GTM-IP Application note

AN021 – ARU Data routing mechanisms

Table of contents

1 Introduction

The Advanced Routing Unit (ARU) is one of the central modules of the GTM-IP. With the ARU it is possible to connect an arbitrary number of modules and their channels with each other in a flexible way. But due to the fact, that at each point in time only one data word can be routed through the ARU, there exists a dependency between the time data is provided or requested at the ARU and the real point in time, when the data flows through the ARU. In this application note the ARU data routing mechanism and its timing is described in more detail.
Section 2 describes the routing mechanism principles and constraints, while section 3 takes a closer look on the ARU timing constraints. Section 4 describes a more advanced routing scheme, which can be used to speed up and control the data transfer for dedicated data path within the GTM-IP.

2 ARU Operation principle

The ARU is used to transfer data within the GTM-IP between submodules in a flexible manner. The data can be measurement data, time stamps or control data. For the ARU the data has no meaning. It only becomes meaningful for the submodules generating or processing the data.
There are some naming conventions used throughout this document. These naming conventions are defined in Table 2.1.

Naming	Description
ARU Write channel	Data source.
ARU Read channel	Data destination.
Data stream	Connection between data source and data destination.
ARU Write address	Unique fixed address of data source within a GTM-IP module. The address is defined in HW and cannot be changed by software.
ARU Read address	Address, a data destination reads from. The ARU Read address is to be programmed by SW and corresponds to the ARU Write address of the data source which forms together with the data destination a data stream.
ARU RD-ID	Address of ARU Read channel, which is used by ARU. This value defines the order in which the ARU serves data destinations. The ARU RD-ID is defined in HW and cannot be changed by SW. The ARU RD-ID is of special interest in ARU dynamic routing mode, since it is used to identify slots which are served more frequently (see section 4 for more details).

Table 2.1: Naming conventions

The ARU routing mechanism can be summed up with a few basic rules of operation:

1. Data is transferred in a timely multiplexed way. This means, that in each ARU clock cycle, one data word can be transferred. The ARU data word layout is shown in Figure 2.1.
2. Exactly one route can be established between data source and data destination.
3. ARU Read channels have to request data from an ARU Write channel.
4. An ARU Write channel cannot actively transfer data to a destination.
5. ARU data word layout:

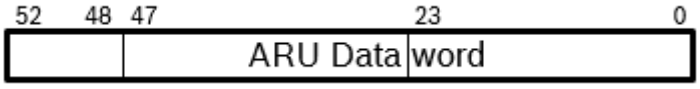

Figure 2.1: ARU Data word layout

6. When a destination reads data from a source, the transferred data is destroyed within the source.
7. If data has to be distributed to multiple destinations, a special submodule of the GTM-IP, the Broadcast Unit (BRC) has to be used.
8. ARU Blocking mechanism: modules requesting data from a source are blocked in their execution as long as there is no data available at the source module
9. ARU Write channel marks its data as invalid, when the data is transferred to an ARU Read channel. Thus, data is only transferred once from a data source to a data destination, and not available for other data destinations. It is therefore not recommended to have multiple ARU Read channels to read from the same ARU Write channel, since only one ARU Read channel would get the data, by chance.
10. The routing is established over GTM-IP cluster boundaries, thus data can be transferred between clusters in a flexible manner.

Figure 2.2 shows an example implementation with a central ARU. The numbers in circles denote ARU Write addresses, where the channel or module provides data to the ARU. The rectangles named ARU RDADDR are registers within a channel or module, where write addresses of ARU Write channels can be defined in software. The channel or module will than read data from this ARU Write channels.
As it can be seen, there can be submodules which implement ARU Write channels (MOD0) and ARU Read channels (MOD3) only. Other submodules can implement ARU Write and Read channels (MOD1 and MOD2). For the GTM-IP the MOD0 could be a TIM module, which samples incoming signals and provide them to the ARU. MOD1 could be a PSM which can consume data via ARU or provide data via ARU to other submodules of the GTM-IP.
The figure shows an example for a data stream. The data flows from the channel in submodule MOD0 to a channel in submodule MOD2. To establish this data stream, the ARU_RDADDR register in the MOD2 channel has to be programmed with the address, the channel in submodule MOD0 writes the data to. Write addresses are depicted as

numbers in circles in the figure. In the example, the channel of submodule MOD0 has the write address “1”. This write address is programmed into the register ARU_RDADDR and forms the ARU Read address for the channel of submodule MOD2.
With this mechanism, routes can be flexible programmed to serve different application needs. The figure also shows, that the ARU submodule itself has an ARU_RDADDR, and thus can be programmed to be receiver for data from a data source. This feature can be used for the dynamic routing mechanism describe later on in this application note.

