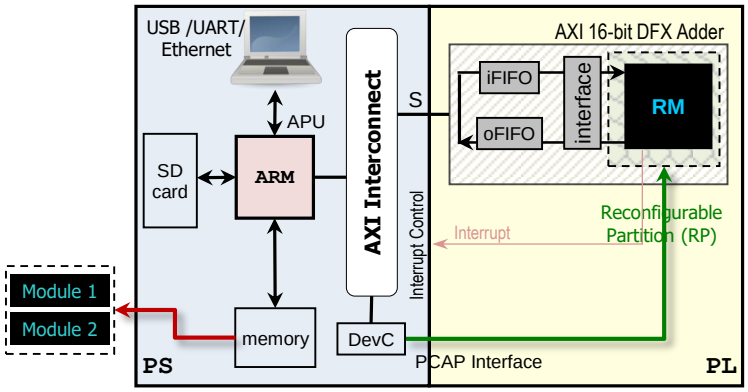
Notes - Unit 7

APPLICATIONS

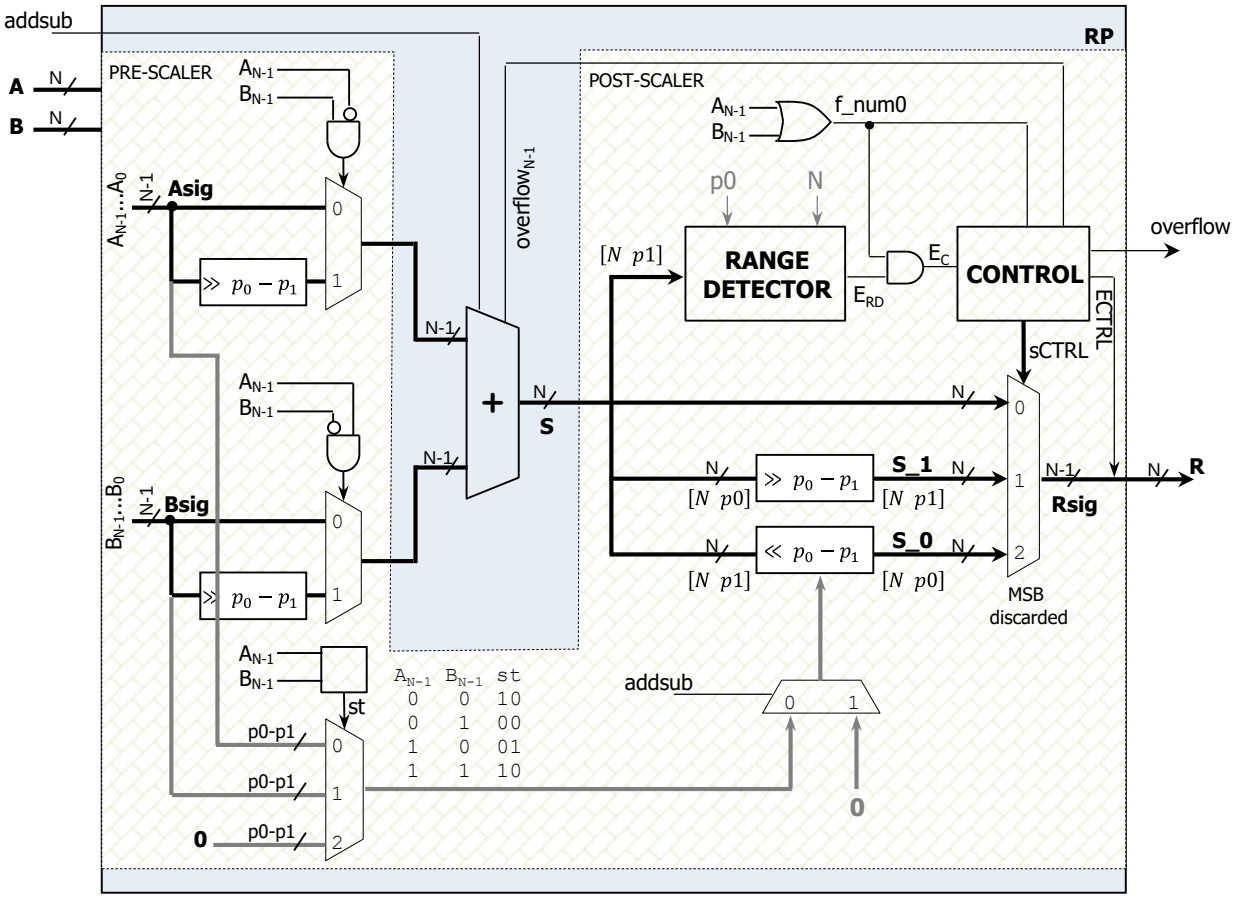
DYNAMIC ARITHMETIC: DYNAMIC DUAL FIXED-POINT ADDER

- Here, we use a 16-bit DFX Adder with overflow output. If overflow occurs, an interrupt is issued from the PL to the PS. The PS then detects the interrupt and alters (at run-time) the DFX format of the adder (by modifying p0 and p1) in order to avoid overflow. The figure depicts the self-reconfigurable embedded system.



