Lab on HW-SW digital systems codesign
SEL B.Sc. Curriculum (Spring Semester)
Embedded Systems Laboratory
EPFL-STI-IEL

Final Project
Out: 2025-04-17
Due: 2025-05-22
Last git commit: 11:59 am

"The Hunger Games, Pareto Chapter: The galactic speed"

1 Introduction

The goal of this project is to generate **Pareto-optimal accelerator designs** to process **radio signals received by the Very Elegant Galactic Antenna (VEGA) radio telescope** [1]. This student-led radio telescope project at EPFL aims to detect the 21 cm neutral hydrogen line and measure the galactic rotation curve. The VEGA radio telescope is currently installed on the roof of one of EPFL's buildings. It can detect radio emissions from space even in cloudy conditions, as radio waves can pass through the atmosphere with minimal absorption.

Why hydrogen? Hydrogen is the most abundant element in our galaxy. When pointing the radio telescope toward the Milky Way, we can observe 21 cm wavelength radio emissions from hydrogen atoms located in its spiral arms.[1]

In this project, we receive 10 s of raw radio-signal data. Our task is to transform these data into a spectrogram and identify peaks corresponding to the 21 cm hydrogen emission line (see Fig. 1). Interestingly, the detected peaks are not located exactly on the 21 cm line. Why? This is due to the *Doppler effect* —a phenomenon where the observed frequency shifts depending on the motion of the source relative to the observer. If the gas is moving toward us, the signal is *blueshifted* to a slightly higher frequency; if it is moving away, then it is *redshifted* to a lower frequency. In this project, we will detect three such frequency shifts, each corresponding to a different galactic arm. Based on the direction of the shift, can you determine whether those arms are approaching or receding?

Our mission is to process these data and detect the peaks both **quickly** and **efficiently**, using an FPGA accelerator.

1.1 Problem definition

Today, researchers count on many algorithms, organized as "processing pipelines," to process radio-astronomy signals. These algorithms are designed to be extremely efficient since the amount of data to be processed in real time is often on the order of GiB or even TiB. Thus, execution time has traditionally been the sole optimization target. However, recent concerns about energy consumption in data centers, both from the environmental and technological points of view, have motivated the switch from "time-to-completion" to "energy-to-completion" [4].

[1] How does hydrogen emit radio waves? This is a fascinating and complex topic! In short, it has to do with the spin-flip transition of hydrogen atoms. If you are curious to learn more, check out these excellent resources: Wikipedia [2] and the Cosmos Encyclopedia of Astronomy [3].

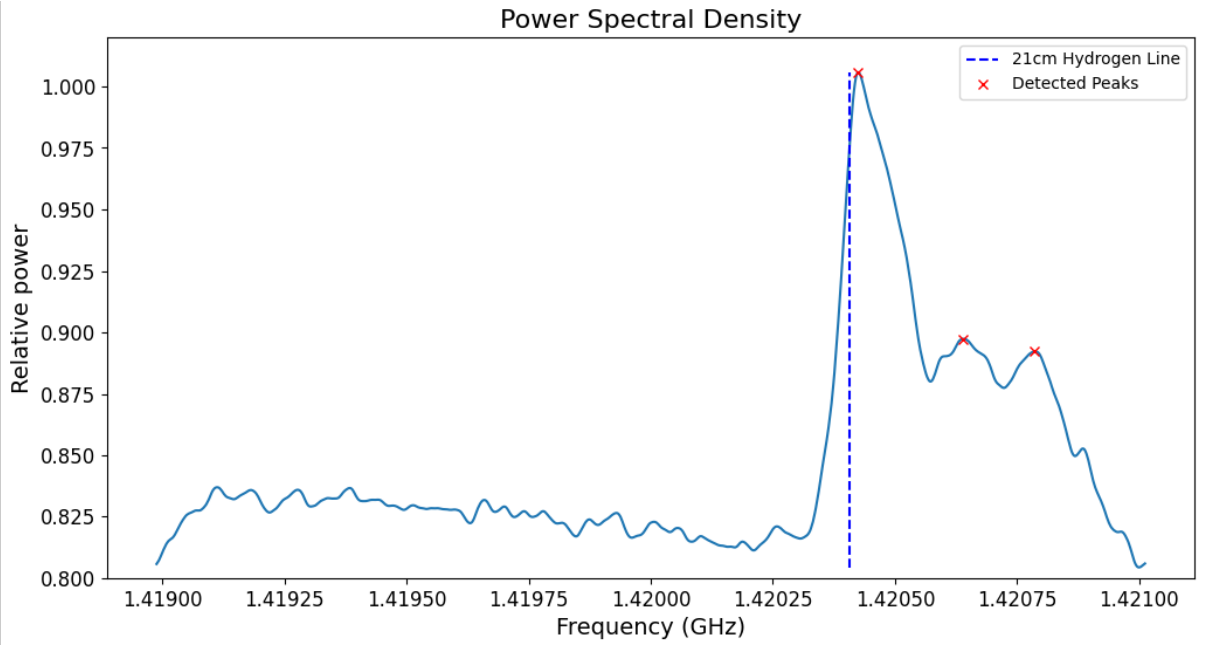

Figure 1: Plot of the signal with the 21 cm neutral-hydrogen band peaks shifted.