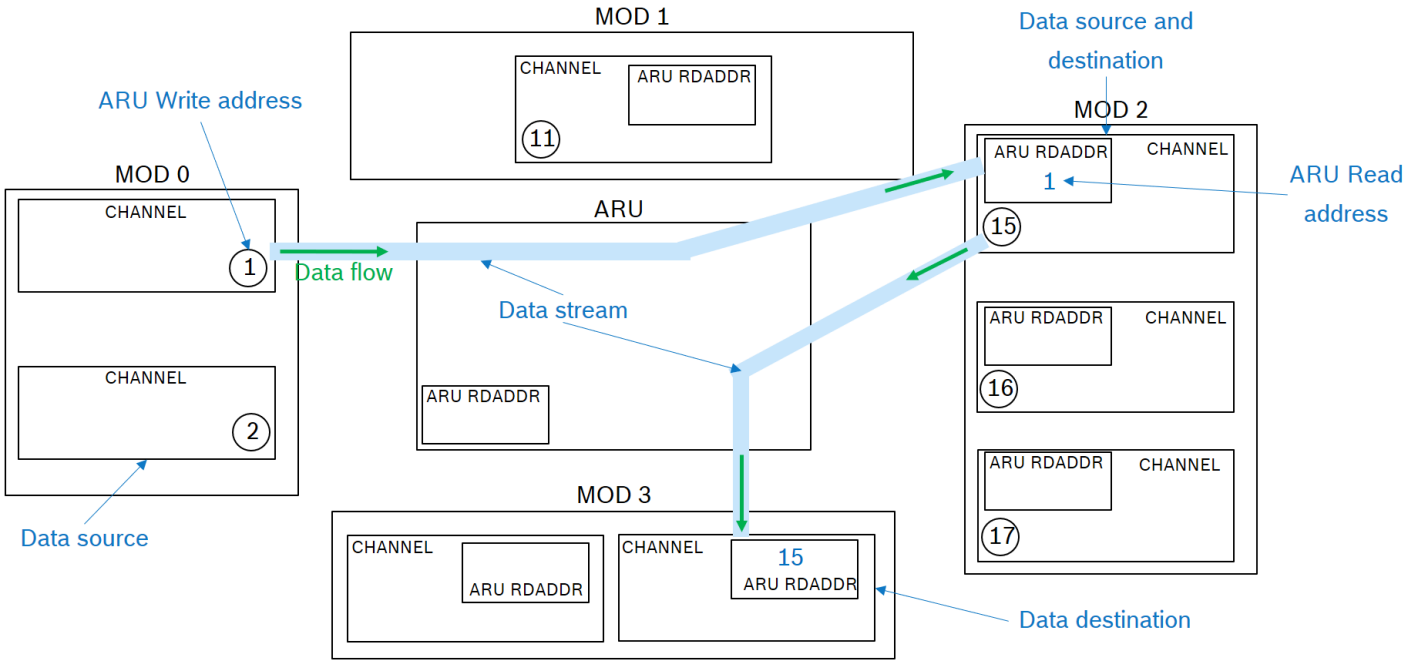

Figure 2.2: Example for ARU Routing mechanism

3 ARU Timing

3.1 ARU Routing scheme

For the ARU routing itself the ARU RD-IDs come into play. The routing scheme is shown in Figure 3.1. Each of the ARU Read channels has an ARU RD-ID, which is used by the ARU to address the channel. The ARU is implemented as a counter which is incremented every clock cycle and addresses therefore in each clock cycle another ARU Read channel. While addressing a Read channel, the ARU checks for a pending read request. If such a read request is pending, the ARU checks whether the addressed write channel has the corresponding write request pending. The data is transferred from the Write channel to the Read channel if both requests are present. A write request is raised by an ARU Write channel if new data for a consumer is available.

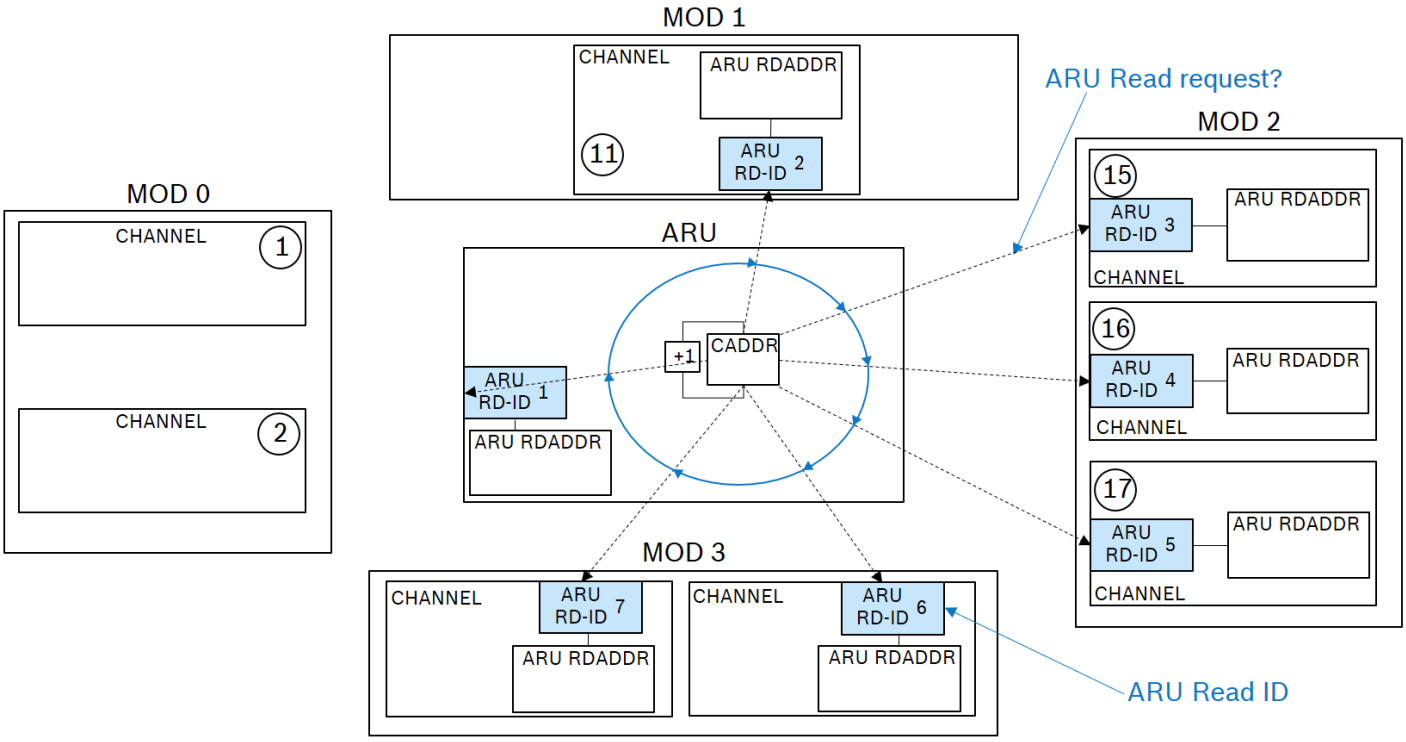

Figure 3.1: ARU Routing scheme

With this addressing scheme, each of the ARU Read channels is served in a cyclic manner by the ARU and is provided with data through the ARU, if there is valid data at the associated data source.

