A Safety Methodology for ADAS Designs in FPGAs

WP-01204-1.0 White Paper

This white paper discusses the use of Altera® FPGAs in safety-critical Advanced Driver Assistance Systems (ADAS). It looks at the general safety concept of such applications and provides examples on how to implement certain diagnostics in the FPGA to detect faults in the application.

Introduction

One of the major goals of the automotive industry is to reduce the number of traffic fatalities and severity of accidents. Throughout the last two or three decades, several innovations like the airbag, anti-lock braking systems (ABS), and electronic stability programs (ESP) have helped to come closer to this goal. However with ever increasing numbers of cars on the road, more than these technologies are needed to address a further reduction of accidents and fatalities. In the last few years new features like radar- or camera-based systems have been introduced to make driving safer. Many of these ADAS applications, like adaptive cruise control, lane departure warning, traffic sign recognition, and others, have up to now been predominantly convenience features, with no or only minimal influence on the vehicle behavior. These technologies, however, are starting to take a more active role in the control of the car, with applications like lane keep assist (LKA) or automatic emergency braking (AEB) to achieve the set goal to reduce traffic accidents. With this comes a challenge to ensure that the systems are not creating an increased risk to the occupants of a vehicle or the environment in case the system is malfunctioning.

In this paper we will look into a high-level analysis of a mono front camera system as an example, determine the critical data flows and look at how faults can be identified with existing diagnostic mechanisms using an Altera Cyclone® V SoC as an example. To enable you to further increase the diagnostic coverage of the entire system, we also will provide examples of potential diagnostics you could implement on the system level by leveraging the flexibility and programmability of a FPGA. Using diagnostics customized to your application implementation may in certain cases increase the application performance compared to using a general-purpose microprocessor, digital signal processing (DSP), or other platform.

FPGAs in ADAS Applications

Many of the aforementioned applications have relatively high compute requirements as they extract features from either radar or visible images and determine objects and their trajectory. Often this requires high-performance CPU architectures with many cores, which may or may not be specialized to the task at hand. These CPUs provide the flexibility of programmability to a specific implementation, but have the disadvantage that they are still relatively inefficient due to functionality, which is either not needed in all cases or only used occasionally.

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Power dissipation is another parameter of these applications which needs to be looked at critically. Many of the systems are either located behind the windshield in front of the rear-view mirror with direct exposure to sunlight or in the bumper in front of the radiator and hence cannot dissipate much heat.

Meeting the performance and power requirements can be a challenge when using general-purpose CPU architectures with often many cores running at high frequency. Using a FPGA can solve both of these problems.

Power

FPGAs generally have higher power dissipation than standard logic, but the much more efficient custom implementation of an algorithm can actually reduce the power consumption compared to a general-purpose compute architecture. Another advantage of the FPGA is that much of the data movement between internal and external memory can be eliminated when implementing a streaming processing algorithm. Data transactions on external DDR memory especially can consume a substantial amount of power.

Performance

Stream processing also reduces the likelihood of running into bandwidth problems on external memory, which can be a concern in some applications. The inherent programmability of FPGAs can counter an advantage attributed often to general-purpose compute architectures. It is even possible to implement in-system programmability for updates in the field during series production. In certain cases it may also be easier to develop on an FPGA as it may be available much earlier than a high-performance, CPU-based chip architecture. FPGAs can combine the best of both worlds as there are more and more SoCs, like the Altera Cyclone V SoC family, that integrate general-purpose CPU systems with an FPGA fabric on one chip. The Cyclone V SoC implements two ARM® Cortex™-A9 CPUs with many of the peripherals found on general- purpose microcontroller and processors.

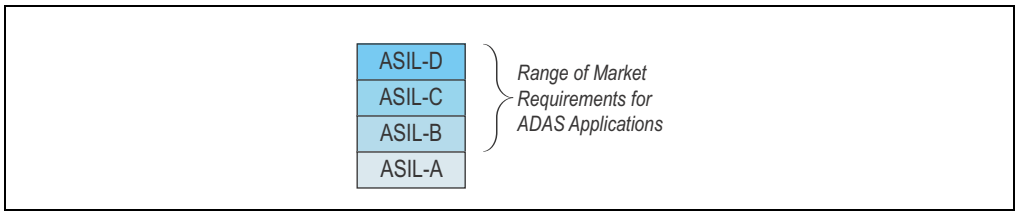
In summary, FPGAs can be a great fit for meeting the functional requirements of these applications. The following also illustrates that it can be as easy or even easier to meet the functional safety requirements of the application compared to other solutions on the market today.

Functional Safety

ADAS need to meet specific functional safety requirements. In 2011, the ISO26262 standard for passenger vehicles up to 3.5 tons was introduced to minimize the risk of a malfunctioning system to create a dangerous situation. This standard tries to address the reduction of systematic faults by implementing a rigorous design process as well as detection of random hardware faults during the application execution. It covers the analysis and development of a system or multiple systems. It also outlines guidelines for individual hardware components used in the system, including potential software running on those hardware components, by specifying requirements for the entire safety lifecycle of the product.