The current implementation is performed on a power-hungry server. Our task is to design a set of hardware accelerators that save both time and energy while computing the pipeline for radio signal processing described in Section 2. With this, we will demonstrate that the process can be carried out in a low-power device saving lots of energy while matching the performance required.

As a starting point, we have received the original non-accelerated —pure software— version of the code. The software package includes also a "`Makefile`" that can be used in the development of the project, and an input dataset consisting of:

- A 9.728 s input signal from the actual telescope.
- The gains applied during the processing to calibrate the spectrum.
- The output references to compare against when developing an accelerator.

1.2 Project requirements and evaluation

This exercise can be solved in multiple ways. Each of them will produce a system that executes in a given time, using a number of hardware resources, and consuming a certain amount of energy. Your task is to design solutions that lay on the Pareto front. Each group will produce one or several solutions, which will be evaluated according to the following metrics, described in detail in Section 3.2:

- LATENCY — Total execution time.
- HW RESOURCES — Resource usage in the FPGA (consult the resource cost below).
- ENERGY — Energy measured on the board.
- RESOLUTION — More frequency points in the spectrum produce more accurate tracking of the movement of the galaxy.
- FLEXIBILITY — Different resolutions are useful for different types of analysis. Covering several resolutions (see Table 1) allows astronomers to use your solution in more science cases.
- SNR — Strength of signal relative to noise, which quantifies the amplitude quality of the spectrum emitted.
- TTS — Time-to-solution based on the date and time of solution submission (astronomers would like to have a solution ASAP).

We are looking for solutions that are optimal in at least one of these metrics. With all the submitted solutions, we will generate a graph with the Pareto front of solutions.

Only those teams that produce solutions on the Pareto front will be hired by the astronomers :-)

During the period granted to implement this project, you will have the opportunity to submit multiple solutions, tagging [2] the corresponding commits in your repository. **It is very important to tag and submit the solutions during the project, so that they can be taken into account for the time-to-solution metric**. **Such commits have to include the bitstream and the source code for the Pynq board** to allow us to execute the proposed solutions ourselves. We will verify the output of the accelerated pipelines to validate their correctness using the same output reference that you have in the files. **The signal to noise ratio (SNR) must be greater than 70** dB **for a solution to be considered as valid.**

Every time a new solution is submitted, we will update the Pareto front of accelerator solutions. The current set of Pareto points will be accessible in a live web-site —the entries will be anonymized. On the day of the final presentation, we will show the complete (anonymized) Pareto front of solutions to the entire class.

2 The processing pipeline of the VEGA radio telescope

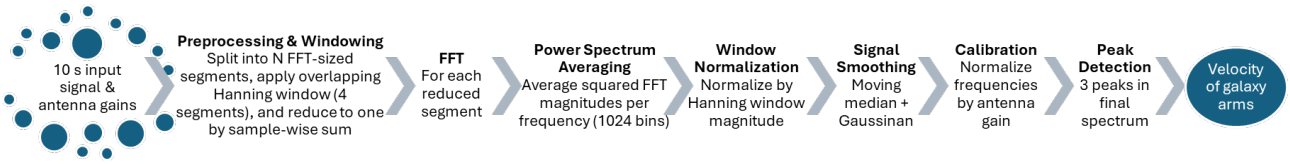

Figure 2: Signal processing pipeline for VEGA. The pipeline replicates the GNURadio advanced spectrometer described in [5].

After the signal is received by the antenna, it is amplified and filtered by the radio frequency (RF) components of the system, and, finally, digitized. The subsequent digital signal processing (DSP) aims to calculate the power spectrum of the signal and highlight the peak that corresponds to the wavelenght of 21 cm [5]. Figure 2 shows the different steps of the DSP pipeline:

- Load the 10 s of the input signal and the antenna gains from the files.
- Split input into segments of the size of the spectrum (NFFT).
- Apply a Hanning window to four consecutive segments and shift one segment every time. There is an overlap of 3 segments in each of two consecutive windows.
- Reduce the four segments to one after applying the Hanning window by adding them sample by sample.
- Compute the fast Fourier transform (FFT) of the resulting segment. Windowing, reduction and FFT are called (#samples)/(FFT size) = 40 000 times.
- Compute the averaged power spectrum. For each frequency (1 024) averaging the square of the FFT coefficients magnitudes over the all segments.
- Spectrum normalization. Normalize the signal with respect to the magnitude of the Hanning window.
- Apply a moving average to the spectrum.
- Smooth the spikes of the signal by applying a moving median to the points in the spectrum above a threshold.
- Apply a moving Gaussian window to the spectrum.
- Calibrate the signal by normalizing each frequency with the antenna's gain.