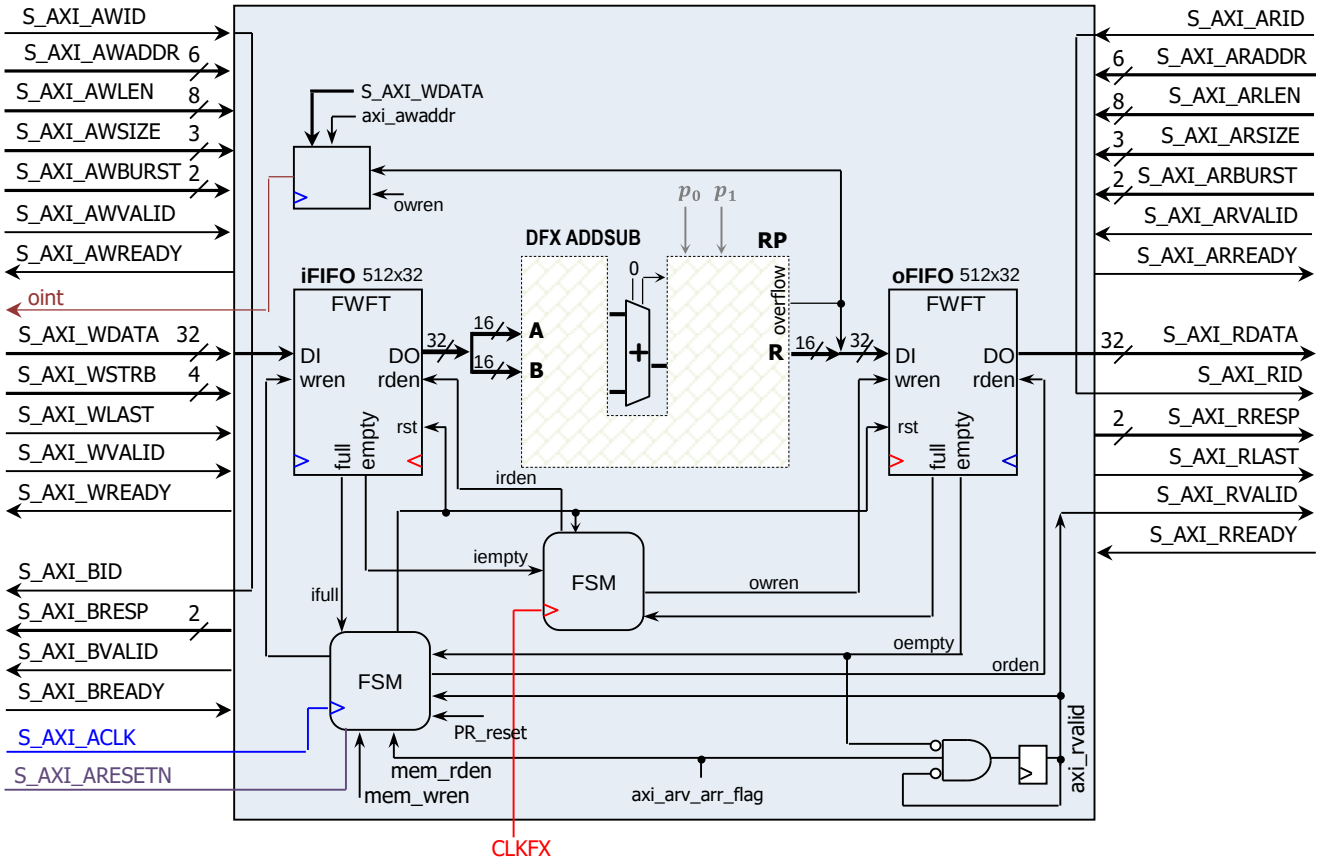
DUAL FIXED-POINT ADDER SUBTRACTOR

- This combinational circuit computes DFX addition/subtraction. We note that the FX adder/subtractor does not depend on p_0 and p_1. Thus, we can modify the DFX format (p_0, p_1) by modifying only the pre-scaler and post-scaler. The figure depicts how we partitioned the design into static and dynamic (run-time alterable) components.