The developer of an application defines specific safety goals and assigns a specific Automotive Safety Integrity Level (ASIL) to each of the goals. The highest ASIL for this application usually defines the requirements to which the development and operation up to end of life of each component has to adhere. Figure 1 shows the current range of ASILs, seen from customer requirements, to which ADAS have to comply.

Figure 1. ADAS ASIL Market Requirements



ASIL-B is the lowest level in the market, but some applications require up to ASIL-D for certain functionality. With an increased ASIL come more stringent requirements.

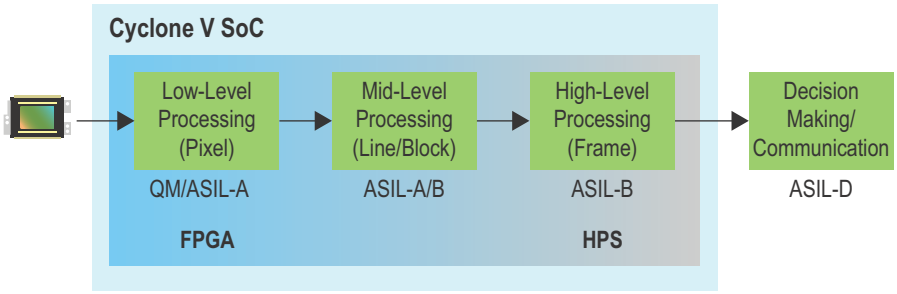
Also generalizing an ASIL on the component or even item (system) level may introduce unnecessary complexities in a specific implementation and can have impact on the cost and schedule of the development. It is sometimes advisable to look into the application requirements in more detail. This is usually done by analyzing the system concept and deriving the safety concept and requirements from it. It may be possible to split the application into several different steps with varying ASILs and thus make it easier and more efficient to implement.

The high-level analysis we will look into in more detail later will provide an example of how it may be possible to achieve this for a mono front camera application.

System Concept of an ADAS Mono Front Camera Application

Our front camera application uses a single image sensor often found in ADAS. Figure 2 shows a high-level block diagram of the system.

Figure 2. High-Level Mono Front Camera ADAS



An image sensor is connected to the image processor, which is the Altera Cyclone V SoC. The signal processing chain and data flow is split into four parts.

- Perform low-level processing on the pixel level by transforming the image into a more useful representation.

- Perform mid-level processing on a line or block of the image, which will extract features like edges by using appropriate algorithms in the form of Sobel filters or the Canny Edge detection algorithm.
- Perform high-level processing where data is extracted on a frame basis to detect and classify objects.
- Track the identified objects, make a decision on whether an action needs to be taken in any dangerous situations, and incorporate the communication of action requests to the microcontroller, for example, the braking and/or steering electronic control unit (ECU).

The low-level and mid-level processing can be implemented very efficiently on the FPGA, but users may also implement some of the mid-level processing on a CPU like the Cortex-A9 processor used in the hard processor system (HPS) of the Cyclone V SoC. The high-level processing, which is predominantly control code, could be mapped to one or both of the Cortex-A9 in the HPS. The last step in the processing chain where the object tracking and decision making is done could be implemented on an external microcontroller.

Throughout the processing, the input data set gets reduced by each step to more meaningful data, and the safety criticality increases with the reduction in data. Therefore, the low-level implementation could be classified to be either quality managed (QM) or a low ASIL level (for example, ASIL-A). The reason for this is that faults introduced during a single pixel can have low or negligible impact on the performance of the subsequent algorithms. The mid-level processing is assumed in this example to comply with either ASIL-A or ASIL-B, and the high-level processing, where objects are identified and classified, could be seen as having to comply with ASIL-B. After the objects are classified, target lists are generated and provided to the microcontroller, which does the tracking of objects and decision making. As this is the most critical portion of the signal chain, we assume it has to comply with ASIL-D as it can have direct impact on the behavior of the car.

In this kind of application, it is beneficial to do an even more thorough analysis of the data flow than done above, as the definition of the safety criticality for each stage can have a direct impact on the performance of the whole system. Putting too stringent safety requirements on the earlier parts of the computation stages could introduce challenges to meet the system performance goals, with very little impact on the overall safety of the system. There may, however, be faults in the lower stages of the processing chain that could have a high impact on the functionality or safety of the system. For example, a permanent fault in the low-level processing function may lead to a permanent corruption of data for the higher-level stage, but this can be easily detected with plausibility checks that have a relatively small impact on the performance of the system. A permanent fault is a fault that cannot be removed even after a power cycle.

For a real system, a more detailed qualitative analysis than the above needs to be conducted to determine possible fault modes and provide the right detection mechanisms for those. For example, this can be done with a fault tree analysis (FTA) or failure mode and effects analysis (FMEA) of the implementation. This would go beyond the scope of this paper and also needs a more in-depth definition of the system concept.