[2] https://git-scm.com/book/en/v2/Git-Basics-Tagging

- Detect the three peaks in the spectrum.
- Considering that each peak corresponds to one spiral arm of the galaxy, compute their relative velocity according to the Doppler effect.

The output of the results is the number of detected galactic arms and their relative speed with respect to us.

The parameters of each functional block of the pipeline (e.g., the FFT size and the average window) can vary depending on the scientific needs. Your solution will be evaluated according to Table 1.

Table 1: Configuration table for different parameter sets

Config ID	FFT size	Freq. resolution (Hz)	Window size[3]	Peak Detection Window Size	Gain file	Golden reference file
0	1 024	2 000.00	12	3	`gain_0.bin`	`ref_0.bin`
1	2 048	1 000.00	24	10	`gain_1.bin`	`ref_1.bin`
2	4 096	500.00	48	10	`gain_2.bin`	`ref_2.bin`
3	8 192	250.00	96	10	`gain_3.bin`	`ref_3.bin`
4	16 384	125.00	192	200	`gain_4.bin`	`ref_4.bin`
5	32 768	62.50	384	10	`gain_5.bin`	`ref_5.bin`
6	65 536	31.25	768	10	`gain_6.bin`	`ref_6.bin`
7	131 072	15.62	2 304	200	`gain_7.bin`	`ref_7.bin`
8	262 144	7.81	3 072	200	`gain_8.bin`	`ref_8.bin`
9	524 288	3.91	6 144	200	`gain_9.bin`	`ref_9.bin`

IMPORTANT! It is not mandatory to deliver a solution for each configuration, nor one single solution to cover all! The number of configurations covered by your solution(s) is considered in the flexibility metric. This means that it is possible to have a very good solution for only one configuration or a less efficient but more flexible design that covers several solutions. Similarly, solutions that cover configurations with larger FFT sizes are harder to implement; this is taken into account with the resolution metric. The decision is up to you.

2.1 Running the pipeline

The application can be executed for an individual configuration with the following command line:

```
./radioastro -s <signal_file> -c <config_id> [-o <output_file> -p <profile_file>]
```

Alternatively, the application can be executed consecutively for all the configurations with the following command line:

```
./run_all.sh
```

The version that only profiles time can be executed both in a Linux PC and in the Pynq board. However, the version with energy measurement works exclusively on the Pynq board. This version requires programming the "`adc_bit.bit`" bitstream and execution with sudo.

The execution of the initial software version on the Pynq board for the configuration 0 produces the following output:

```
Running with:
Configuration ID: 0
Signal file: ../data_bin/signal_data.bin
Gain file: ../data_bin/gain_0.bin
Frequency resolution: 2000 Hz
Velocity resolution: 422.414 m/s
+----------------------------+-------------------+
+                  Config Table                  +
```

[3]The same window size applied to the moving average, spike smoothing, and Gaussian filtering.

```
+-----------------------------+--------------------+
| LOG2 FFT SIZE               |                 10 |
| FFT SIZE                    |               1024 |
| Window MA                   |                 12 |
| Window PS                   |                 12 |
| Window GF                   |                 12 |
+-----------------------------+--------------------+
Arm 0: Peak Frequency = 1.42042e+09Hz, Velocity = 3.8389 km/s
Arm 1: Peak Frequency = 1.42064e+09Hz, Velocity = 49.4732 km/s
Arm 2: Peak Frequency = 1.42079e+09Hz, Velocity = 80.3196 km/s
SNR: 128.387 dB
+-----------------------------+--------------------+--------------------+------------------+
+                                Time Profiling Table                                        +
+-----------------------------+--------------------+--------------------+------------------+
| Task                        | Time (s)           | Time (ns)          | Percentage (%)   |
+-----------------------------+--------------------+--------------------+------------------+
| Total processing time       |             19.180 |       19179674839 |            100.0 |
| + Power Spectral density    |             19.178 |       19178064480 |            100.0 |
|   + Compute Hanning Gen     |              0.003 |           2666560 |              0.0 |
|   + Compute Hanning Window  |              2.650 |        2650323188 |             13.8 |
|   + Compute Window Reduct   |              1.266 |        1265660745 |              6.6 |
|   + Compute FFT             |             14.099 |       14098600891 |             73.5 |
|   + Compute Magnitude       |              1.061 |        1060661333 |              5.5 |
|   + Compute Normalization   |              0.000 |             26412 |              0.0 |
|   + Compute Reordering FFT  |              0.000 |              4677 |              0.0 |
|   + Compute Frequency Gen   |              0.000 |             78889 |              0.0 |
| + Moving average result     |              0.000 |             53656 |              0.0 |
| + Peak smoothing result     |              0.001 |           1106477 |              0.0 |
| + Gauss smoothing result    |              0.000 |            370376 |              0.0 |
| + Calibration result        |              0.000 |             34655 |              0.0 |
| + Detect peaks              |              0.000 |             26369 |              0.0 |
| + Compute velocity          |              0.000 |              1661 |              0.0 |
+-----------------------------+--------------------+--------------------+------------------+
+-----------------------------+--------------------+------------------+
+                       Energy Profiling Table                        +
+-----------------------------+--------------------+------------------+
| Task                        | Energy (J)         | Percentage (%)   |
+-----------------------------+--------------------+------------------+
| Total processing energy     |             27.518 |            100.0 |
| + Power Spectral density    |             27.518 |            100.0 |
|   + Compute Hanning Gen     |              0.000 |              0.0 |
|   + Compute Hanning Window  |              3.898 |             14.2 |
|   + Compute Window Reduct   |              1.933 |              7.0 |
|   + Compute FFT             |             20.193 |             73.4 |
|   + Compute Magnitude       |              1.494 |              5.4 |
|   + Compute Normalization   |              0.000 |              0.0 |
|   + Compute Reordering FFT  |              0.000 |              0.0 |
|   + Compute Frequency Gen   |              0.000 |              0.0 |
| + Moving average result     |              0.000 |              0.0 |
| + Peak smoothing result     |              0.000 |              0.0 |
| + Gauss smoothing result    |              0.000 |              0.0 |
| + Calibration result        |              0.000 |              0.0 |
| + Detect peaks              |              0.000 |              0.0 |
| + Compute velocity          |              0.000 |              0.0 |
+-----------------------------+--------------------+------------------+
```