AXI4-FULL INTERFACE

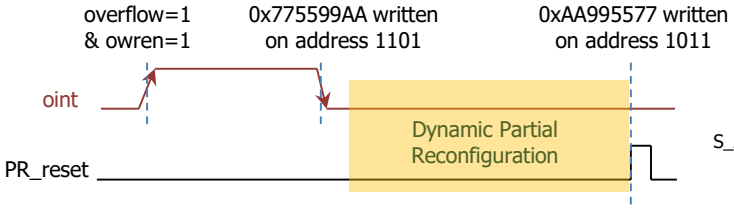
- For simplicity's sake, we set $N = 16$ and $addsub = 0$ for the DFX Adder/Subtractor. This effectively makes a 16-bit DFX adder. The figure below depicts the AXI4-Full interface. The interface also includes the output interrupt $oint$ signal.

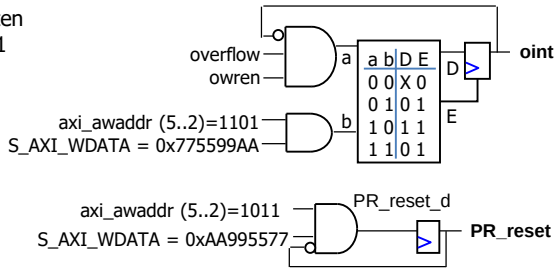


Interrupt and Partial Reconfiguration Control:

- The AXI4-Full interface generates an interrupt signal $oint$:
  - ✓ The $oint$ signal is asserted when an overflow is detected ($overflow = 1$) and when data is valid on the output of the DFX Adder (this happens when $owren = 1$, see FSM @ CLKFX).
  - ✓ The interrupt signal remains asserted until the PS detects it. At this point, the ISR de-asserts the interrupt signal (so that the signal does not continuously interrupt the PS). This is performed by writing a specific word (`0x775599AA`) on address 1101<u>00</u>. Make sure that when writing to this peripheral, we must avoid writing on address 1101<u>00</u>, otherwise it might write an undesired word (`0x775599AA`) on the iFIFO.

- After $oint$ is de-asserted, we are free to execute dynamic partial reconfiguration (DPR).
  - ✓ Usually, because of RP output toggling, the control signals of the FIFOs and the $oint$ block vary randomly. In this design, note that during DPR, $owren = 0$ (no word is written onto oFIFO); this prevents $oint$ from being altered unintendedly.
  - ✓ After Partial Reconfiguration, we have to reset the FIFOs (in case data was still written between the overflow and the start of DPR) and the PR FFs (nonexistent in this example). This is carried out by the signal PR_reset.
  - ✓ PR_reset: This signal (a pulse of one clock cycle) resets both the RP FFs (non-existent in this example) and the FIFOs via a simple software command (we write the word `0xAA995577` on address 1011<u>00</u>). Make sure that when writing to this peripheral, we must avoid writing on address 1011<u>00</u>, otherwise it might trigger an undesired PR_reset.