Application Functionality on Component

Figure 3 shows a high-level block diagram of a mono front camera system example. An external power management circuit supplies power to the Cyclone V SoC. Separate voltage monitoring will generate a reset in case the supply voltages are not in the normal operating range. External non-volatile memory could be connected to the quad serial peripheral interface (quad SPI) module to be used during the booting process of the system to load the application and configure the FPGA. We use DDR memory for the application code execution and storage of data and image frames. The external microcontroller connected via an SPI provides the object tracking, final decision making, and communication to the rest of the vehicle infrastructure via the CAN interface.

Figure 3. Mono Front Camera System Example

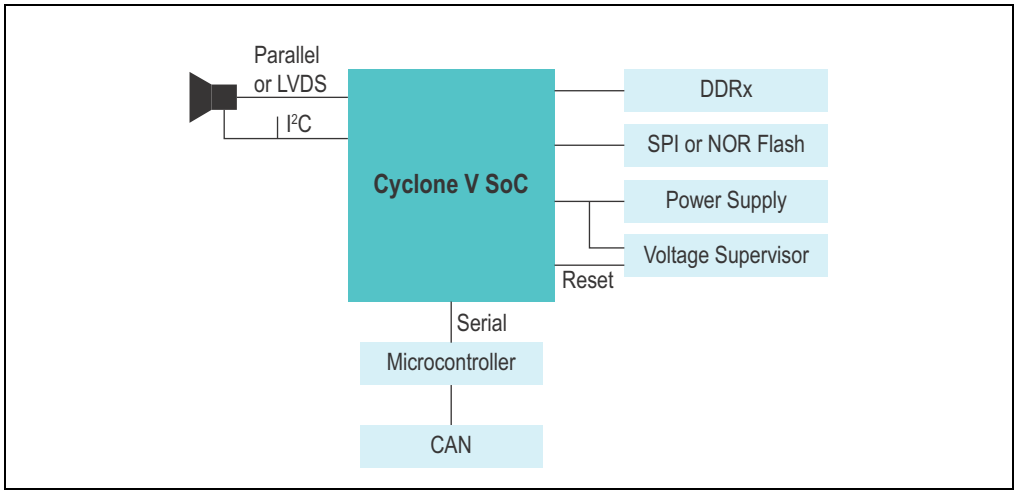


Figure 4 shows the image processor modules involved in a mono front camera application. The FPGA implementation does not go into much detail as each specific implementation may vary, but for the purpose of this paper it is sufficient to show the different analysis steps when looking at a higher abstraction level.

Figure 4. Cyclone V SoC View of Modules

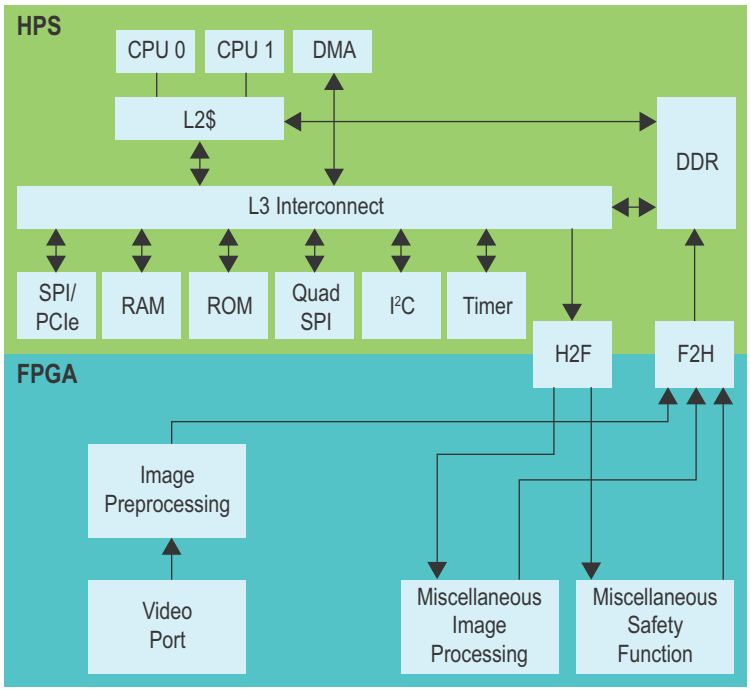


Image data from the image sensor is received by the video port and passed on to the image preprocessing block. This block represents the low-level image processing stage. Once the data passed through the image preprocessing block, it is written to the DDR memory via the FPGA-to-HPS (F2H) bridge in this example, but it could also be passed to the next stage for a more efficient implementation. This could save power and memory bandwidth, and reduce the likelihood of faults occurring during the additional data transactions. The second stage, which is the mid-level processing, is performed by the miscellaneous image processing block. It retrieves the data previously stored in the DDR memory via the HPS-to-FPGA (H2F) bridge, processes it, and writes it to the DDR memory again. The high-level processing stage is performed by the HPS in this example.