3.2 Timing consideration

3.2.1 ARU Round trip time

The cyclic serving of ARU Read channels by the ARU has to be taken into account, when designing applications running on or making use of the GTM-IP. The round robin scheme used for the ARU routing of data streams needs to be taken into consideration regarding data transfer times between a data source and a data consumer. For the example in Figure 3.1 there are seven ARU RD-IDs which have to be served by the ARU.
Besides these seven ARU RD-IDs there is a slot “0” for debugging purposes, which is also served by the ARU. Therefore, for the given example in Figure 3.1, the ARU asks each ARU Read channel in every 8th ARU clock cycle for a read request. Since there exists a two stage pipeline mechanism implemented for the data flow inside of the ARU there is an additional adder of two ARU clock cycles until the data is available at the ARU Read channel. This adder has to be considered, when the ARU Round trip time has to be calculated for a specific device.
In a typical GTM-IP scenario there are much more ARU Read channels to serve, than shown in the example of Figure 3.1. The total available number of ARU Read channels depends on the GTM-IP configuration and therefore the ARU Round trip time differs from GTM-IP to GTM-IP.
The maximum number of ARU Read channels of a GTM-IP can be determined by reading the register `GTM.ARU.CADDR_END` after reset. This register defines the value, where the internal ARU RD-ID address counter wraps around. By adjusting this register, the ARU Round trip time can be manipulated. This is described in more detail in subsection 3.2.2.
Table 3.1 show the layout of register `GTM.ARU.CADDR_END`.

31	30	29	29	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																									CADDR_END						
R																									RW						
0x000 000																									X						

Table 3.1: Register GTM.ARU.CADDR_END

With the information of register GTM.ARU.CADDR_END the ARU Round trip time (ARU_{RTT}) can be calculated with Equation 1. ACP defines the ARU Clock Period and is also device dependent.

$$ARU_{RTT} = \big((GTM.ARU.CADDR\_END + 1) + 2\big) * ACP$$

Equation 1: Calculation of ARU Round trip time

Since the register GTM.ARU.CADDR_END is seven bits wide this results in a maximum of 128 ARU RD-IDs and therefore in an overall maximum ARU Round trip time of (127+1)+2 ARU clock cycles. For a 100MHz ARU clock, this corresponds to a 1.3us round trip time. This results in a worst case frequency of roughly 760 kHz for serving repeatedly the same ARU RD-ID, assuming an ARU running at 100 MHz for such a device.
Coming back to the example from Figure 3.1, when the ARU is clocked with 100MHz this results in a round trip time of 100ns (((7+1)+2)*10ns) for device configuration shown there. Therefore, input signals with a maximum frequency of 10MHz can be routed within this device. If signals with a higher frequency should be routed, it is not guaranteed, that the data is always transferred. According to aforementioned Rule 9 of the ARU routing mechanism, data can be overwritten at the data source before it was transferred by the ARU.

3.2.2 Data transfer calculation for specific GTM-IP submodules

The ARU Round trip time defines the time data needs to flow from a source to a destination. This time can be increased further by additional register stages implemented in submodules connected to the ARU. If, for example, a PWM characteristic for an ATOM channel should be served via ARU, there is an additional adder of one clock cycle needed, when the data flows from a source via ARU to the ATOM channels shadow register and from there to the operation register.
Therefore, if a new PWM characteristic should be served via ARU to ATOM, the minimal period of the PWM is defined according to Equation 2.

$$PWM_{min} = ARU_{RTT} + (1 * ACP)$$

Equation 2: Minimal ATOM PWM period when using ARU as source for PWM parameters

If the PWM period is smaller than this time, it is not guaranteed that new PWM values arrive at the ATOM channel in time before the PWM has to be generated on behalf of these new parameters.

3.2.3 ARU Round trip timing adjustment by ARU.CADDR_END

Due to the configurable manner of the GTM.ARU.CADDR_END register, the data throughput can be increased, by defining a different value from the reset value. This is shown in Figure 3.2 for GTM.ARU.CADDR_END=4. With this configuration, the original ARU Round trip time of 100ns is shortened to 70ns (((4+1)+2)*10ns). On the other hand, the last channel of MOD2 and all the channels of MOD3 are not served by the ARU anymore and can therefore not receive and process data via ARU. Nevertheless, these modules can be used standalone, when configured and operated by the CPU.
When the register GTM.ARU.CADDR_END is misconfigured with a higher value than the number of connected ARU Read channels, the ARU round trip time gets longer, because the ARU tries to address virtual (non-existent) ARU Read channels.
Please note, that the register GTM.ARU.CADDR_END is write protected by bit RF_PROT of register GTM.CTRL.