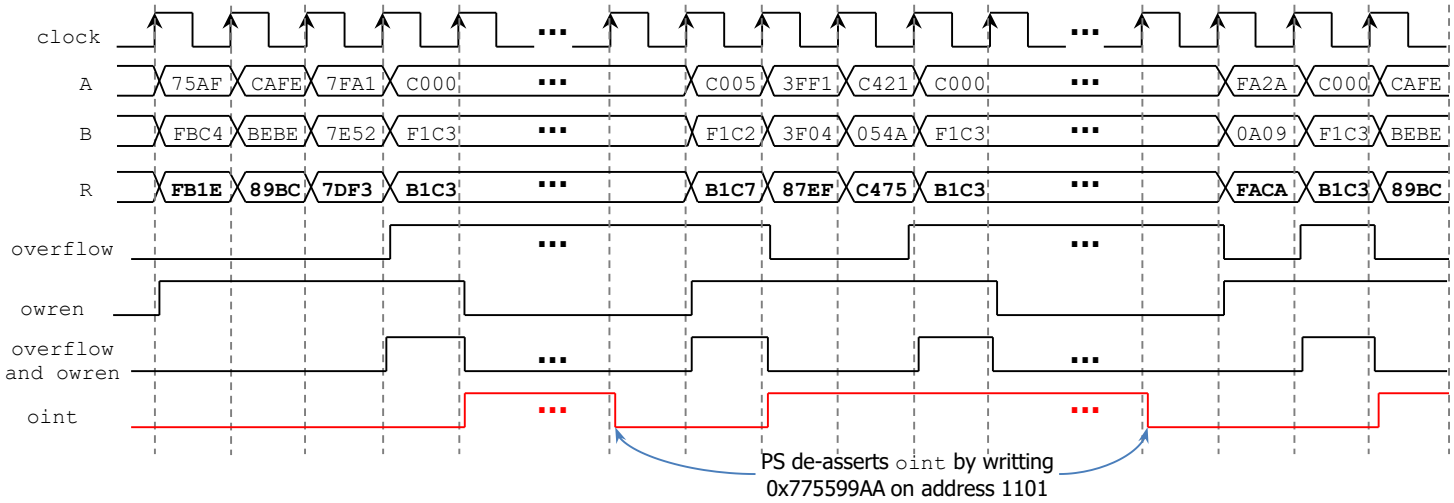




- Unlike the case of the Pixel Processor, here we cannot use AXI reads to de-assert the interrupt: If we issue an AXI read, we expect data from the FIFO to be read. If the FIFO is empty or if we read unintended data, the system will fail.

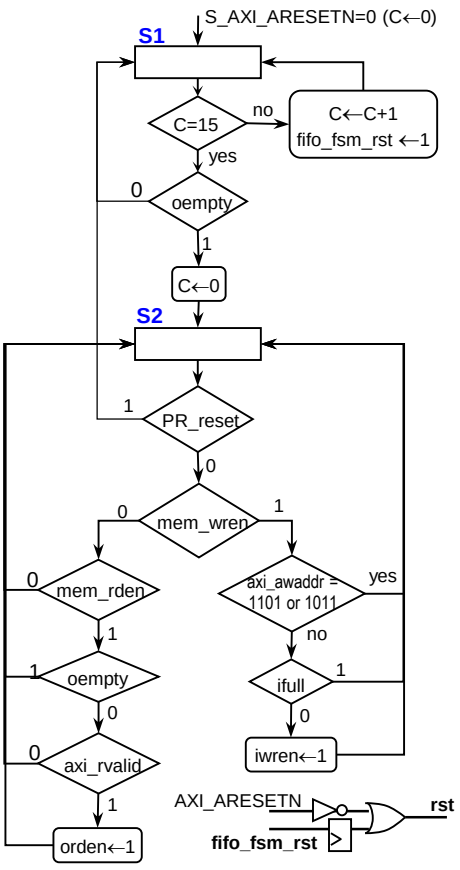
Timing diagram for $oint$

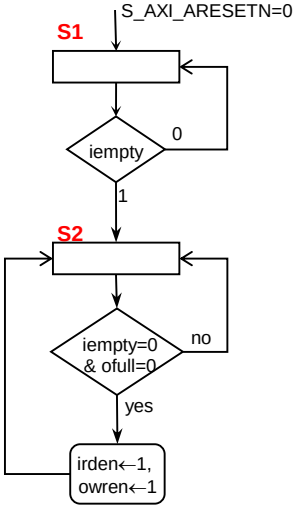
- We depict $oint$ assertion and de-assertion for a DFX adder in format [16 8 4]. If $overflow = owren = 1$ then $oint = 1$. During this time, other overflows are ignored. $oint$ is only de-asserted if we write `0x775599AA` on 110100.
- In the example, data is read and written in bursts of four 32-bit words. The inputs `A` and `B` fit in a 32-bit word, while the output `overflow&R` fit in a 32-bit word. The output is processed in one clock cycle and we can use $owren = 1$ to indicate valid data. Because of the design, the last data is kept on the inputs of the DFX adder until the next burst.
- We show a particular case when the last data in the first burst generates an overflow (note how $overflow = 1$ until the next burst includes a case with no overflow). Also, note how the first word of the second burst also generates an overflow. To detect this second overflow, we need $owren = 1$ (as $overflow = 1$ during this entire time). In general, to properly detect an overflow, we need: $overflow = owren = 1$.
- Note that when $oint$ is asserted, any subsequent overflow is ignored until $oint$ is de-asserted. In the second burst, the first and the third data generates and overflow, but only the first generates $oint$, an the second overflow is ignored.



