPGA which was configured to contain a Nios II softcore microprocessor for handling the tasks of reading and saving the 16 bit digital words transmitted over the cable as well as controlling the analogue to digital converter on the interface board.

During the project work various tasks had to be fulfilled which included soldering the cable for parallel transmission of the 16 bit digital data words and the control signals between the boards as well as adjusting the analogue interface board with the correct voltage supplies and jumper settings. Furthermore the hardware circuit inside the FPGA had to be configured and the program running on the Nios II processor had to be written in C language.
## Contents

1- Introduction  
1.1 Project description  
1.2 Definition FPGA  

2- DE2 Development and Education board  
2.1 The hardware specification of the board  
2.2 The Quartus II software  
  2.2.1 Creating a project  
  2.2.2 Entering a schematic diagram  
  2.2.3 Compiling the designed circuit  
  2.2.4 Assigning the pins  
  2.2.5 Simulating the circuit  
  2.2.6 Programming the FPGA  
2.3 The Nios II processor and the SOPC-builder  
  2.3.1 Nios II processor introduction  
  2.3.2 Register structure  
  2.3.3 Creating Nios II systems with the SOPC Builder  
2.4 Altera monitor program  
  2.4.1 Configuring a Nios II system  
  2.4.2 Configuring and loading a Nios II program  
  2.4.3 Running and debugging programs  

3- 5-6K Interface Board  
3.1 Power connections to the interface board  
3.2 Signal conditioning sites  

4- Evaluation module: ADS1606  
4.1 Clock input  
4.2 Control signals and data retrieval
5- The design and realization of the project

5.1 Soldering the cable................................................................. 19
5.2 Configuring the hardware inside the FPGA ......................... 21
   5.2.1 Synchronization and data retrieval .................................. 21
   5.2.2 The Nios system............................................................ 22
   5.2.3 The C-program for the Nios II processor ......................... 25

6- Results, conclusion and future work

6.1 Results.............................................................................. 27
6.2 Future work....................................................................... 27

References.............................................................................. 29
1 – Introduction

1.1 Project description

The goal of the project was to connect a DE2 board [1] with a 5-6k analogue interface board [12], containing a 16 bit Analogue to Digital converter. The DE2 board contains an FPGA which can be programmed in such a way that it includes a “Nios II” softcore microprocessor. Analogue signals should be fed into the AD converter and the digital output data should be transmitted to the DE2 board and saved in the memory. To achieve this functionality the following steps had to be performed.

- The cable for connecting the 36 bit Expansion header of the DE2 board with the output pins of the AD converter as well as with the pins for controlling the AD converter had to be soldered.
- The circuit in the FPGA had to be designed and configured.
- The software for the Nios II microprocessor had to be written in C language.
- The 5-6 interface board had to be adjusted with the correct jumper settings and supply voltages.

In the following some basics about FPGAs in general shall be explained before the different hardware modules which were included in the project will be described.

1.2 Definition of FPGA

A field-programmable gate array (FPGA) is an integrated circuit, which means it is an electronic circuit of very small dimensions produced in a thin substrate of semiconductor materials. An FPGA can be configured after the manufacturing process by using for example a hardware description language, thus it is called “field programmable”. By its configuration various circuits of different complexity can be realized like simple counters or complex microprocessors [2].

A classic FPGA has got the following structure: It contains a field of many identical logic blocks, programmable interconnections and input and output cells. Each of the logic blocks has got n inputs and one output and usually consists of a D flip-flop and a look up table, which can be configured to perform complex combinational logic or simple gates like “and” or “xor” [2]. The Cyclone II FPGA on the DE2 board used in the project consists of about 33,000 of those logic blocks which have got four inputs each [3, page 6]. In addition to the three basic resources of any FPGA, which are the logic blocks, the input and output cells and the programmable interconnections, does the Cyclone II FPGA on the DE2 board also contain phase locked loops, DSP-blocks and memory blocks [6, page 10].

In the following the different hardware and software components included in the project shall be described before the actual design will be explained.
2 – DE2 Development and Education board

The DE2 is a so called education board which means it has been developed for teaching and evaluation purposes. The main function of the board is to offer a tool for people who want to improve their knowledge about digital logic, computer organization and FPGAs [1].

2.1 The hardware specifications of the board [1]

FPGA
- Cyclone II EP2C35F672C6 FPGA and EPCS16 serial configuration device

Input/Output Devices
- Built-in USB Blaster for FPGA configuration
- 10/100 Ethernet, RS-232, Infra red port
- Video Out (VGA 10-bit DAC)
- Video In (NTSC/PAL/Multi-format)
- USB 2.0 (type A and type B)
- PS/2 mouse or keyboard port
- Line-in, Line-out, microphone-in (24-bit audio CODEC)
- Expansion headers (76 signal pins)

Memory
- 8-MB SDRAM, 512-KB SRAM, 4-MB Flash
- SD memory card slot

Switches, LEDs, Displays, and Clocks
- 18 toggle switches
- 4 pushbutton switches
- 18 red LEDs, 9 green LEDs
- Eight 7-segment displays
- 16 x 2 LCD display
- 27-MHz and 50-MHz oscillators, external SMA clock input

Figure 1. DE2 board

The input and output cells of the FPGA chip are connected to all other components of the DE2 board like it can be seen in the following block diagram: [3, page 6]
In the final design for the project one of the two 36 bit expansion headers of the DE2 board was used for the connection with the analogue interface board. Various toggle switches were used to control the necessary control signals for the evaluation module on the 5-6k analogue interface-board.

2.2 The Quartus II software

As explained before is an FPGA programmable after the manufacturing process. The Quartus-II software allows the user of the DE2 board to define the circuit in the FPGA using either VHDL or Verilog hardware description language or a schematic design entry method, in which a graphical diagram of the circuit can be drawn [4, page 1]. Furthermore, the necessary pin assignments can be adjusted with the Quartus II software, to define in which way the circuit of the FPGA shall be connected to the other components on the board such as switches, LEDs or expansion headers. In the following we will briefly explain the different steps that have to be carried out for programming and configuring the FPGA on the DE2 board using the Quartus II software [4].