Now we will look at some of the diagnostics that could be used to detect faults in the different areas of the design. This will not be an exhaustive analysis of the possibilities, but should give you a starting point for your own analysis. Some of the discussed diagnostics may be able to detect permanent faults while others only transient faults, or both. A transient fault is a fault that appears and subsequently disappears again. For the analysis, we need to consider faults in used memory for a specific function and also potential faults in the logic involved in performing the function. Although memory has the highest failure rate due to its higher susceptibility

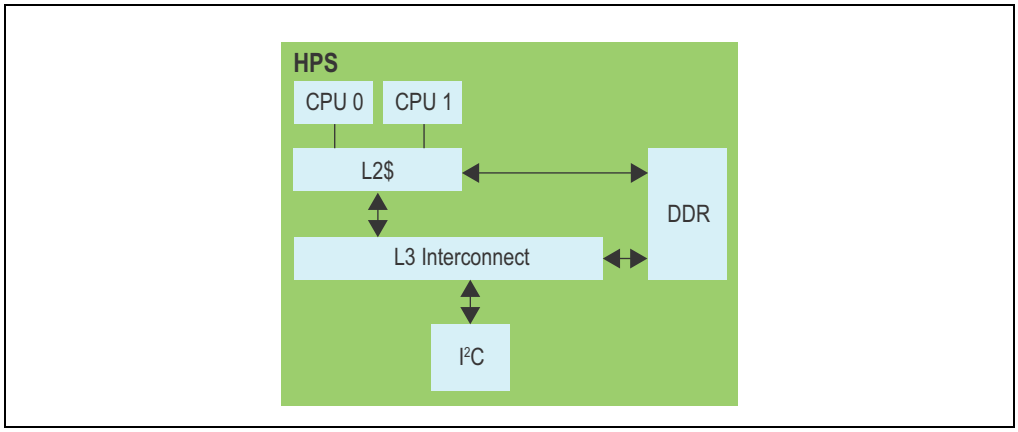
to soft errors, logic cannot be neglected as the amount of it in current and future SoCs is ever increasing, making it challenging to meet the hardware architectural metrics defined by ISO26262. In this analysis, we will make no claim with regard to the achieved diagnostic coverage of the proposed mechanisms as this is in many cases implementation specific.

Image Sensor Configuration

The image sensor needs to be configured before it can be used by the application, and the configuration is constantly modified during the application execution to adapt it to different environmental conditions. For example, the exposure settings may be changed for different lighting conditions, especially when driving into a tunnel or out from a tunnel.

As the image sensor is very critical for the application operation, it is advisable to check its configuration at least once during the Fault Tolerant Time Interval (FTTI). This will not cover all of the potential faults in the sensor, but will take care of the configuration register set.

Figure 5. Modules Involved in Image Sensor Configuration



It is assumed that the sensor is configured by the HPS system. Figure 5 shows the modules that may be involved in the image sensor configuration. Code is executed by one or both Cortex-A9 CPUs and data is transferred from DDR memory via the I^2C module to the sensor. The involved memories are protected by single error correction-double error detection (SECDED) error correction code (ECC) for DDR memory and L2 cache. The L1 caches of the CPUs are protected by parity. This includes the TAG RAMs and also the global history buffer and branch target access cache of the branch prediction unit. The logic (CPUs, L3 interconnect, I^2C, and so on) involved in the sensor configuration is only available once and has no specific diagnostics implemented. To detect potential faults, a write and read-back scheme can be used. For example, CPU0 writes the configuration data to the sensor and CPU1 reads it back. A subsequent comparison between write and read data can detect faults that may have been introduced during the write transaction. Writing the data with one CPU and reading it back with the second CPU can have the advantage that permanent faults in one CPU may be detected. Diversified software may also be able to detect permanent faults when only one CPU would be used for the write-and-read transaction. However, it is more difficult to implement because a detailed analysis of the CPU's internal functionality would be required and it could increase software complexity.

Some of the sensors used in automotive also allow you to transmit certain configuration register data in auxiliary lines of each image frame. With this feature, you could check the settings of the sensor for each frame without having to read the registers via the I^2C interface. This checking could even be implemented in the FPGA while streaming the frame data without overhead on the CPU.

All of the above configuration register tests have the advantage that not only the Cyclone V SoC faults will be covered, but also faults that may be introduced on the external interface to the sensor or inside the sensor itself. Some application implementations may use more sophisticated processor architectures, which implement more internal diagnostics on busses and so on. But faults in the I^2C module and external faults, which may not be covered by built-in diagnostic mechanisms, still need to be considered and require similar schemes as discussed before. In this scenario, the added diagnostics on the processor may have limited benefits.

Table 1 provides a summary of the image sensor configuration diagnostics.

Table 1. Summary of Image Sensor Configuration Diagnostics

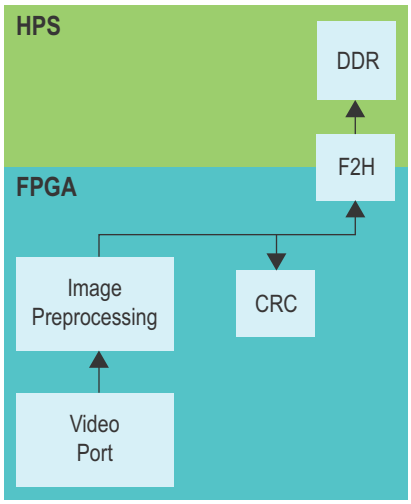
#	Potential Available Diagnostic
1	ECC on DDR memory
2	ECC on Cortex-A9 processor L2 cache
3	Parity on Cortex-A9 processor L1 instruction and data cache, parity on branch prediction unit
4	Sensor register configuration: write and periodic read-back scheme with independent Cortex-A9 processor
5	Sensor register configuration: write and periodic read-back scheme with single Cortex-A9 processor and software diversity
6	Sensor register configuration: periodic check of register content in auxiliary information of image frame with the FPGA fabric

Low-Level Image Processing

As mentioned earlier, it is very unlikely that a change in a single pixel will significantly influence the behavior of the application, so such faults can in many cases be neglected. However, faults that could lead to missing or entirely corrupted frames should be examined.