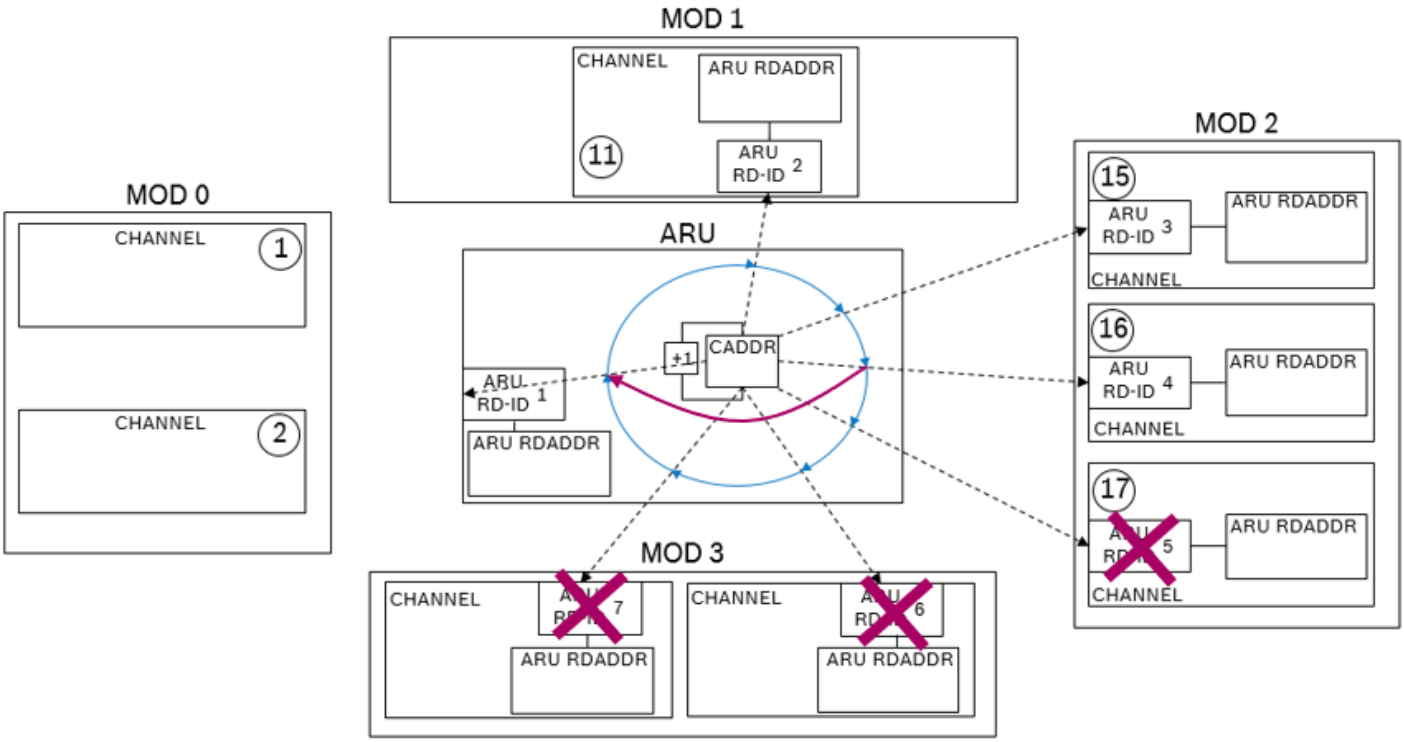

Figure 3.2: Shortening the ARU round trip time by manipulating the register GTM.ARU.CADDR_END

3.2.4 ARU Round trip timing adjustment using multiple ARU read ports

There is a second possibility to shorten the time, the ARU Read channels are served in. This is achieved by adding a second port to the ARU and connecting the submodules to either one or the other port of the ARU. This architecture is shown in Figure 3.3.
In our example the modules MOD0, MOD1, MOD2 and MOD3 are now served by a two-ported ARU. One port (blue circle) addresses the ARU Read channels of the modules MOD1 and MOD2, and the other port (green circle) addresses the ARU Read channels of the modules MOD0 and MOD3.
Due to this approach, the round trip is now just half of the original time, while still all ARU Read channels are served in a cyclic manner and all data sources can be reached by all data destinations.