2.2.1 Creating a project

All logic circuits which are designed in the Quartus II software are called projects. When a new project is created by the user, the software generates a new folder with the name of the project in the file system. All information about the project including pin assignments or definition of the FPGA circuit is saved in this project directory. A new project can be designed with the help of a wizard. Important is here to specify the correct device name and family name of the FPGA. The FPGA on the DE2 board which was used in the project is called EP2C35F672C6 and is part of the cyclone II device family [4, page 7].

2.2.2 Entering a schematic diagram

As mentioned above, it is possible to describe the circuit for the FPGA either with a Hardware description language or with a graphical diagram. The tool for specifying the circuit in form of a block diagram is called Quartus II Graphic Editor. In figure 3 we can see a simple block diagram created with the Graphic Editor.

![Figure 3. Block diagram displayed in the Graphic Editor](image)

This simple example circuit consists only of one and gate, one inverter, two input pins and one output pin. The Graphic Editor supports much more components like other logic gates, storage elements, buffers or multiplexers.
2.2.3 Compiling the designed circuit

After the design of the block diagram has been finished, the circuit has to be compiled. During the compilation the Quartus II software executes the following steps: The block diagram file is analysed and investigated for possible errors. If the block diagram contains no errors, a circuit is synthesized from the diagram and afterwards an implementation of the circuit is generated for the FPGA. When the compilation is finished a compilation report appears, showing the possible errors in the block diagram. In case of no errors the message “compilation successful” appears and other information is given, for example how many logic elements or pins and logic elements of the FPGA are used to implement the designed circuit [4, page 14].

2.2.4 Assigning the pins

When the circuit has been successfully compiled, the pin assignments have to be made. This means that the user has to define how the inputs and outputs of the circuit inside the FPGA shall be connected to the other components of the DE2 board. For example could inputs to the FPGA circuit be connected to switches and outputs could be connected to LEDs so that the circuit functionality could be tested. The pin assignments are made by using the assignment editor [4, page 18]. The DE2 board has got hard-wired connections between the other components on the board and the FPGA pins. The user has to assign the names which he specified (in the block diagram) for the in and output pins of the FPGA circuit to the pins of the FPGA which are connected to the desired input and output components of the circuit. Therefore it is necessary to look up in a pin-table how the pins of the FPGA are named, which are connected to the various other components of the board.

Since the pin assignment can be a quite time consuming task, it is possible to save pin assignments and import them to other projects [4, page 19].

2.2.5 Simulating the circuit

After assigning the pins it is recommended that the circuit is simulated, before implementing it on the FPGA. The Quartus II software offers a waveform editor which allows the user to enter necessary information for the simulation in order to test the circuit functionality. The user can specify certain time durations for which the circuit shall be simulated and he can define how the input signals shall look during this time period. After the simulation the resulting output signals of the circuit can be observed. There are two different possibilities how the circuit can be simulated, the functional simulation and the timing simulation [4, page 24]. In the functional simulation all components of the circuit are considered to be ideal without any propagation delays. This allows the user to check if he has designed the circuit correctly regarding to its logic functionality. The timing simulation on the contrary takes all propagation delays of the elements in the FPGA into account and allows the user to observe how much the output signals of the circuit will be delayed compared to the input signals.

2.2.6 Programming the FPGA

After the circuit has been designed and simulated and the necessary pin assignments have been made, the FPGA can be programmed and configured to implement the circuit. To program the FPGA one USB connector of the DE2 board has to be connected over a cable
with an USB port of the host computer, where the Quartus II software is installed. For this purpose an USB-blaster driver has to be installed on the host computer. There are two different modes for programming the FPGA, the JTAG mode and the Active Serial mode [4, page 26]. If the FPGA is programmed in the JTAG mode, the configuration will only be saved as long as the DE2 board is powered on. If it is turned off the information of the FPGA configuration will be lost. If the FPGA is programmed in the Active Serial mode on the other hand, the configuration data is saved in a flash memory on the board and still remains after the board has been turned off. The DE2 board contains a switch which allows the user to choose in which of the two modes the FPGA shall be programmed. If the switch stands in the “RUN” position, the JTAG mode is chosen while the “PROG” position selects the Active Serial mode. After the FPGA has been programmed and configured the design is completed and the circuit functionality can be tested on the board.

2.3 The Nios II processor and the SOPC-builder

The Quartus II software offers the user the possibility to design embedded systems on the DE2 board containing a micro-controller. The embedded processor which has been developed for the Altera FPGAs is called Nios II processor. Nios II systems can be created by using a tool called SOPC builder in conjunction with the Quartus II software. In the following the Nios II processor and the SOPC builder shall be explained in detail.

2.3.1 Nios II processor introduction

The Nios II processor is a so called softcore processor [5, page 1]. A softcore processor is a microprocessor which is fully described in software, usually in a hardware description language. Therefore it can be realized in programmable hardware like for example FPGAs [6, page ii]. The Nios II processor furthermore has got an RISC architecture, which stands for reduced instruction set computer [5, page 3]. The goal in an RISC architecture design is to increase the performance of the processor by only supporting certain simplified instructions, which can be executed in relatively short time [7].

The Nios II processor can be implemented in the following three different configurations with varying performance and hardware costs [5, page 3].

- Nios II/f: The f stands for “fast” version. In this variant of the processor the most configuration options are available for the user, which offers a lot of possibilities to optimize the processor’s performance.

- Nios II/s: In this “standard” version the hardware costs inside the FPGA is reduced as well as the performance of the processor.

- Nios II/e: The e stands for “economic” version. This configuration offers the most limited amount of configuration options for the user but also requires the fewest hardware resources inside the FPGA. This version in total requires less than 700 logic elements inside the FPGA.