Figure 6 shows the modules involved in the image preprocessing stage. In many cases the image sensor connects to the image processor via a parallel video interface. In the case of the Cyclone V SoC example, the video port of the Altera Video and Image Processing Suite can be instantiated to receive the data from the sensor. This data is then passed on to the image preprocessing block. After the data has been processed, it is written to the DDR memory.

Figure 6. Modules Involved in Low-Level Image Processing

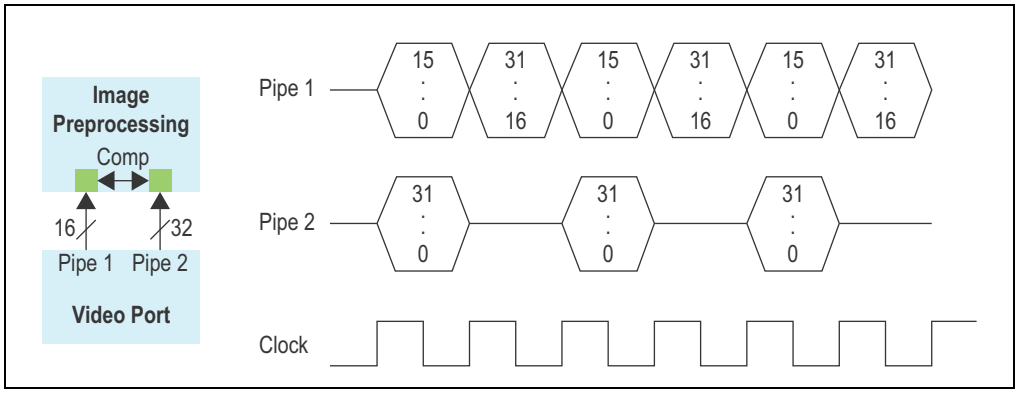


Most image sensors include a feature to transmit a defined test frame instead of the normal image data. As the input data is defined, the output data of the image preprocessing block is also defined. This is followed by a subsequent test. For example, a cyclic redundancy check (CRC) of the output data can find any permanent faults in the system. This test could cover permanent faults in the complete datapath. It allows testing of external connection problems, issues in the video port and image preprocessing block, and problems in the transmission of the data to the DDR memory. Another method to check for permanent faults is to implement a test pattern generator in the FPGA fabric, which could be multiplexed into the input path of the video port. Defined test patterns could cover many of the faults in the entire path.

Another feature of some image sensors is to transmit a frame counter with each frame. This feature helps to detect frame-to-frame gaps or even pixel clock or frame synchronization issues. In addition to this, the video port implements counters for pixels per line and number of lines. An interrupt will be generated if the counter does not correspond with the programmed number. This could also detect pixel clock and vertical and horizontal synchronization problems.

Data changes during transmission from one block in the FPGA should be detectable as well. The test pattern or test frame methodology mentioned above covers most of the permanent faults, but they will not detect transient faults. Transient faults could be caught by implementing a diverse image pipeline. When using the Altera Avalon® streaming protocol for the data transmission between modules, it is possible to instantiate two diverse interfaces, as shown in Figure 7, to ensure that a transient fault impacts the transmission differently in the two paths.

Figure 7. Example of Diverse Image Pipeline



Both diverse data streams then have to be compared at the endpoint to detect a potential problem.

In many designs, it may be necessary to also use memory buffers for temporary storage of data. For this case, it is possible to also instantiate parity or SECDED ECC, as the user memories in the FPGA fabric already provide necessary parity bits for the implementation.

When data is finally written to the DDR memory, a checksum of the data could be calculated on the fly while it is transmitted. This can be done for each line in the frame or each frame. In most cases, it may be done on the line because the later processing in the mid-level stage could check this data easier as the data may be fetched on a line or block basis. Altera provides a CRC engine that can be instantiated at different places in the user design and can generate CRC checksums on streaming data. Attaching a checksum to the generated data will also cover potential faults in the F2H bridge and DDR controller. The data in the DDR will then be protected by SECDED ECC and the checksum. Another potential fault to consider is that the address of the data could also be changed. To avoid overwriting other critical application data, the DDR memory controller implements a memory protection feature that allows it to define up to 20 different memory regions with different access permissions and which master has access to the specific region.

Table 2 provides a summary of the low-level image processing diagnostics.