AXI4-Full Interface Control

- The FSM @ S_AXI_ACLK is shown. As a precaution, we do not allow writes on iFIFO for addresses 101100 and 110100. For example, when de-asserting $oint$, we do not want the word `0x775599AA` to be written on iFIFO.
  - ✓ Since writes on address 101100 or 110100 are ignored, we have to be careful not to attempt to write on those addresses and then try to read as the system will freeze. This is straightforward when using simple write commands. But when using DMA, make sure that the BurstType is set to FIX (not to INCR or WRAP), otherwise the address will be automatically increased and it might reach 110100 or 101100, this will cause the system to ignore some words. This can be indicated in the DMA parameter DmaCmd→ChanCtrl.DstInc. By default it is 1 (Inc), make it 0 (Fix).
- We also depict the FSM @ CLK_FX. This is a very simple FSM as the DFX adder and its interface is purely combinational.
- Finally, note that when writing software applications that involve DMA (or large write loops), the challenge is to spot where the overflow occurred. Then if we reconfigure, we need to re-start processing from a certain point.





TESTING SCHEME

- We use two variations for the 16-bit DFX adder: formats [16 8 4] and [16 7 2].
- We show the 3 datasets (9 data points each) along with their DFX format:

FIRST DATASET

DFX format	Input A&B	Output overflow&R
[16 8 4]	0x75AFFBC4	0x0000FB1E
	0xCAFEBEBE	0x000089BC
	0x7FA17E52	0x00007DF3
	0x3FF13F04	0x000087EF
	0xC421054A	0x0000C475
	0xFA2A0A09	0x0000FACA
	0xC000F1C3	0x0001B1C3
	0xD001F170	0x0000C171
	0xFAF8300A	0x00005F8A

SECOND DATASET

DFX format	Input A&B	Output overflow&R
[16 7 2]	0x75AFFBC4	0x0000FB71
	0xCAFEBEBE	0x000089BC
	0x7FA17E52	0x00007DF3
	0x3FF13F04	0x000083F7
	0xC421054A	0x0000C44B
	0xFA2A0A09	0x0000FA7A
	0xC000F1C3	0x0001B1C3
	0xD001F170	0x0000C171
	0xFAF8300A	0x0000FC78

THIRD DATASET

DFX format	Input A&B	Output overflow&R
[16 7 2]	0x78D75E20	0x000056F7
	0xF2BF8FAF	0x0000826E
	0x7FD07F29	0x00007EF9
	0x1FF81F82	0x00003F7A
	0xF10802A5	0x0000F11D
	0x51500504	0x00005654
	0xF000FC70	0x0000EC70
	0xF400FC5C	0x0000F05C
	0x57C01805	0x00006FC5

Test Procedure

- File: test_dfxadd_rp.c, xtra_func.h
- The software routine works as follows:
  - ✓ It is assumed that the initial format is [16 8 4] (use full configuration for this).
  - ✓ With the DFX adder in format [16 8 4], the first dataset is tested. The results should match the shown output. An overflow will be generated by data C000 + F1C3. This will generate an interrupt. The ISR only de-asserts the interrupt signal oint. Once the 9 data points are processed (results retrieved), if the interrupt was issued, the software routine will reconfigure the DFX adder to the format [16 7 2].
  - ✓ With the DFX adder in format [16 7 2], the second dataset is tested. Note that these are the same binary values as in the case for [16 8 4], but data is treated as in the format [16 7 2]. We do this to demonstrate that we successfully performed reconfiguration (note that the output results are different). Here, overflow is also detected, an interrupt is issued, but reconfiguration is not performed.
  - ✓ With the DFX adder still in format [16 7 2], the third dataset is tested. These are the same real values of the first dataset, but represented in format [16 7 2]. Here, no overflow is generated.
  - ✓ Finally, we unconditionally reconfigure the DFX adder back to the format [16 8 4] and run the first data set. An overflow is detected, an interrupt is used, but reconfiguration is not performed.

- The VHDL code of this IP, the PR project structure, and the software application files are available at Tutorial: Embedded System Design for Zynq SoC - Unit 9.