There are three different modes in which a Nios II processor may operate; the Supervisor mode, the User mode and the Debug mode [5, page 3]. After the processor has been reset it automatically enters the Supervisor mode in which all instructions and functions of the
processor can be used. In the User mode, some of those instructions and functions, which deal with system purposes, are not available. The Debug mode finally is used by software debugging tools, like for example the Altera monitor program. In this mode features like setting breakpoints or watch-points or manually starting and stopping the program execution are supported. [5, page 3]

2.3.2 Register structure

The Nios II processor has got 32 general purpose registers. All arithmetic and logic operations on operands are executed in those general purpose registers. The assembly instruction set of the processor includes load and store commands to transfer data between the general purpose registers and the memory. The word length of the Nios II processor and the length of all registers are 32 bits [5, page 3].

The 32 general purpose registers include the following special registers with specific purposes [5, page 4]. They have got special names, so that they are recognized by the assembler.

• Register r0: This register is called the zero register, because the constant value zero is saved in it at all time. Writing to this register has no effect.

• Register r1: This register is used as a temporary register by the Assembler. Thus it should not be referenced in user programs.

• Registers r24 and r29 are reserved to handle exceptions. They cannot be accessed when the processor is in the user mode.

• Register r25 and r30 are only used by the JTAG debug module.

• Register r27 and r28 are reserved for controlling the stack, which is used by the Nios II processor.

• Register r31 has got the purpose to save the return address if a subroutine is executed.

In addition to the general purpose registers there are six 32 bit control registers implemented in the Nios II processor [5, page 4]. They have got special names as well and can only be accessed by special assembly instructions. As their name implies those registers are used for control purposes. One of those registers stores the mode in which the processor is running and the information if external interrupts shall be accepted or not. Two other registers save copies of that information. One during exception processing and the other one during debug break processing. The forth of those control registers can be used to enable certain external interrupts while the fifth control register indicates which interrupt is being executed. The last control register finally saves a unique value for identifying the processor in a multi chore system.
2.3.3 Creating Nios II systems with the SOPC Builder

To implement a useful Nios II system it is required to add other functional components, except for the processor itself, like memories, timers or input and output interfaces [8, page 2]. An example for such a system can be observed in figure 4 [8, page 2]. We can see in the figure, that the Nios II processor and the interfaces which are needed for the connection with other components on the DE2 board are implemented inside the FPGA.

Those interfaces and the processor are connected over the Avalon switch fabric, which is an interconnection network that specifies the interface and protocol used between master and slave components [8, page 2]. The Avalon Switch fabric allows multiple masters and slaves on the bus simultaneously.

As discussed before, the Cyclone II FPGA chip contains memory cells, which can be used as on-chip memory for the Nios II processor. This on-chip memory can either be connected with the Nios II processor directly or through the Avalon switch fabric as in figure 4 [5, page 3]. The JTAG UART interface is needed for the USB connection with the host computer. The combination of this circuit and the associated software is called the USB-Blaster. The JTAG debug module which can be seen in figure 4 is necessary to enable the Host computer to control the Nios II processor. This module allows the Host computer to execute processes like...
downloading programs into memory, starting and stopping execution of programs, setting program breakpoints and observing real time execution information [5, page 3].

All the components of such a Nios system which can be implemented on the FPGA are defined by a hardware description language. Since the HDL code for such a system will be relatively long it is convenient to use a CAD tool like Altera’s SOPC Builder for the system design. This tool allows the user to choose and specify the required components of the system he wants to design and the HDL code describing the system is generated automatically based on the specifications given by the user.

For creating a Nios system with the SOPC Builder, the following components have to be included in the design:

• The Nios II processor. The user can choose between the three different configurations of the processor described above, when adding it to the design in the SOPC Builder.

• The On chip memory. Here the data width and the total size of the on chip memory can be specified.

• The input and output interfaces have to be added. The user can specify how many bits the parallel input or output ports shall have.

• The JTAG UART interface for communicating with the host computer.

Furthermore, the Nios II processor requires a clock to make it operate [8, page 5]. One possible option to provide this clock signal is to use the 50 MHz reference clock which is integrated in the DE2 board. This can be done by specifying the clock frequency for the processor to 50 MHZ inside the SOPC builder and the source for the clock signal has to be defined as External. Later when the Nios system is integrated in a Quartus II project, the clock input of the Nios II processor can be connected to the 50 MHz reference clock provided by the DE2 board by specifying the appropriate pin assignments.

When all components have been added to the system in the SOPC builder, there are some more adjustments to be made. Since the SOPC builder software chooses names automatically for the various components included in the Nios system, like for example the input and output interfaces, the user can specify new names for those components to make their identification easier. Furthermore the base and end addresses of the different components in the designed system can be either assigned manually by the user or automatically by the SOPC builder [8, page 12]. For the option of assigning them automatically the user has to select the command Auto-Assign Base Addresses. Finally it is necessary to specify the reset vector and the exception vector. The reset vector defines the behaviour of the Nios II processor after it has been reset. It saves the location inside the memory of the instruction, which the processor shall execute after a reset [8, page 13]. Analogous to that, the exception vector saves the memory address of the command to be executed if an exception occurs. The locations for those two vectors can be specified. Usually it is convenient to specify them to be inside the on-chip memory, which is how they were defined in all Nios systems that have been created with the SOPC builder during the project work.
After all the components have been added to the system and the necessary adjustments have been made, the Nios system can be generated, by pushing the Generation button on the bottom of the SOPC builder window [8, page 14]. When the system has been generated the message “System Generation Successful” appears. It is possible for the user at any time, to reopen the Nios system with the SOPC builder, change some configurations or add new components and generate the system again. To finally complete the design the following steps have to be performed afterwards: The Nios system has to be integrated inside the Quartus II project. There could be more hardware circuitry implemented inside the FPGA and outside the Nios system. This additional circuitry can be defined in HDL language, in which case we have to integrate the HDL code, which has been created by the SOPC builder and defines the Nios system, inside the top level HDL file. In addition the necessary connections for the input and output interfaces of the Nios system and for the clock and reset input have to be defined by naming the port signals [8, page 15].