2.2 Recreating the execution graphs

To recreate the graphs after execution, the following prerequisites are needed:

```
pip install matplotlib numpy pandas
```

If the Pynq board cannot be connected to the Internet, the graphs can be generated on a PC using the metrics generated on the Pynq board.

The spectrum graph can be plotted with the following command, which will generate the file "`spectrum.png`":

```
python plot_res.py --file out.bin
```

Finally, the results can be plotted with the following command, generating "`profiles.png`":

```
python plot_timings.py -f metrics.csv
```

Table 2: Proportion of execution time and energy consumed in each step for different configurations

Conf.	Task	Subtask	Time (s)	Time (ns)	Proportion (%)	Energy (J)	Proportion (%)
0	TOTAL		19.288	19 287 960 183	100.0	27.703	100.0
0	**Power spectral density**		**19.286**	**19 286 412 455**	**100.0**	**27.692**	**100.0**
0		Hanning Gen	0.003	2 725 960	0.0	0.000	0.0
0		Hanning window	2.656	2 655 878 599	13.8	3.848	13.9
0		Window reduct	1.089	1 088 858 943	5.6	1.534	5.5
0		FFT	14.367	14 366 852 786	74.5	20.689	74.7
0		Magnitude	1.072	1 071 555 377	5.6	1.621	5.8
0		Normalization	0.000	26 428	0.0	0.000	0.0
0		Reorder FFT	0.000	4 677	0.0	0.000	0.0
0		Frequency gen	0.000	59 606	0.0	0.000	0.0
0	Moving average		0.000	46 628	0.0	0.000	0.0
0	Peak smooth		0.001	1 045 416	0.0	0.012	0.0
0	Gauss smooth		0.000	380 037	0.0	0.000	0.0
0	Calibration		0.000	30 670	0.0	0.000	0.0
0	Peak detection		0.000	27 840	0.0	0.000	0.0
0	Compute velocity		0.000	1 596	0.0	0.000	0.0
1	TOTAL		23.200	23 199 529 645	100.0	33.798	100.0
1	**Power spectral density**		**23.194**	**23 193 689 673**	**100.0**	**33.786**	**100.0**
1	Moving average		0.000	119 246	0.0	0.000	0.0
1	Peak smooth		0.004	4 415 791	0.0	0.000	0.0
1	Gauss smooth		0.001	1 100 508	0.0	0.012	0.0
1	Calibration		0.000	71 255	0.0	0.000	0.0
1	Peak detection		0.000	113 677	0.0	0.000	0.0
1	Compute velocity		0.000	1 843	0.0	0.000	0.0
...	...						
9	TOTAL		714.307	714 306 643 351	100.0	1 039.419	100.0
9	Power spectral density		79.099	79 099 135 652	11.1	132.988	12.8
9	Moving average		0.038	38 431 969	0.0	0.061	0.0
9	**Peak smooth**		**540.252**	**540 251 767 570**	**75.6**	**770.153**	**74.1**
9	Gauss smooth		94.818	94 818 292 874	13.3	136.072	13.1
9	Calibration		0.020	20 349 901	0.0	0.025	0.0
9	Peak detection		0.079	78 632 591	0.0	0.120	0.0
0	Compute velocity		0.000	2 363	0.0	0.000	0.0

3 Instructions to implement the problem

3.1 Original execution times

The pipeline is meant to work on data segments that continuously arrive from the network. Instead, the system has a deadline to complete the processing of each segment — precisely, the length of the segment. Therefore, in our measurements we have excluded the time required to load the data from the SD card into the system DRAM. Table 2 shows a break-down of the execution time of each phase of the pipeline when executing different configurations on the Pynq board. The output of this analysis is also shown in Figure 3.