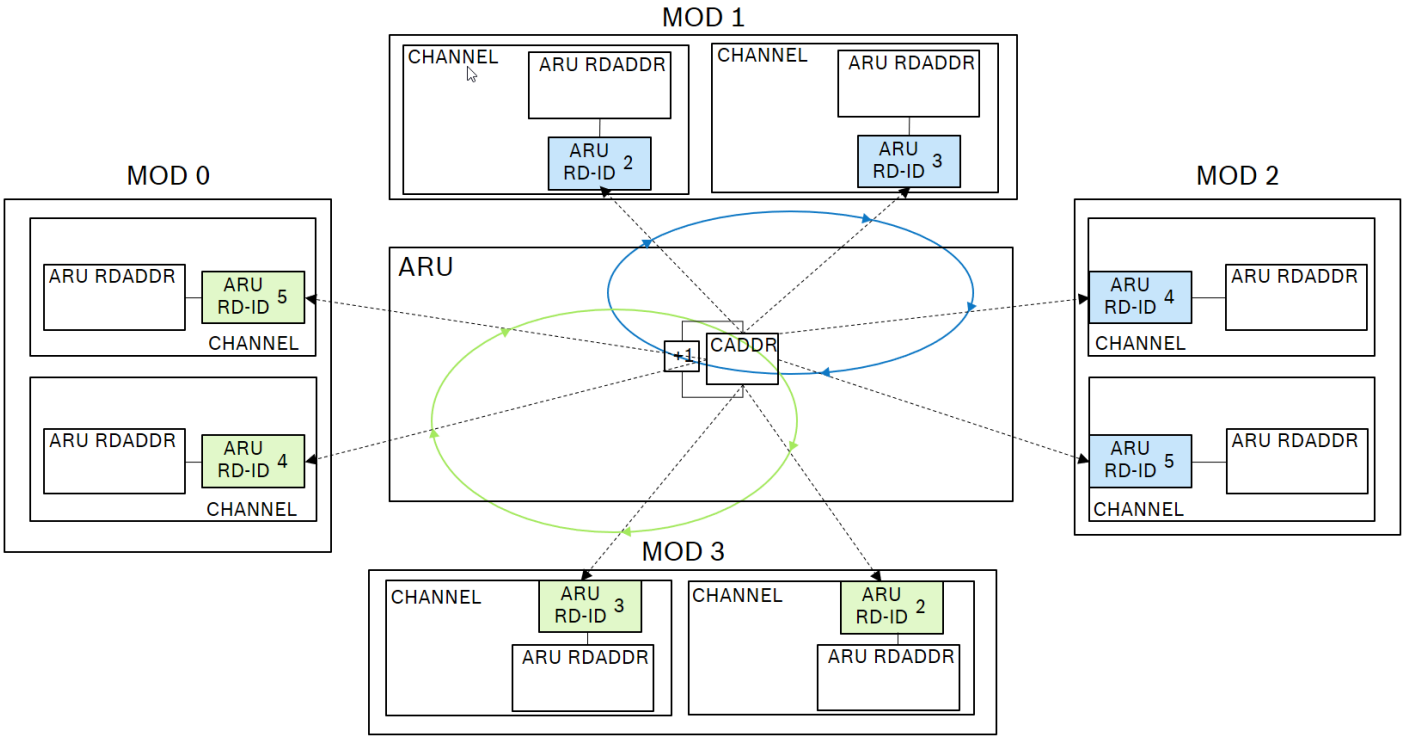

Figure 3.3: Shortening the ARU round trip time by adding additional port to ARU

Figure 3.4 shows the behaviour for two ARU ports with the counters *aru_caddr* and aru_*caddr2*, the register setting `GTM.ARU.CADDR_END=4` and a 100MHz ARU clock frequency.

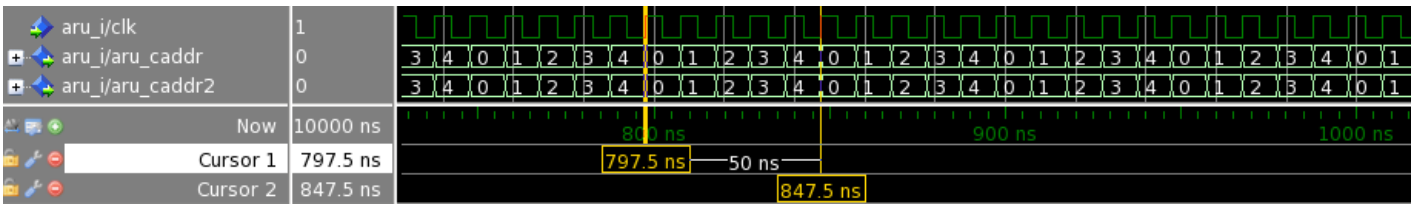

Figure 3.4: Shortening the ARU round trip time by adding additional port to ARU

Please note, that even if both address counters address the same ARU RD-ID, different ARU Read channels are addressed from each port.

4 ARU Dynamic routing

4.1 Overview

As mentioned in section 3, the ARU round trip time can be shortened by GTM-IP architectural measures and by changing the content of the register GTM.ARU.CADDR_END. Starting from GTM-IP Generation 3, there was a feature added that can be used to alter the behaviour of the ARU counter itself. The feature is called *ARU Dynamic routing* and can be used to route signals in a more flexible way between data sources and destinations.
The idea behind this feature is, to introduce additional slots in the ARU round trip, where user defined ARU RD-IDs are served in addition to their serving during the original ARU round trip. Registers to configure and control the ARU dynamic routing are:

- GTM.ARU.CTRL
- GTM.ARU.DYN_CTRL
- GTM.ARU.DYN_RDADDR
- GTM.ARU.DYN_ROUTE_HIGH
- GTM.ARU.DYN_ROUTE_SR_HIGH
- GTM.ARU.DYN_ROUTE_LOW
- GTM.ARU.DYN_ROUTE_SR_LOW

The registers GTM.ARU.CTRL and GTM.ARU.DYN_CTRL can be used to configure the behaviour of the ARU addressing. With the register GTM.ARU.CTRL the dynamic routing can be individually enabled for two different ARU ports. The register contains an additional bit to configure the *ARU Dynamic Routing Ring* mode. This mode is described in more detail in section 4.4. The register GTM.ARU.DYN_CTRL can be used to configure the usage and update mechanism for the four routing configuration registers GTM.ARU.DYN_ROUTE_HIGH, GTM.ARU.DYN_ROUTE_LOW, and their shadow registers GTM.ARU.DYN_ROUTE_SR_HIGH and GTM.ARU.DYN_ROUTE_SR_LOW.
The routing configuration swapping mechanism is described in more detail in section 4.3. The GTM.ARU.DYN_CTRL.DYN_ARU_UPDATE_EN enable bit, enables updating of the routing configuration bits via the ARU itself. This can be done for example by reloading the shadow registers by a FIFO or MCS channel. The source, where the route configuration is provided is defined with the register GTM.ARU.DYN_RDADDR. The reconfiguration procedure of the ARU routing behavior over the ARU itself is described in section 4.5.
The ARU RD-IDs, which should be inserted into the ARU routing scheme and the timing behavior of this insertion can be configured with the four DYN_ROUTE registers. Two registers hold three IDs each and have a shadow register which can be reloaded, while the ARU works on other ones. The ARU dynamic routing mechanism and its timing in general is described in the next section.