If the user chooses the schematic design method for the overall system, the Nios system can be added to the design as a block inside the Graphic editor. The in and outputs of the Nios system can then be connected manually by drawing the appropriate wires. The following required steps are, to make the necessary pin assignments and then to compile the circuit. After the compilation has been successful the FPGA can be programmed and configured like described before. When the FPGA has been programmed and configured, it is necessary to create and execute an application program for the Nios II processor, which is able to perform the requested operations. For this purpose a special software tool called the Altera monitor program can be used, which shall be described in the following.

2.4 Altera monitor program

To run and debug programs for the Nios II processor the Altera monitor program has been used during the project work. This software has to be installed on the host computer and allows the user to compile, run and debug assembly language or C programs for the Nios II processor [14]. Breakpoints in the program can be set and it is possible to single step through the program which is running on the Nios II processor. Furthermore, this tool enables the user to observe various runtime information like the contents of processor registers and on chip memory locations. Before it is possible to run programs on the Nios II processor the user first has to configure the Nios II system and the Nios II program inside the Altera monitor program.

2.4.1 Configuring a Nios II system

In order to load and compile programs for a Nios II processor by using the Altera monitor program, it is necessary to provide the monitor program with some information about the Nios system, which has been initialized on the DE2 board [9, page 5]. The reason for this is, that the SOPC builder offers quite many different configuration options for the peripherals and memories of the system, and those options have to be considered when implementing programs on the Nios II processor. The information which is needed by the Altera monitor program is saved in a .ptf file inside the folder of the designed project [9, page 8]. The user has to choose the configure system menu inside the Altera monitor program, where he can specify the .ptf file of the project which he wants to implement on the DE2 board. Inside this menu furthermore the cable connection has to be defined as an USB-Blaster.
2.4.2 Configuring and loading a Nios II program

After the system configuration, the user has to specify the source file containing the program, which shall be run on the Nios II processor [9, page 9]. After selecting the option configure program inside the monitor program, the user can specify if he wants to run a program written in C or assembly language, and the file-name of the suitable program. When the system and program have been configured, the user can compile and load the program to the Nios processor by pressing the Compile and load button in the monitor program. The FPGA of the DE2 board must have been configured before with the Quartus II software as described above. Possible errors during the compilation will be shown in the info & Error window of the monitor program [9, page 14]. The error messages include the line of the program which causes the error as well as an indication of the cause of the error. After the program has been successfully compiled and loaded on the DE2 board, it is paused at its first instruction. It is now possible to run and test the program. In the following some different features, which are useful for testing and debugging the program shall be explained.

2.4.3 Running and debugging programs

The user can manually start and stop the program which has been loaded on the DE2 board, by clicking on different buttons inside the Altera monitor program. Furthermore allows another button to single step command by command through the program. The software includes the following different debugging windows, which are all updated every time the program has been stopped.

Assembly window

In this window a human readable version of the machine code of the program can be observed. It allows the user to observe which machine instructions have been created by the compiler out of the assembly instructions or C code which has been written before. In the assembly window both the original C or assembly code as well as the equivalent machine instructions are displayed in different colours [9, page 16]. The assembly code and machine code are different from each other, since the assembly instruction set includes several so called pseudo instructions, which are implemented by more than one machine instructions [5, page 7]. With help of the assembly window the user can examine the program flow. If the user stops the program, the command which will be executed next is highlighted. Furthermore, it is possible to mark certain instructions inside the program as breakpoints, which will cause the program to stop automatically when reaching the specified line.

Breakpoint window

The breakpoint window allows the user to further specify the breakpoints, for which the program execution shall be stopped. One possibility to set a breakpoint is as described before, to stop the program if the program execution has reached a particular address in the program memory. But there are three more possibilities to set breakpoints [9, page 20]. The program can be stopped as well if the processor has performed a read command on a specific register, has written to a certain register or if the processor has accessed the memory at a specific address. In addition to those options the breakpoint window offers the user the possibility to define so called conditional breakpoints. In this case the breakpoint will only trigger if another condition specified by the user is fulfilled as well. The user could for example
specify, that the program should only stop if the program execution has reached a certain address inside the program memory, and the general purpose register r4 contains the value zero at the same time. After the program has been paused due to a breakpoint, the user can continue the program execution from this point by pressing the start button.

**Register window**

In this window the contents of all Nios II registers are displayed. Different formats can be chosen for displaying the register contents, namely binary, octal, decimal and hexadecimal format each in both signed and unsigned representation [9, page 21]. After the program has been stopped, all values in the register are updated, and those register contents which changed since the last program stop are displayed in a red colour so that they can be easily identified. While the processor is paused, register contents can be changed manually by the user as well.

**Memory window**

This window shows the complete contents of the memory space of the Nios II system [9, page 23]. If the program is paused, the user can press a refresh memory button, so that all memory contents will be updated. Similar to the presentation of register contents in the register window, changed memory contents will be highlighted in a red colour whenever the memory window has been updated. The memory contents can be displayed as numbers in different formats or as ASCII characters. Furthermore it is possible for the user to edit memory values at specific addresses.
3 – 5-6K Interface Board

The 5-6K interface board, which was used as the main analogue interface board in the project, has been developed by the company Texas Instruments. It offers the user a platform for running various data converting evaluation modules. In this project the board was used as a platform for an analogue to digital converter as described before. Before the functionality of the analogue to digital converter will be explained, a brief overview is given of the components included in the 5-6k interface board.

The interface board contains two signal conditioning sites, two serial EVM (Evaluation module) sites and a parallel EVM site system [12, page 1]. All evaluation modules which are compatible with the 5-6k board must have a standardized analogue interface as well as a standardized power connection [12, page 1].

3.1 Power connections to the interface board

The board requires several supply voltages to make it operate. These voltages can be applied externally over different screw terminals. The interface board contains 6 screw terminals in total, two for applying analogue voltages and four for digital voltages [13, page 1-2]. The analogue voltages are necessary for the signal conditioning, sensor boards and amplifiers on the board [13, page 1-2]. The two needed analogue supply voltages are a +5V or -5V voltage supply and one analogue voltage Va within the range of +18 V to -18V.