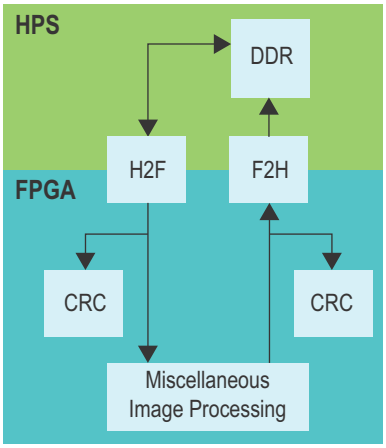
Table 2. Summary of Low-Level Image Processing Diagnostics

#	Potential Available Diagnostic
1	Image sensor test frame with subsequent CRC check of data in DDR memory
2	Test Pattern in video port input path with subsequent CRC check of data in DDR memory
3	Image sensor frame counter
4	Video port line length and frame length checking
5	Diverse image pipeline
6	Parity or ECC for FPGA user memories
7	CRC calculation on pre-processed data
8	Memory protection feature of DDR memory controller

Mid-Level Image Processing

As we discussed before, the miscellaneous image processing implements edge or corner detection algorithms and could also apply feature extraction algorithms. Therefore, the generated data set is reduced by looking at only the interesting features of an image. With the reduction in data comes also an increase in risk that a missed feature due to a fault could lead to a missed object at a higher processing step and thus increases the risk of a malfunction of the application. Figure 8 shows the modules involved in mid-level image processing.

Figure 8. Modules Involved in Mid-Level Image Processing



The data generated previously by the image preprocessing stage and stored in the external DDR memory needs to be read back for the miscellaneous image processing. Because a CRC checksum was attached with the data previously, it can be checked again to see if the data has been changed either during the time it was stored in the external memory or while it was being transferred to the FPGA.

There will most likely be some memory buffers implemented in this processing stage for temporary data storage. As we can instantiate again parity or SECDED ECC for these memories, we can achieve a relatively high diagnostic coverage.

To detect a fault in the logic itself, we could place two blocks of the same logic and lock-step it. In order to account for common-cause faults, we could use either different clock networks for both or run one logic block with, for example, a two-cycle clock delay. A successive comparison of the output of both blocks should be able to detect faults that occurred in one of the blocks.

Once new data is generated in this mid-level processing stage, it needs to be written again to the external DDR memory. Just like in the previous stage, we can also calculate a checksum over the new data and write this checksum to memory as well. The memory protection feature in the DDR memory controller will detect addressing faults and avoid overwriting other critical application data. Table 3 lists a summary of the mid-level image processing diagnostics.

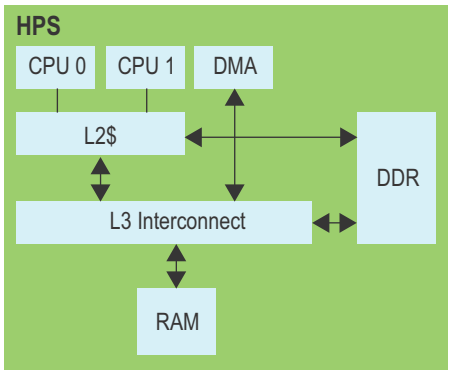
Table 3. Summary of Mid-Level Image Processing Diagnostics

#	Potential Available Diagnostic
1	CRC check of read data
2	CRC calculation of write data
3	Parity or ECC for FPGA user memories
4	Lock-step of logic
5	Memory protection feature of DDR memory controller

High-Level Image Processing

The high-level image processing stage includes the object detection and classification of objects. Speaking in software terms, this stage involves mainly control code so it is well-suited to run on a CPU. In our application example, this kind of functionality can be developed on the HPS. Figure 9 shows the modules involved in high-level image processing.

Figure 9. Modules Involved in High-Level Image Processing



Although two CPUs are implemented, they are not running in lock-step mode where one CPU checks the other automatically to detect faults during the program execution. If a fault happens in a CPU that is executing safety-critical code or data, the application needs to ensure that the result of the computation is checked. This can be done by redundantly executing the code either on the same CPU or on the second CPU. If a permanent fault developed in one CPU and the same code is executed twice on a single CPU, it is likely that the fault in the CPU will produce the wrong result twice and goes undetected. This problem could be mitigated by implementing two

diverse programs, which has, however, the drawback of increased development effort for implementing two different versions of the same program. Another option could be to run the redundant calculation on the second CPU. A permanent fault in one CPU may not affect the other CPU and thus a wrong execution could be detected without having to run diverse software as long as the fault is not in a shared resource. Running the algorithm twice with subsequent comparison of the result can also detect transient faults. In the case where software is executed on both CPUs, it is very likely that the same code is not executed exactly at the same cycle on both CPUs so a calculation followed by a comparison of the output of both redundant executions could catch the fault. The benefit of code executed at different times can be an effective measure against common cause faults.

As mentioned earlier, the memories involved in the processing are either protected by SECDED ECC or parity.