4.2 ARU dynamic routing foundations

This subsection should show the basic principles of the ARU dynamic routing mechanism. This is done by using different configurations of routes and showing the behaviour of the ARU addressing counters *aru_caddr* and *aru_caddr2*, when addressing the ARU RD-IDs. In the examples, the ARU end address is shortened to a value of ten, to show also the wrap around of the regular ARU addressing counter.

4.2.1 ARU Dynamic routing: One port, DYN_CLK_WAIT = 0

```
GTM.CTRL           = 0x0;    // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END  = 10;     // set caddr/caddr2 wrap around to ARU RD-ID 10
GTM.CTRL           = 0x1;    // enable RF_PROT again

GTM.ARU.DYN_ROUTE_LOW[0]  = (42 | (44<<8)|(46<<16)); // ATOM2_CH0 .. CH2
                                                     // ARU RD-IDs
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 | (50<<8)|(52<<16)); // ATOM2_CH3 .. CH5
                                                     // ARU RD-IDs
                                                     // DYN_CLK_WAIT = 0

GTM.ARU.CTRL = 0x2;   // enable ARU dynamic routing on ARU port 0 only
```

Code 1: ARU Dynamic routing: One port, DYN_CLK_WAIT = 0

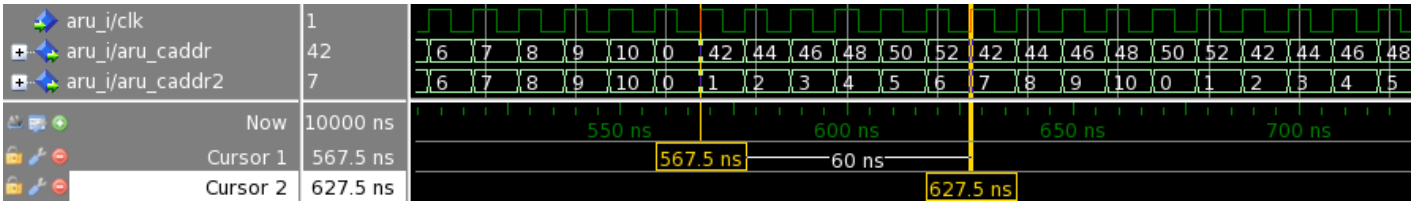

Figure 4.1: ARU Dynamic routing: One port, DYN_CLK_WAIT = 0

4.2.2 ARU Dynamic routing: One port, DYN_CLK_WAIT = 2

```
GTM.CTRL           = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END  = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL           = 0x1;     // enable RF_PROT again

GTM.ARU.DYN_ROUTE_LOW[0]  = (42 | (44<<8)|(46<<16)); // ATOM2_CH0 .. CH2
                                                     // ARU RD-IDs
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 | (50<<8)|(52<<16)   // ATOM2_CH3 .. CH5
                                 | (2<<24));         // ARU RD-IDs
                                                     // DYN_CLK_WAIT = 2

GTM.ARU.CTRL = 0x2;   // enable ARU dynamic routing on ARU port 0 only
```

Code 2: ARU Dynamic routing: One port, DYN_CLK_WAIT = 2

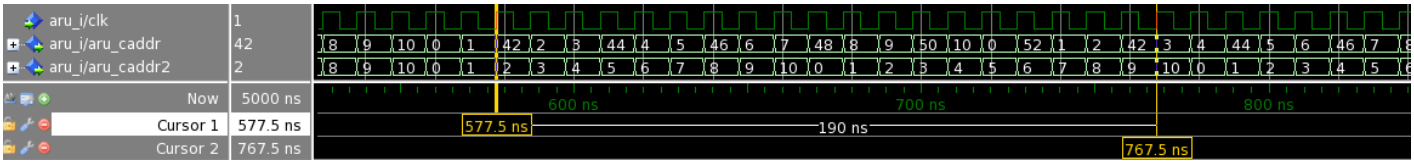

Figure 4.2: ARU Dynamic routing: One port, DYN_CLK_WAIT = 2

4.2.3 ARU Dynamic routing: One port, DYN_CLK_WAIT = 15

```
GTM.CTRL           = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END  = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL           = 0x1;     // enable RF_PROT again

GTM.ARU.DYN_ROUTE_LOW[0]  = (42 | (44<<8)|(46<<16)); // ATOM2_CH0 .. CH2
                                                     // ARU RD-IDs
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 | (50<<8)|(52<<16)   // ATOM2_CH3 .. CH5
                                 | (15<<24));        // ARU RD-IDs
                                                     // DYN_CLK_WAIT = 15

GTM.ARU.CTRL = 0x2;   // enable ARU dynamic routing on ARU port 0 only
```

Code 3: ARU Dynamic routing: One port, DYN_CLK_WAIT = 15

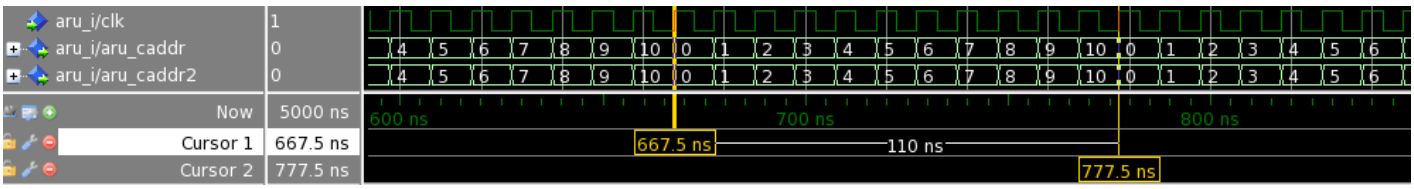

Figure 4.3: ARU Dynamic routing: One port, DYN_CLK_WAIT = 15

As it can be seen from Figure 4.3, when `DYN_CLK_WAIT` is set to 15, the ARU dynamic routing is disabled which means, that the ARU port 0 works in regular round robin mode serving the ARU RD-IDs up to `GTM.ARU.CADDR_END`.