The digital voltages are necessary to provide the power for the digital logic needed in the evaluation modules like ADCs or DACs [13, page 1-2]. These evaluation modules require at least two supply voltages of 3.3V and 5V for their operations. The needed voltage supplies do not always have to be provided over the screw terminals of the 5-6k board. If the board is connected to other components like an Evaluation module, it is as well possible to apply those voltages to the EVM instead. In this project, for example, have all voltage supplies been connected to the analogue to digital converter.

3.2 Signal conditioning sites

The two signal conditioning sites each contain a 20 pin analogue input/output header [13, page 2-2]. If those headers are used as input pins or output pins depends on the functionality, for example, if the board is connected to an ADC or a DAC as Evaluation module. Those 20 pin headers can be used for data transmission of up to eight single ended or four differential signals to or from the Evaluation modules. Furthermore, external reference voltages can be applied to the data converter over these headers as well [13, page 2-2]. Depending on the converter type, different numbers of analogue signals and reference voltages might be required. If the evaluation module is a Single-channel converter, most of the 20 pins will remain unused. Both signal conditioning sites share the same power bus, so that the analogue power supplies need to be provided only once to the 5-6k board. Furthermore, there are two more 20 pin headers called Analog 1 and Analog 2, which are not connected to anything inside the 5-6k board [13, page 2-2]. Their purpose is to provide stability for the signal conditioning board [13, page 2-2]. In this project the signal conditioning board which was used was basically working as a buffer for the analogue input signal.
4 – Evaluation module: ADS1606

The ADS1606 is a high speed, high precision, delta-sigma analogue to digital converter with a resolution of 16 bits. It has got a default oversampling ratio of 8 and it achieves a data output rate of 5M samples per second [11, page 1]. The voltage of the analogue input signal is compared to a reference voltage, which can be either provided externally or can be created on the chip as well. The digital output data is provided over a simple parallel interface, which can be connected to the parallel EVM site of the 5-6k interface board. The ADS1606 needs a 5V analogue supply and a 3V digital supply, to make it operate [11, page 1].

In case the analogue input voltage signal is too high and goes beyond the supported input range, this will be reported by an out of range monitor and indicated over a specific digital output pin [11, page 17]. Furthermore, the board contains an LED to inform the user that an out of range event has happened [10, page 2-3]. The reference voltage Vref is scaled inside the ADC in such a way, that the voltage input range of the ADC is limited by the minimum value of -1.467 Vref and the maximum value of 1.467 Vref. Since the ADC has got a resolution of 16 bits and a data representation in two’s complement, the analogue input voltage of 1.467 Vref will result in the maximum positive digital output value of 7FFFh while an input voltage of -1.467 Vref will produce the maximum negative digital output value of 8000h. However, to accomplish the most accurate results the producers recommend to choose an input range between -1.165 Vref and 1.165 Vref for the analogue input voltage [11, page 17].

4.1 Clock input

The ADS1606 chip device requires a clock signal which controls the sampling of the modulator [11, page 20]. There are two possible options to apply the clock signal. The user can either choose to apply an external clock signal, or use an on-board 40 MHz oscillator as the clock [10, page 1-4]. One of the two options can be chosen by setting a jumper named W7 on the ADS1606 module. If the clock signal shall be supplied externally, the board contains a CLK input pin for this purpose. Taking into account the high speed in which the ADC shall operate, it is necessary to use a clock of high quality to assure results of satisfying accuracy [11, page 20]. Therefore Texas Instruments recommends the user not to take inadequate clock sources like for example frequency synthesizers for the clock supply but to make usage of Crystal clock oscillators instead, which offer clock signals of very high precision [11, page 20].

4.2 Control signals and data retrieval

In the following the various control signals of the ADC shall be explained and how the digital data can be obtained. The output data of the evaluation module is accessible via a parallel data connector which is named as J2/P2. This header contains 40 pins in total. The 16 bit digital output Data word appears on the first 16 odd numbered pins, starting with pin1 and ending with pin31. All pins of even numbers are connected to the ground of the evaluation module. The four remaining odd pins, (pin33, pin35, pin37 and pin39) are not connected to anything. The output data is represented in binary two’s complement and the most significant bit or the sign bit can be observed from pin31 while the least significant bit appears at pin number 1.
The ADC can be controlled by applying external control signals to another parallel 20 pin interface, named J3/P3 [10, page 2-5]. The functionality of the different pins of the J3/P3 control header can be seen in the following table.

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Function</th>
<th>Pin Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS</td>
<td>2</td>
<td>IO Ground</td>
</tr>
<tr>
<td>3</td>
<td>WR</td>
<td>4</td>
<td>IO Ground</td>
</tr>
<tr>
<td>5</td>
<td>RD</td>
<td>6</td>
<td>IO Ground</td>
</tr>
<tr>
<td>7</td>
<td>Not connected</td>
<td>8</td>
<td>IO Ground</td>
</tr>
<tr>
<td>9</td>
<td>Not connected</td>
<td>10</td>
<td>IO Ground</td>
</tr>
<tr>
<td>11</td>
<td>A1</td>
<td>12</td>
<td>IO Ground</td>
</tr>
<tr>
<td>13</td>
<td>A2</td>
<td>14</td>
<td>IO Ground</td>
</tr>
<tr>
<td>15</td>
<td>Not connected</td>
<td>16</td>
<td>IO Ground</td>
</tr>
<tr>
<td>17</td>
<td>Not connected</td>
<td>18</td>
<td>IO Ground</td>
</tr>
<tr>
<td>19</td>
<td>DRDY out</td>
<td>20</td>
<td>IO Ground</td>
</tr>
</tbody>
</table>

Table 1, pin functionalities of the control header J3/P3

It can be observed from the table, that the Evaluation module can be controlled by 6 different control signals. The Signal DRDYout is the only output signal of those control signals. The falling edge of this signal indicates that a new data word is available [11, page 20]. The other 5 signals are input signals to the Evaluation module and have got the following purposes.