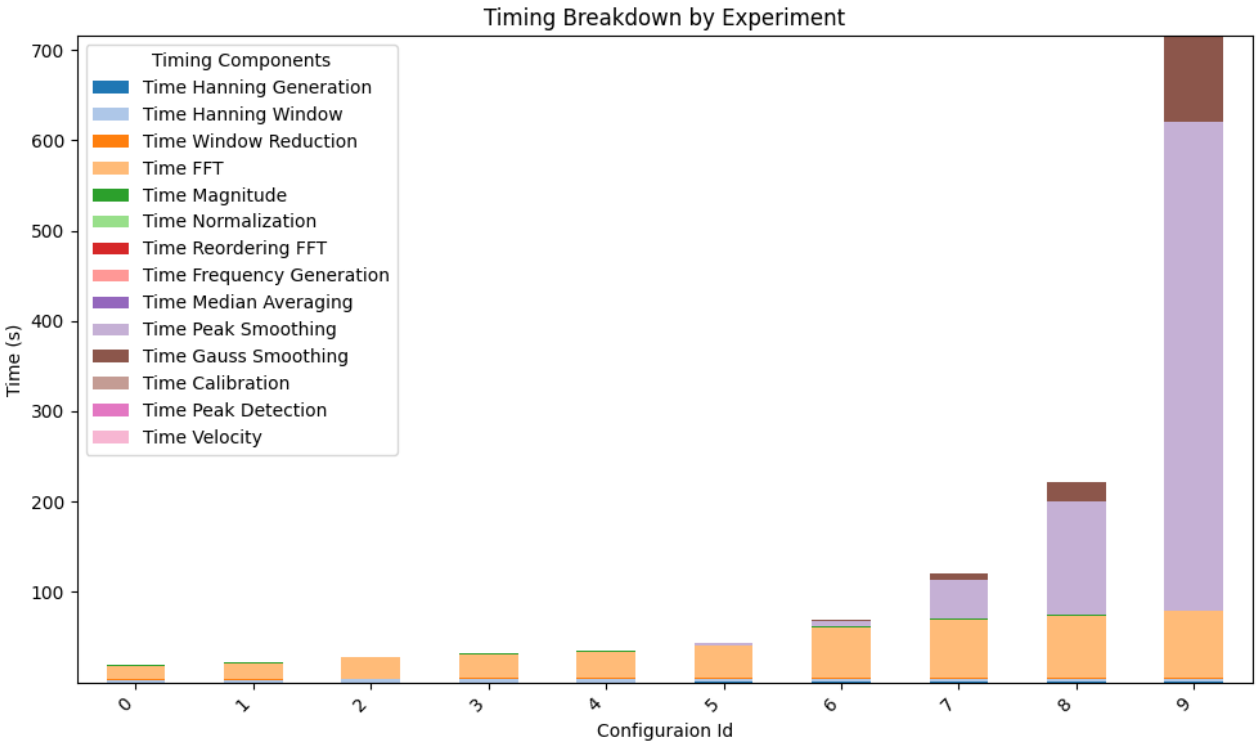

Figure 3: Profiling of all configurations in the PYNQ-Z2 board.

Table 3: Resource availability and computation of the usage factor

RESOURCE	AVAILABLE	FACTOR
CPU cores	2	20/%utilization
BRAM	140	20/140
DSP	220	20/220
LUT	53 200	20/53 200
FF	106 400	20/106 400

3.2 Metrics for project evaluation

The resources available on the FPGA are limited. Specifically, our `xc7z020clg400-1` contains two ARM Cortex-A9 cores, 140 BRAMs (560 KiB in total), 220 DSPs, 53 200 LUTs, and 106 400 FFs. In the context of this exercise, we will evaluate the use of resources for each solution as shown in Table 3.

The percentage of CPU utilization is reported in the supplied baseline code. The total resource cost value of a solution can be computed extracting the resource usage from Vivado and applying the following formula:[4]

$$C_R = \frac{20}{100} \times (\%CoreUtilization) + \frac{20}{140} \times (\#BRAMs) + \frac{20}{220} \times (\#DSPs) + \frac{20}{53200} \times (\#LUTs) + \frac{20}{106400} \times (\#FFs) \quad (1)$$

As example, Table 4 shows the use of resources and the quality of the initial SW solution when executed on the Pynq board.

[4]The maximum resource utilization adds up to 100, although it is generally not possible to reach such a number: As the usage of each component increases in a design, Vivado will struggle more to find a feasible routing, thus compromising the feasibility of finding a solution.