4.2.4 ARU Dynamic routing: Two ports, DYN_CLK_WAIT = 2/4

```
GTM.CTRL           = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END  = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL           = 0x1;     // enable RF_PROT again

GTM.ARU.DYN_ROUTE_LOW[0]  = (42 |(44<<8)|(46<<16));    // ATOM2_CH0 .. CH2
                                                       // ARU RD-IDs
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 |(50<<8)|(52<<16)      // ATOM2_CH3 .. CH5
                                |(2<<24));             // ARU RD-IDs
                                                       // DYN_CLK_WAIT = 2
GTM.ARU.DYN_ROUTE_LOW[1]  = (66 |(68<<8)|(70<<16));    // ATOM3_CH0 .. CH2
GTM.ARU.DYN_ROUTE_HIGH[1] = (72 |(74<<8)|(76<<16)      // ATOM3_CH3 .. CH5
                                |(4<<24));             // DYN_CLK_WAIT = 4

GTM.ARU.CTRL = 0xA;   // enable ARU dynamic routing on ARU port 0 and 1
```

Code 4: ARU Dynamic routing: Two ports, DYN_CLK_WAIT = 2/4

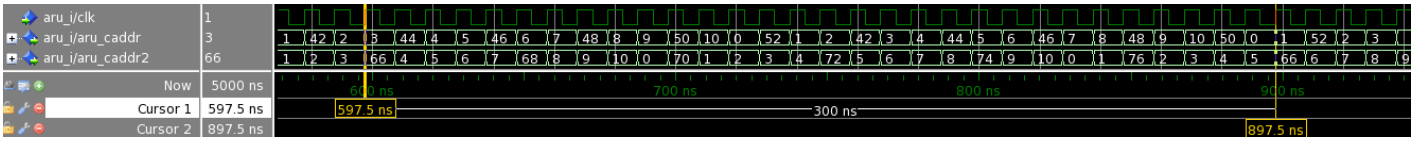

Figure 4.4: ARU Dynamic routing: Two ports, DYN_CLK_WAIT = 2/4

4.2.5 ARU Routing configuration update

This subsection shows the update mechanism of the ARU RD-IDs used by dynamic routing. The update is done via CPU at an arbitrary point in time. For an update of the ARU RD-IDs via ARU please refer to section 4.5.

```
GTM.CTRL           = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END  = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL           = 0x1;     // enable RF_PROT again

GTM.ARU.DYN_ROUTE_LOW[0]  = (42 |(44<<8)|(46<<16));    // ATOM2_CH0 .. CH2
                                                       // ARU RD-IDs
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 |(50<<8)|(52<<16));    // ATOM2_CH3 .. CH5
                                                       // ARU RD-IDs
                                                       // DYN_CLK_WAIT = 0
GTM.ARU.CTRL = 0x2;   // enable ARU dynamic routing on ARU port 0

/*
 * Do some some stuff here → than update the route
 */
GTM.ARU.DYN_ROUTE_SR_LOW[0]  = (26 |(28<<8)|(30<<16));    // ATOM0_CH0 .. CH2
GTM.ARU.DYN_ROUTE_SR_HIGH[0] = (32 |(34<<8)|(36<<16)      // ATOM0_CH3 .. CH5
                                   |(1<<24)|(1<<28));     // DYN_CLK_WAIT = 1
                                                          // DYN_UPDATE_EN = 1
```

Code 5: ARU Dynamic routing: One port, DYN_CLK_WAIT = 0, DYN_CLK_WAIT 1

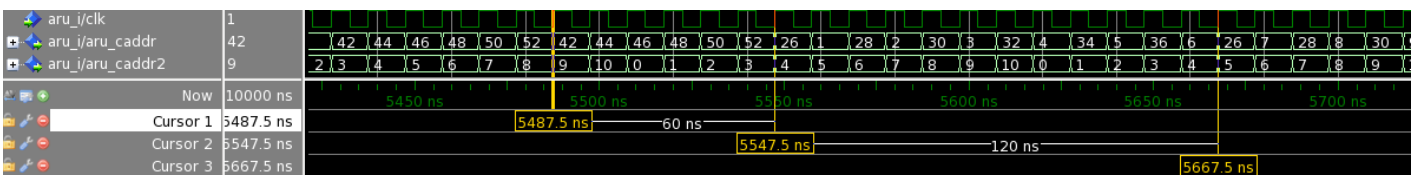

Figure 4.5: ARU Dynamic routing: One port, DYN_CLK_WAIT = 0, DYN_CLK_WAIT 1

4.3 ARU Dynamic routing swap mechanism

With the aforementioned configurations it is possible to insert six specific ARU RD-IDs per ARU port which are served more frequently in comparison to the regular round robin addressing scheme. When more than six ARU RD-IDs are needed, it is possible to update the routing configuration by using the shadow registers manually, or the ARU dynamic