The signals A1 and A2 carry the information about the base address of the Evaluation module, which shall be accessed. It is possible to define one out of four possible base addresses for the evaluation module. The base address can be specified by the two Jumpers of the jumper terminal J8 on the EVM [10, page 2-3]. This makes it possible to design systems, which include up to 4 different evaluation modules at the same time. By changing the input signals A1 and A2, it is possible to choose in real-time which of the different Evaluation modules shall be accessed. If only one Evaluation module is included in the design, the user just has to assure, that the logic state of the Jumper settings on J8 matches the external control input signals A1 and A2 [10, page 2-3]. If the signals and the jumper settings match, the EVM generates a CS (chip select) signal for the ADC [10, page 2-3]. This can then further be specified as a read interval or a reset interval.

By applying an external CS control signal, it is possible to disable the chip select signal. It is then neither possible to read out data from the EVM or to reset the EVM. Similar to this can the read out function of the EVM be disabled, by applying a voltage to the RD pin. Thus it is only possible to read out digital data from the analogue to digital converter if both control signals RD and CS are low, like shown in the following table [11, page 20].

<table>
<thead>
<tr>
<th>CS</th>
<th>RD</th>
<th>Dout [15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Active</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>High impedance</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>High impedance</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>High impedance</td>
</tr>
</tbody>
</table>

Table 2, CS and RD control signals related to output data
By applying a voltage to the WR control pin finally, it is possible to reset the evaluation module. In addition to this way of resetting the board, the EVM offers the possibility to manually reset it by pushing the switch SW2 on the board [10, page 2-3]. The user can specify which of the two possibilities for resetting the board shall be activated, by adjusting certain jumper settings on the EVM. During the reset, all digital circuits of the EVM are cleared, the digital output Data word (Dout[15:0]) is forced low and the DRDY signal, which indicates the data availability, is forced high [11, page 21]. After the reset pin has been released, the DRDY signal will go low on the second rising edge of the clock signal. It is recommended to wait at least 47 DRDY cycles after a reset before collecting digital output data from the EVM. The reason for this is that the digital filter inside the ADS1606 requires some time until immediate changes of the input signal voltage level will propagate to the digital output [11, page 22].

Rapid changes in the input signal level can occur amongst others if a multiplexer is placed in front of the inputs, if the device gets out of the powered down mode or after the device has been reset. The time-duration which the device needs until the output data can be considered to be relatively accurate again is called the settling time. After 47 DRDY cycles the output data of the ADS1606 can be assumed to be fully settled, since the settling error drops below 0.001% [11, page 22]. A DRDY cycle has got the length of 8 clock periods. For a clock frequency of 40 MHz the settling time of the evaluation module can therefore be calculated to be 9.4 us long.

As described before, the falling edge of the DRDY signal indicates that a new digital data word is available at the output. However, it will still take a certain time after a falling edge of the DRDY signal, until the accurate digital output data indeed appears at the output. A timing diagram of the DRDY signal, the clock signal and the available output data can be seen in figure 5 [11, page 8].

![Figure 5. Timing diagram for the data retrieval of the ADS1606](image)

The EVM device can operate for clock frequencies between 1 and 50 MHz. If we consider a clock frequency of 40 MHz, which is equal to the frequency of the on-board oscillator, the time durations seen in the picture will have got the lengths like in the following table [11, page 8].
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Time duration in ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Clock period</td>
<td>25</td>
</tr>
<tr>
<td>t2</td>
<td>Rising edge of clock to DRDY low</td>
<td>10</td>
</tr>
<tr>
<td>t3</td>
<td>DRDY pulse width</td>
<td>100</td>
</tr>
<tr>
<td>t4</td>
<td>Data invalid to falling edge of DRDY</td>
<td>Max. 10</td>
</tr>
<tr>
<td>t5</td>
<td>Falling edge of DRDY to Data valid</td>
<td>Max. 15</td>
</tr>
</tbody>
</table>

Table 3, time durations included in the timing diagram

The time delay from the falling edge of the DRDY signal until the valid Data is available has to be taken into account when collecting data from the ADC. A convenient method to assure that accurate data is retrieved might be to collect the data in the moment of the rising edge from the DRDY signal, since this moment is exactly in the middle of the time duration in which valid data is available at the output of the EVM.

The timing diagram shown in figure 5 requires that both control signals RD and CS are low so that the digital data can be read out from the EVM. However, if one of those two signals appears to be high there will be no output data available any more. The changes of output data availability will be delayed compared to the changes of the control signals RD and CS. A timing-diagram for changes in RD and CS can be observed in figure 6.

![Figure 6. Dout Inactive/Active timing](image)

It can be seen in the diagram that if - during the operations of the EVM - either the RD or the CS signal changes from low to high, it will take 15 ns until no valid output data is any longer available. If on the other hand both RD and CS are changed to low, after at least one of them had been high before, it will take a time of 15 ns as well, until the output data is accessible again.
5 – The design and realization of the project

5.1 Soldering the cable

The main task of the project was to transmit digital output data from the ADC to the DE2 board, and to save this data on the memory of the board. In order to realize this data transmission the cable for connecting the DE2 with the ADS1606 Evaluation module had to be soldered. The cable had to fulfill basically the two following tasks:

- It had to connect the output data header J2/P2 of the Evaluation module to the DE2 board, so that the digital data could be transmitted to the DE2 board.
- Furthermore, the DE2 board had to be connected to the control header J3/P3 so that the two boards could be synchronized with each other and the evaluation module could be controlled over the DE2 board.

The cable connection to the DE2 board was realized over one of the two 40 pin expansion headers of the board. Those 40 pins include two ground connections as well as two voltage levels of 5V and 3.3V. This leaves 36 pins which can be assigned as in and outputs to the FPGA on the DE2 board. The number of pins was sufficient to realize the data transmission and the connection of the control signals as well by using only one of the two expansion headers. The following drawing shows a schematic of how the cable was used for the necessary connections.