This leaves potential faults in the remaining logic of the L3 interconnect, DDR memory controller, on-chip RAM controller, and so on. For large data sets, a CRC checksum can be used to protect against changes in the data. As we've created a checksum for the data generated by the mid-level processing block, we can check this data before it is used for further operations. For smaller data sets, it may be sufficient to store the data and read it back to make sure it has been transferred correctly. For data that is read for processing, it can be read twice and compared before it is used. This should not create too big a processing overhead when the data size is limited, but can provide a good measure against potential faults.

Other features that can be used to detect faults are the memory management unit (MMU) in each Cortex-A9 processor, the memory protection feature in the DDR memory controller, and watchdog timers. We will have a closer look at the watchdog timers later in the document. Table 4 lists a summary of the high-level image processing diagnostics.

Table 4. Summary of High-Level Image Processing Diagnostics

#	Potential Available Diagnostic
1	ECC on DDR memory
2	ECC on Cortex-A9 processor L2 cache
3	Parity on Cortex-A9 processor L1 instruction and data cache, parity on branch prediction unit
4	Software diversified redundant calculation on one CPU
5	Redundant calculation on two CPUs with time delay
6	CRC checksum for large amount of data
7	Write and read-back mechanism for small set of data
8	Cortex-A9 processor MMU for memory protection
9	Memory protection feature in DDR memory controller
10	Watchdog timers

Miscellaneous Diagnostic Features

Apart from the processing stages that were previously discussed, it is necessary to also look at other features that are used in the application to increase diagnostics coverage. Some of those may have only minor impact on the hardware architectural metrics, but can be very critical to the overall application performance or functionality.

General-Purpose Timers

The general-purpose timers in the HPS are in many cases used to generate the ticks for the operating system. If only one timer is used, it could happen that a fault renders this timer inoperable, and this could mean that the operating system will not advance its state anymore. We can implement several possibilities to mitigate the risk of this happening.

One approach is to use two timer modules available in the HPS, where one creates the primary tick for the operating system and the other one creates a redundant interrupt. Both interrupts can be used to monitor each other and if one is not generated, it can be flagged accordingly, but the redundant interrupt guarantees that the operating system still has a time base. The HPS drives its timer modules with a single clock, so if this clock has an issue, it would affect both timers. To avoid this situation, either a redundant timer in the FPGA fabric that is clocked by a different clock source is implemented, or the FPGA implements a more sophisticated timer with redundancy and multiple interrupts routed to the HPS.

The simplest solution is to use a watchdog timer that may not be triggered when the operating system does not advance its state. Table 5 lists a summary of the general-purpose timer diagnostics.

Table 5. Summary of General-Purpose Timer Diagnostics

#	Potential Available Diagnostic
1	Use of two-timer modules in HPS for cross-checking of interrupts
2	Use of redundant timer in FPGA fabric with independent time base
3	Watchdog timer

Watchdog Timer

The HPS implements several different watchdog timers. There is one private watchdog timer for each Cortex-A9 processor and two global system watchdog timers. These watchdog timers are very simple implementations and will likely only achieve low diagnostic coverage. In addition, the watchdog timers are driven by the same clock as the rest of the HPS, so if there is a clock-related problem, it can affect the watchdog timers as well.

The application could implement a more sophisticated watchdog timer in the FPGA fabric. This application could include a time window in which it has to trigger the watchdog timer and logical monitoring of the program sequence. This could be implemented with a sequence of keys that have to be written in the correct order to clear the watchdog timer. Implementing the watchdog timer in the FPGA fabric has the added benefit that it can be driven by a separate clock source, and the fabric and HPS can also be supplied by different voltage sources. This will provide a strong mitigation against common-cause faults. Another possible method is to implement the watchdog timer in the external microcontroller with features to detect time and execution flow issues. Table 6 lists a summary of the watchdog diagnostics.

Table 6. Summary of Watchdog Diagnostics

#	Potential Available Diagnostic
1	Private watchdog timers in ARM Cortex-A9 processor
2	Global system watchdog timer in HPS
3	User implemented watchdog timer in FPGA fabric
4	Watchdog timer in external microcontroller

HPS Global Interrupt Controller

Interrupts are very critical in most control systems. You need to ensure that interrupts are generated and handled correctly. The HPS includes a global interrupt controller (GIC) that aggregates all interrupts in the HPS system. It also receives interrupts from the FPGA fabric and then routes them to the interrupt inputs of the Cortex-A9 CPUs. The GIC is only instantiated once so a potential fault could be difficult to identify if you do not implement diagnostics on the system level.

Most of the HPS peripheral interrupts are also available in the FPGA fabric. This allows you to design a monitoring and controlling circuit for the critical interrupts. A scheme, for example, could be to implement a watchdog timer that gets started by the interrupt and raises an error if it times out after a defined time. The interrupt service routine in the HPS that handles this interrupt would have to write to the watchdog timer to clear it before it times out. Another option is to instantiate the Altera soft Nios® II processor to monitor the interrupt.

If there are periodic interrupts, then another watchdog timer could be implemented to detect missing interrupts. For example, the watchdog timer can be used to monitor the periodic frame complete interrupt from the video port. The period of the watchdog timer needs to be longer than the period of the interrupt. Table 7 lists the summary of the GIC diagnostics.