Table 4: Resource usage and quality of the initial SW solution when executed on the Pynq board

Solution	Config.	Latency (s)	HW resources	Energy (J)	Resolution (Hz)	SNR (dB)	TTS (days)	Flexibility [5] (%)
SW	0	19.288	10	27.703	2 000.000	128.4	1	
SW	1	23.200	10	33.798	1 000.000	130.8	1	
SW	2	28.815	10	43.606	500.000	131.6	1	
SW	3	32.703	10	50.376	250.000	134.2	1	
SW	4	35.397	10	54.799	125.000	127.0	1	
SW	5	43.174	10	68.503	62.500	126.9	1	
SW	6	68.202	10	112.439	31.250	136.4	1	
SW	7	121.015	10	190.884	15.625	123.0	1	
SW	8	222.204	10	336.362	7.812	119.7	1	
SW	9	714.307	10	1 039.419	3.906	126.3	1	
SW	(overall)		10				1	100

3.3 Optimization suggestions

During the design of this accelerator, it is possible to follow many different approaches. The following is a list of ideas and hints that can be used to generate new implementations. DISCLAIMER: These optimizations are not mandatory. You are encouraged to devise your own solutions, and there is no preferred order for the following suggestions:

- Carefully analyze the profiling results of each configuration and select which kernels to accelerate on the FPGA's programmable logic (PL). We recommend focusing on a single configuration or a small subset for more effective optimization.
- The full input signal is 152 MiB —too large for a single contiguous physical memory allocation. When accelerating any block within the Welch PSD (e.g., the FFT), process the data in batches of segments.
- If accelerating multiple functional blocks (e.g., Hanning windowing and FFTs), we strongly recommend implementing them as **tasks** and connecting them via **streams**. This enables efficient pipelining between modules.
- Design a small FFT module that can be replicated to process multiple segments in parallel. A simple design can be based on the sequential C implementation provided. Within each stage, you can exploit intra-stage parallelism.
- Sine and cosine evaluations used for computing twiddle factors in the FFT are expensive in hardware. Pre-compute these values and store them in look-up tables (LUTs), which can be loaded into the field-programmable gate array (FPGA) once and reused across FFTs.
- The FFT size coverage is also an optimization opportunity. In the C code, a buffer is allocated for the largest FFT size (512 k elements). Reducing this buffer size can free up resources, allowing more parallel workers—at the cost of reduced flexibility.
- The FFT computation involves multiple stages: $log_2(\text{FFT size})$. While the C code allocates space for all stage buffers, hardware implementations processing one FFT at a time do not need all of them. Ping-pong buffering can reduce memory usage.
- You can pipeline the FFT module to support back-to-back processing, where different FFTs are at different stages in the same hardware module. Note: pipelined FFT stages are incompatible with ping-pong buffering.
- The previous two options will enable you to perform some form of design-space exploration (DSE). For example, you may consider using a pipelined FFT (higher throughput, more resources) instantiated less times, versus an in-place FFT (lower throughput, less resources) instantiated many times. Which solution is faster to process the entire dataset?

[5] E.g., if a solution supports configurations 0 and 2 out of 10, then its flexibility is 20 %.

- The 4-block windowing strategy is similar to the convolution that we implemented in previous sessions. Instead of loading four new segments each time, reuse three segments and shift the window. Then, apply the Hanning window and pass the result to the FFT.
- To tackle multiple configurations, it is possible to either design one single HW system (i.e., bitstream) that copes with all the sizes, or to design a different HW system for each configuration or group of configurations, and program that bitstream dynamically in the FPGA as required.
- Floating-point operations are both slow and resource-intensive. Signal degradation up to 70 dB is acceptable in our problem. To reduce resource usage, use fixed-point (FXP) arithmetic. Ensure the dynamic range is suitable: values can be as small as 10^{-6}. To preserve precision, scale up the data before computation (e.g., multiplying by a constant) and scale it down afterward by dividing by the same factor.
- It is also possible to consider using the `Xilinx FFT IP` core. If using FXP inputs, ensure the data is rescaled to match the format expected by the IP. This includes scaling before and after the computation.
- All functional blocks are connected through SW buffers. If these buffers are allocated on non-cacheable memory to ease the processing with the hardware, then adjacent SW blocks can suffer large slowdowns since the processors are much less efficient when working with non-cacheable memory. Be mindful of memory placement.