![Figure 7. Schematic of the cable connections](image)

The cable was produced by merging a simple 40 pin parallel IDE cable and another 20 pin cable. Various pins of the cables had to be cut and soldered together so that all necessary connections could be realized. In the following table all connections of the cable are documented. The cable connectors are named according to the headers to which they were connected in the project as DE2, J3/P3 and J2/P2.
In this table all connections of the soldered cable can be seen. Not all of those cable connections are actually used in the project design. 16 pins are used for the digital data transfer, one for connecting the GND of the DE2 with the GND of the EVM and 6 more pins are used for transmitting control signals, including the DRDY for synchronization purposes. The red cable which is connected to the 3.3V level of the DE2 has been connected to the EVM in order to provide a digital voltage supply. The blue cable, connected to the 5V supply of the DE2 board, has not been used and was isolated.

<table>
<thead>
<tr>
<th>Pin number of the cable</th>
<th>Pin number of the cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side connected to DE2</td>
<td>Side connected to J2/P2</td>
</tr>
<tr>
<td>1,3,5,7,9</td>
<td>1,3,5,7,9 (same pin numbers)</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11 (5V connected to a blue cable)</td>
<td>Not connected</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>13,15,17,19,21,23,25,27</td>
<td>same pin numbers</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>29 (3.3 V connected to a red cable)</td>
<td>Not connected</td>
</tr>
<tr>
<td>30 (GND)</td>
<td>30 (GND)</td>
</tr>
<tr>
<td>31,33,35,37,39</td>
<td>same pin numbers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin number of the cable</th>
<th>Pin number of the cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side connected to DE2</td>
<td>Side connected to J3/P3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>18</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>22</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 4, pin connections of the soldered cable
5.2 Configuring the hardware inside the FPGA of the DE2 board

The circuit for the FPGA was designed using the schematic entry method. The design includes a Nios II system which has been created with help of the SOPC builder. The complete final schematic diagram of the hardware circuit inside the FPGA can be seen in the following picture.

The left side of the graphic shows a screen shot of the complete Hardware schematic which has been created with the graphics editor in the Quartus II software. The right side shows the same screen shot plus an explanation of the structure and the various elements included in the design. The hardware circuit of the FPGA has got the following in and outputs:

**Inputs:**
- 17 pins from the expansion header, transmitting the 16 bit data word and the DRDY signal.
- Reference Clock from the 50MHz on board oscillator of the DE2 board.
- 7 inputs from switches of the DE2 board. Five of them allow the user to manually define the five control signals which will be sent to the EVM. One switch allows clearing the memory inside the Nios system, where the digital 16 bit data words shall be saved. Another switch is included for resetting the Nios II processor.

**Outputs:**
- 5 pins of the expansion header over which the control signals are transmitted towards the EVM.

In the following the various hardware blocks included in the design shall be explained.
5.2.1 Synchronization and data retrieval

The hardware circuit inside the FPGA has been designed in such a way, that only valid Data words can enter the Nios system. The task of retrieving accurate data at appropriate times is handled in the FPGA hardware circuit outside the Nios system. After entering the FPGA, the sixteen data signals and the DRDY signal coming from the Expansion header pins all have to be synchronized with the clock of the DE2 board.

For this purpose, all of those seventeen signals are fed into a D flip-flop each as it can be observed in figure 9. The clock inputs of those D flip-flops are connected to the 50 MHz reference oscillator of the DE2 board. After the digital Data signals and the DRDY signal have been synchronized with the DE2 clock, the next task is, to obtain accurate digital data words. As described before, the falling edge of the DRDY signal indicates that a new data word is available. However, since it still takes a certain time from the falling edge of the DRDY signal until the data is available, it is more convenient to collect the data at the moment of the rising edge of the DRDY signal in order to assure that the data is accurate. After the synchronization the sixteen data bits are fed into sixteen flip flops again. Those flip flops are only enabled when the DRDY signal reaches its rising edge. To realize this it was necessary to transform the DRDY signal into another signal \( \text{DRDY}_{\text{new}} \), which is only high for a certain time, after the rising edge of the DRDY signal. The two signals can be seen in figure 10.

![Figure 9. Flip flops for synchronization](image)

![Figure 10. DRDY and DRDYnew](image)

To create the \( \text{DRDY}_{\text{new}} \) signal out of the \( \text{DRDY} \) signal it was necessary to build a certain logic circuit inside the FPGA. This could be achieved by inventing a simple solution which included a D flip-flop, an inverter and an AND gate. After drawing the circuit in the graphics editor and simulating it for several test vectors, it could be seen that the circuit showed the desired functionality and could be used for creating the \( \text{DRDY}_{\text{new}} \) signal.
The circuit can be seen in figure 11. It is included in figure 8 as well as the part which is named *DRDY logic*.

![Figure 11. DRDY logic](image)

We can see that the output of the “and gate” will only be high, if both the momentary *DRDY* signal is high, and the previous value of the *DRDY* signal which is delayed by the flip-flop has been low. Those two conditions will only both be true in case of a rising edge of the *DRDY* signal, which is exactly the time moment this circuit shall identify. The *DRDYnew* signal is connected afterwards to the enable input of the sixteen flip flops which are connected to the data bits.

![Figure 12. Circuit for data retrieval](image)

Figure 12 shows the complete circuit of synchronization and data retrieval for one input bit (Dout[0]). The upper branch in this figure is identical for all 16 data word inputs and the *DRDYnew* signal is connected to all of those 16 branches in the same way as shown in this figure for Dout[0]. The complete schematic for all 16 bits can be observed in figure 8 where it is included in the blocks “DRDY logic” and “Flip Flops for synchronization”.