Table 7. Summary of GIC Diagnostics

#	Potential Available Diagnostic
1	Interrupt watchdog timer in FPGA fabric with trigger capability from HPS
2	Interrupt monitoring by the Nios II processor
3	Interrupt watchdog timer for periodic interrupt

Clocks

Clocks are one of the most important parts of a component. Faults in clocks usually have severe impact on the behavior of an application. Fortunately several mechanisms can be used to detect potential clocking problems.

The Cyclone V SoC provides several clock inputs for the HPS and FPGA. These clocks can be routed between the HPS and FPGA fabric. Using different clocks for the HPS and FPGA can help with common-cause fault mitigation.

The phase-locked loops (PLLs) in the FPGA fabric also provide a feature to detect a missing input clock to the PLL for the primary input clock path. When a missing clock input is detected, there is the possibility to switch automatically to a clock connected to the secondary input port of the PLL.

Altera also provides a clock checker intellectual property (IP) specifically developed to IEC 61508 up to SIL3 capability. This clock checker IP is used for monitoring the frequency and presence of a clock signal against a stable reference clock. The user can then set up a high and low frequency threshold. An error flag is generated when the ratio of the two clocks falls out of these defined thresholds. This clock checker IP can be used to check the input clocks to the device, the PLL outputs, or any other clocks used in the application. A watchdog timer can also provide a detection mechanism for clock-related issues. Table 8 lists the summary of the clock diagnostics.

Table 8. Summary of Clock Diagnostics

#	Potential Available Diagnostic
1	Independent clock inputs on HPS
2	Independent clock inputs on FPGA
3	Independent clock inputs between HPS and FPGA
4	Missing input clock detection in FPGA PLL with automatic switchover to secondary input clock
5	Clock checker IP
6	Watchdog timer

Power Supply

As mentioned earlier, the HPS and FPGA fabric can be supplied with dedicated supplies for the core logic and also I/Os. Some of the supplies are internally monitored and will generate a reset when they are not in the normal operating range. It is advisable to implement external monitoring with proper voltage supervisors at least for the supplies that are not internally monitored. Monitoring with the analog-to-digital converter (ADC) of an external microcontroller should be avoided, as the ADCs are usually not fast enough to detect short brownout conditions that can lead to device malfunction. Table 9 lists the summary of the power supply diagnostics.

Table 9. Summary of Power Supply Diagnostics

#	Potential Available Diagnostic
1	Independent core logic and I/O supplies between HPS and FPGA
2	Partial internal voltage monitoring
3	External voltage monitors

Data Communication with External Components

In this application example, an external microcontroller is connected to the Cyclone V SoC. The data transmitted over this interface is highly safety critical. Most standard interfaces for this communication do not provide sufficient protection against communication errors. The transmission should be treated as a black channel with a higher-level protocol, ensuring the integrity of the data. This can be achieved with checksums over a set of data, challenge or response implementations for synchronization between sender and receiver, and others. Calculating a checksum has the added benefit that not only the transmission over the physical interface will be

covered, but also the internal transmission to or from the communication module and the communication module itself. Table 10 lists the summary of the external communication diagnostics.

Table 10. Summary of External Communication Diagnostics

#	Potential Available Diagnostic
1	Checksum over sent data
2	Challenge / response implementation

A detailed system analysis derived from a sound safety concept and requirements is essential to reduce the risk of a malfunctioning behavior of the system leading to a dangerous situation.

Systematic Fault Handling

All the above mentioned diagnostics can help to detect potential random hardware faults. Another important aspect of functional safety is to ensure systematic fault reduction. This is accomplished by using robust development processes and tools. The ISO26262 standard specifies in detail the requirements for the management of functional safety, such as the use of confirmation measures for the different activities during the safety lifecycle and supporting processes, like configuration and change management. Used tools need to be analyzed if they can contribute to potential failures of the application, and measures need to be put in place to mitigate the risk of this happening. A more detailed look into this topic goes beyond the scope of the white paper.

Conclusion

ADAS are the next wave of innovations to make driving on our more and more congested roads safer. These systems have performance requirements that challenge existing and future standard commercial-of-the-shelf (COTS) products and where the programmability of FPGAs can provide a tremendous advantage. Implementing application-specific diagnostics, like custom pattern generators or custom watchdog timers, which may not be possible to implement on a standard product, can increase the diagnostic coverage of the system. Stream processing in the initial stages of the image processing can reduce the potential for faults, reduce power consumption, and increase the application performance, due to the reduction in memory read or write transactions. Many COTS products have not been designed with functional safety in mind, and using a platform, development environment, and partner who has the right expertise in functional safety can benefit the overall implementation of the system.

References

- ISO26262:2011 Road Vehicles – Functional Safety, International Organization for Standardization (ISO)

Acknowledgements

- Frank Noha, Safety Specialist, System Solutions Engineering, Industrial and Automotive Business Unit

Document Revision History

Table 11 shows the revision history for this document.

Table 11. Document Revision History

Date	Version	Changes
November 2013	1.0	Initial release.