4 Tips

4.1 Using the project "`Makefile`"

The included "`Makefile`" is equivalent to the one provided in Session 8. You will have to adapt it to the specifics of your final project and update the associated TCL files. As a reminder, this "`Makefile`" contains the following targets:

- Vitis HLS targets

  hls_project: Just creates the Vitis HLS project
  hls_sim: Creates the Vitis HLS project and runs the C++ simulation
  ip: Creates the Vitis HLS project, synthesizes the design and exports the IP core

- Vivado targets

  vivado_project: Just creates the Vivado project
  bitstream: Creates the Vivado project and runs synthesis up to bitstream generation
  extract_bitstream: If the Vivado bitstream has already been generated, extracts it to this folder

- Generic targets

  clean: Deletes log files and Vitis HLS and Vivado projects; deletes the files in the IP catalog, but keeps the IP catalog ZIP file
  cleanall: Additionally, deletes the bitstream files and the IP catalog ZIP file
  help: Displays a short help message

4.2 Disabling the swap file before taking measurements

Since this project uses a significant amount of RAM memory, it is convenient to disable the swap file before taking time and energy measurements on the Pynq board. To do this, the following command can be used every time the board is started:

```
sudo swapoff -a
```

Alternatively, the swap file can be permanently disabled commenting the corresponding line in the "`/dev/fstab`" file:

```
>sudo nano /dev/fstab

/dev/mmcblk0p1 /boot vfat defaults 0 2
#/var/swap none swap sw 0 0
```

The presence of the swap space can be confirmed using either the `htop` or the `free` commands. It is also possible to increase the amount of available memory by stopping the Jupyter server.

4.3 Characterization of execution time and energy consumption

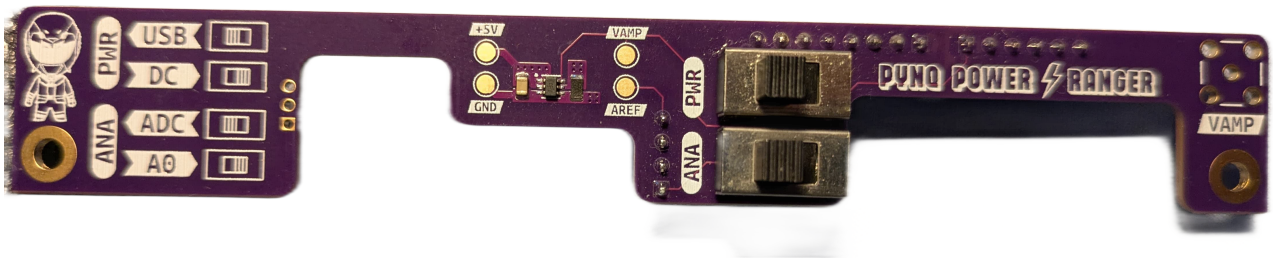

Figure 4: View of the "Pynq Power Ranger" board to measure energy consumption in the Pynq board. Ensure at all times that the two switches are on the position "USB" and "ADC," respectively (as shown in the picture).

To measure the energy consumption of our designs running on the Pynq board, we can use the "Pynq Power Ranger" extension board, shown in Figure 4. The design of the board is available in: `https://github.com/esl-epfl/pynq-power-ranger`.

The initial source files for the project include an example design that can be used to monitor in real time the energy consumption of any software running on the Pynq board. To use this example, program the bitstream, compile the example (with `make`) and run the "`PowerRanger`" executable in one terminal window. In a different terminal window, run any program whose energy consumption you want to monitor. Alternatively, leave the board idle to see the baseline consuption of your board. This simple example does not allow us to monitor any different HW design.

To monitor the energy consumption of our HW designs, we need to include the XADC module in our own Vivado design.

4.3.1 Adding the XADC interface in Vivado

To add the analog-to-digital converter (ADC) module and configure it in a way that is compatible with our drivers, follow these steps:

1. Create a Vivado project in the normal way.
2. Paste the following piece of TCL code in Vivado's TCL console to instantiate and configure the "XADC Wizard IP" from Xilinx:

```
set xadc_wiz_0 [ create_bd_cell -type ip -vlnv xilinx.com:ip:xadc_wiz:3.3 xadc_wiz_0 ]
set_property -dict [list \
  CONFIG.ACQUISITION_TIME {10} \
  CONFIG.ADC_CONVERSION_RATE {39} \
  CONFIG.ADC_OFFSET_AND_GAIN_CALIBRATION {false} \
  CONFIG.CHANNEL_AVERAGING {256} \
  CONFIG.CHANNEL_ENABLE_VP_VN {false} \
  CONFIG.ENABLE_CALIBRATION_AVERAGING {false} \
  CONFIG.ENABLE_EXTERNAL_MUX {false} \
  CONFIG.ENABLE_RESET {false} \
  CONFIG.ENABLE_VCCDDRO_ALARM {false} \
  CONFIG.ENABLE_VCCPAUX_ALARM {false} \
  CONFIG.ENABLE_VCCPINT_ALARM {false} \
  CONFIG.EXTERNAL_MUXADDR_ENABLE {false} \
  CONFIG.EXTERNAL_MUX_CHANNEL {VAUXP0_VAUXN0} \
  CONFIG.INTERFACE_SELECTION {Enable_AXI} \
  CONFIG.OT_ALARM {false} \
  CONFIG.POWER_DOWN_ADCB {false} \
  CONFIG.SENSOR_OFFSET_AND_GAIN_CALIBRATION {false} \
```

```
    CONFIG.SEQUENCER_MODE {Off} \
    CONFIG.SINGLE_CHANNEL_ACQUISITION_TIME {true} \
    CONFIG.SINGLE_CHANNEL_SELECTION {VP_VN} \
    CONFIG.TIMING_MODE {Continuous} \
    CONFIG.USER_TEMP_ALARM {false} \
    CONFIG.VCCAUX_ALARM {false} \
    CONFIG.VCCINT_ALARM {false} \
    CONFIG.XADC_STARUP_SELECTION {single_channel} \
  ] $xadc_wiz_0
```