Since the DRDY logic circuit will cause a certain delay in the *DRDYnew* signal, the 16 input signals have to be delayed accordingly before the data is collected. This is necessary to assure that the original data word will be collected which appeared at the input simultaneously with the rising edge of the *DRDY* signal. In order to realize such an equivalent delay for the data word one delay flip flop is included in each of the sixteen branches like it can be seen in figure 12 and figure 8.
After the 16 data bit data word has passed this circuitry and the accurate data bits have been collected, they are then fed into the Nios system.

5.2.2 The Nios system

The Nios system was created with the SOPC builder like described above. The version of the Nios processor was specified as “economic” and the on chip memory was chosen to be 4 Kbyte large with a data width of 32 bits. Furthermore the JTAG UART interface was included and the following in and output interfaces, which can be seen in figure 13.

• The clock input of the Nios system was connected to the 50 MHz on board clock oscillator (pin name: N2) like all the D flip-flops for synchronization as well.

• The input for resetting the Nios II processor (reset_n) was connected to the pushbutton “Key3” (pin name: W26)

• The input which is named “in_port_to_the_clear” was connected to the Toggle switch “SW16” (pin name: V1). The purpose of this input was to offer the user the possibility to manually set the values of the on-chip memory back to zero, were the sampled input data words were saved.

• The input “ctrswitches” is a five bit parallel input interface, which is connected to the five Toggle Switches “SW0” to “SW4”. These switches allow the user to set the logic states for the 5 input control signals to the EVM, which were described before. Over SW0 the logic state for the WR resetting signal can be defined. Setting SW1 allows the user to specify the CS signal and setting SW2 defines the RD signal. The logic states for the address selecting signals A1 and A2 finally can be set by the Switches SW3 and SW4. In order to enable the transmission of the control signals defined by the Switches to the EVM, the Nios system furthermore contains:
• A 5 bit output port named “output_port_from_the_controlls_signals” as can be seen in figure 13. This output port is connected to the expansion header from where the control signals can via cable be transmitted to the EVM.

• Finally the Nios system contains two more 8 bit parallel input interfaces over which the 16 bit data words can be fed inside the system. It might be of importance to mention here that the DRDY signal has not been fed inside the Nios system in this design, since the collection of the accurate data has already been handled before outside the Nios system.

5.2.3 The C-program for the Nios II processor

The program which ran on the Nios II processor can be observed in figure 14. It had to perform several tasks which included to continuously copy the logic state from the address of

```c
#define datain1 (volatile char *) 0x00030000  // Defines the addresses of the various
#define datain2 (volatile char *) 0x00030100  // inputs and outputs of the Nios system
#define ctrlsignals (char *) 0x00030200  // as they have been set inside
#define cthres (volatile char *) 0x00030300  // the SOPC builder.
#define cclear (volatile char *) 0x00030400

void main()
{
    int *memo = 0x0001fe0;  // defines two pointers to the on-chip memory
    int *memo1 = 0x0001fe0;  // where the samples shall be saved
    int a=0, b, c, d=0, g=0;

    while (1)
    {
        if (a<=100 && b!= *datain1 || c!= *datain2)  // collects hundred data word samples
        {
            a++;
            *memo = *datain1;  // and saves them in the on-chip memory.
            memo = memo+1;
            *memo = *datain2;  // A new data word is saved if the input
            memo = memo-3;
            b = *datain1;
            c = *datain2;
        }
        else if (*clear != 0)  //Sets the on-chip memory values which are
            // reserved for the data samples back to zero.
        {
            for (g=0; g<200; g++)  // (If toggle switch "SW16" has been pressed)
            {
                *memo1 = 0;
                memo1--;
            }
            memo1 = 0x0001fe0;  // sets the addresses of the pointers for the
            memo = 0x0001fe0;  // samples back to their original value
            g=0;
            *clear=0;
            a=0;
        }

        *ctrlsignals= *ctrlSwitches;  // copies the input of the control switches to the output
        // of the expansion header for the control signals
    }
}
```

Figure 14. The C-program for the Nios II processor
the 5 bit input from the control switches to the address of the 5 bit output connected to the expansion header in order to allow for the transmission of the control signals.

In addition, a certain number of samples from the digital input data word had to be saved in the on-chip memory and a “clear” functionality was included which allowed the user to set the on-chip memory of the processor, which was reserved for saving the samples, back to zero. This makes it possible to create a new number of samples without resetting the Nios II processor.

The program has been written in such a way, that a new data sample will only be saved in the on-chip memory if there has been a change in the input data. This approach is convenient for sampling signals with a continuously varying voltage level, like for example sinusoids or triangular signals. However, this method is somewhat insufficient for collecting input signals which have constant voltage levels over certain time periods like for example rectangular signals.
6 – Results, conclusion and future work

When the design was tested it was necessary to apply four different supply voltages for the 5-6k interface board and the evaluation module. All those voltage levels were applied directly to the EVM. Three external voltage sources were used for applying the +VA and –VA analogue voltages of +5V and – 5V as well as the 5V supply for the digital circuitry inside the EVM. Another 3.3V digital voltage supply was taken from the DE2 board by connecting the red wire of the soldered cable. After the necessary jumper settings had been made and the two boards were connected together by the cable, the design could be tested.

6.1 Results

The communication between the two boards over the soldered cable worked well in both directions. The control signals for the EVM could be manually controlled by the toggle switches and it was possible to collect 100 data samples and save them on the on-chip memory of the DE2 board. The clear functionality in which the on-chip memory values were set back to zero and new samples could be collected afterwards worked fine as well. However the signals which were collected from the EVM were similar to noise. The signals which were fed into the buffered input of the 5-6k interface board did somehow not influence the digital output of the EVM. The reason for this might have been some wrong jumper settings on the 5-6k interface board.

6.2 Future work

The problem with the signal evaluation inside the 5-6k interface board and the EVM module has to be solved. Different input signals could be evaluated and the accuracy of the samples which are collected on the DE2 board should be verified for those signals. Other possible future work could be to develop a system design where not only new data words would be collected if the input has changed, to allow for collection of signals with constant voltage levels over certain time periods.
References