3. Execute the following TCL line to create a top-level port that will connect the ADC ports to the analog signals coming from the Pynq Power Ranger board:

```
set Vp_Vn_0 [ create_bd_intf_port -mode Slave -vlnv xilinx.com:interface:diff_analog_io_rtl:1.0 Vp_Vn_0 ]
```

4. Use the following TCL line to connect the input port of the XADC IP to the external port of the FPGA:

```
connect_bd_intf_net -intf_net Vp_Vn_0_1 [get_bd_intf_ports Vp_Vn_0] [get_bd_intf_pins xadc_wiz_0/Vp_Vn]
```

5. Finally, add the supplied constraints file ("`constraints.xdc`") to specify the physical FPGA pin to which the `Vp_Vn_0` design port should be connected.

The above TCL fragments are available in Moodle in the file "`recreate_xadc.tcl`" .

4.3.2 Using the XADC in a SW application

Measuring times with the XADC module is very similar to measuring times with the standard function `clock_gettime()`. We need to follow the following steps:

1. Add the source files to the project and update the Makefile if necessary. The necessary files are: "`CAccelProxy.hpp`", "`CAccelProxy.cpp`", "`CXADCProxy.hpp`" and "`CXADCProxy.cpp`" .
2. Declare the base address of the XADC peripheral in the C program as usual. The default address may be `0x43C00000`.
3. Instantiate an object of the `CXADCProxy` class and initialize it as usual.
4. At the beginning of the C program, call the method `StartMeasurements()`. The proxy class generates a background thread that reads values from the ADC with a frequency of approximately 123 Hz. The ADC acquires samples at a higher frequency (31.57 kHz), averaging 256 acquired samples. In other words, the ADC produces samples at a frequency of approximately 123 Hz, each of which is the average of 256 samples acquired over 8 130 µs. This helps to smooth our measurements and avoids generating a large overhead on the ARM cores.
5. Before and after the section of code that needs to be profiled, call the method `GetEnergy()`. This method returns a monotonically increasing value of consumed energy, in J. Therefore, it is possible to simply subtract two measurements taken before and after the section of interest.
6. At the end of the program, the method `StopMeasurements()` should be called to stop the background thread.

The source files available in Moodle contain two different applications that can serve as example for your own measurements. First, the PowerRanger app is the simple monitoring application described above. Second, a time- and energy-instrumented version of the pipeline for this project is also included.

4.4 Programming a bitstream at run-time

It is possible to generate different bitstreams, each optimized for a specific problem size. In that case, the software can load the correct bitstream at run-time before using the accelerators. Programming the bitstream can be done from the C application itself, and it takes less than a second. If you are interested in this possibility for your project, contact us directly for more information.

5 Acknowledgments

We would like to acknowledge the help of Mr. Aurélien Thomas Mathieu Verdier (EPFL LASTRO), Mr. Léonard Georges Théodore Lebrun, Mr. Matteo Veneziano, Mr. Tom Vadot, and the Callista Astronomy Association from EPFL, for introducing us to the VEGA radio telescope and the challenges they are facing in their pipeline design.

We would also like to acknowledge the work of Dr. Taraneh Aminosharieh Najafi, Mr. Alejandro López-Rodríguez and Mr. Rubén Rodríguez, from EPFL, in the design and production of the "Pynq Power Ranger" PCB board used to measure energy consumption.

References

[1] A. Verdier, "The EPFL students radio telescope VEGA," in *Swiss SKA Days*, Sep. 2024. [Online]. Available: https://indico.skatelescope.org/event/1146/contributions/10836/attachments/9866/17292/VEGA_SKACH_DAYS.pdf

[2] Wikipedia. Hydrogen line. [Online]. Available: https://en.wikipedia.org/wiki/Hydrogen_line#Cause

[3] C. E. of Astronomy. Spin-flip transition. [Online]. Available: https://astronomy.swin.edu.au/cosmos/S/Spin-flip+Transition

[4] SEAMS. Seams: Sustainable & energy aware methods for SKA observatory. [Online]. Available: https://seams-project.com/

[5] PhysicsOpenLab. GNURadio software for the 21 cm neutral-hydrogen line. [Online]. Available: https://physicsopenlab.org/2020/07/26/gnuradio-software-for-the-21-cm-neutral-hydrogen-line/