routing swapping feature can be used. This allows twelve ARU RD-IDs to be inserted into the regular routing. Configuration of the swapping feature is shown in Code 6 and Figure 4.6.

```
GTM.CTRL              = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END     = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL              = 0x1;     // enable RF_PROT again
GTM.ARU.DYNC_CTRL[0] = 0x2;      // enable swap mechanism
GTM.ARU.DYN_ROUTE_LOW[0]  = (42 |(44<<8)|(46<<16));      // ATOM2_CH0 .. CH2
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 |(50<<8)|(52<<16)        // ATOM2_CH3 .. CH5
                                 |(1<<24));              // DYN_CLK_WAIT = 1
GTM.ARU.DYN_ROUTE_SR_LOW[0]  = (26 |(28<<8)|(30<<16));   // ATOM0_CH0 .. CH2
GTM.ARU.DYN_ROUTE_SR_HIGH[0] = (32 |(34<<8)|(36<<16));   // ATOM0_CH3 .. CH5
                                                         // DYN_CLK_WAIT = 0
GTM.ARU.CTRL = 0x2;   // enable ARU dynamic routing on ARU port 0
```

Code 6: ARU Dynamic routing, swap mechanism: One port, DYN_CLK_WAIT = 0, DYN_CLK_WAIT 1

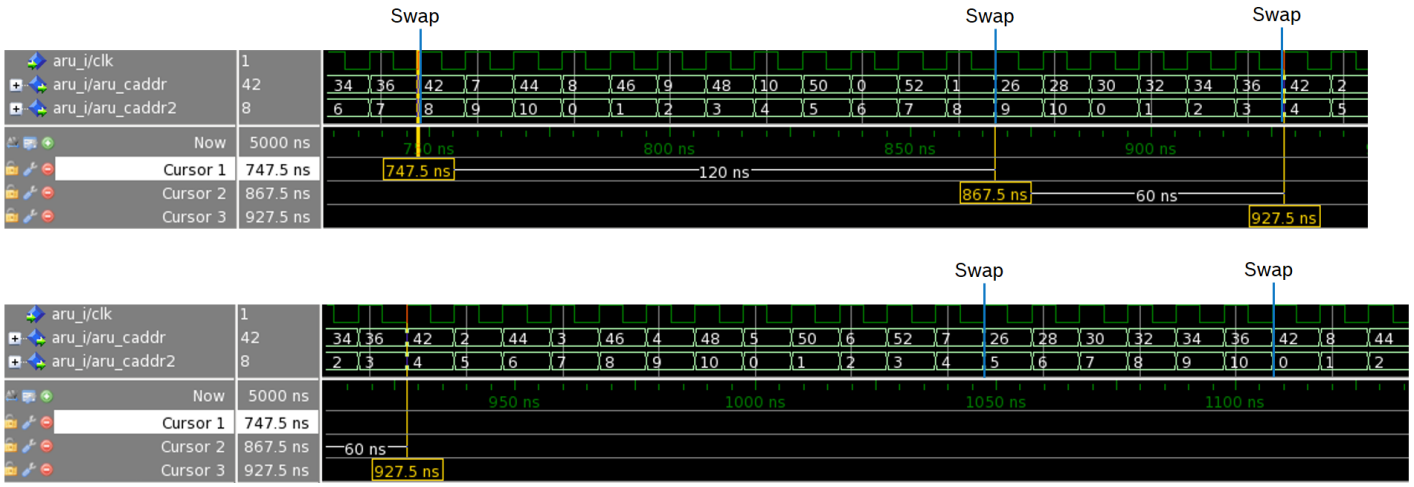

Figure 4.6: ARU Dynamic routing, swap mechanism: One port, DYN_CLK_WAIT = 0, DYN_CLK_WAIT 1

4.4 ARU Dynamic routing ring mode

Subsection 4.3 described, how the six dynamic routing ARU RD-IDs can be extended by their six shadow register counterparts to form twelve privileged ARU RD-IDs. These twelve ARU RD-IDs can be extended even further by the twelve ARU RD-IDs of the second ARU port. This can be done by enabling the ARU dynamic routing ring mode in register GTM.ARU.CTRL, and specifying the additional ARU RD-IDs.
It is important to note, that the ARU dynamic routing ring mode can be used for one or both ARU ports, while the other one uses the ARU round robin addressing scheme. The configuration code and timing using one ARU port is shown in the following code snippet and Figure 4.7.

```
GTM.CTRL              = 0x0;     // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END     = 10;      // set caddr overflow to ARU RD-ID 10
GTM.CTRL              = 0x1;     // enable RF_PROT again
GTM.ARU.DYN_ROUTE_LOW[0]  = (42 |(44<<8)|(46<<16));     // ATOM2_CH0 .. CH2
GTM.ARU.DYN_ROUTE_HIGH[0] = (48 |(50<<8)|(52<<16)       // ATOM2_CH3 .. CH5
                                |(1<<24));              // DYN_CLK_WAIT = 1
GTM.ARU.DYN_ROUTE_SR_LOW[0]  = (26 |(28<<8)|(30<<16));  // ATOM0_CH0 .. CH2
GTM.ARU.DYN_ROUTE_SR_HIGH[0] = (32 |(34<<8)|(36<<16));  // ATOM0_CH3 .. CH5
                                                        // DYN_CLK_WAIT = 0
GTM.ARU.DYN_ROUTE_LOW[1]     = (66 | (68<<8)|(70<<16));  // ATOM3_CH0 .. CH2
GTM.ARU.DYN_ROUTE_HIGH[1]    = (72 | (74<<8)|(76<<16));  // ATOM3_CH3 .. CH5
GTM.ARU.DYN_ROUTE_SR_LOW[1]  = (35 | (37<<8)|(39<<16));  // ATOM1_CH0 .. CH2
GTM.ARU.DYN_ROUTE_SR_HIGH[1] = (41 | (43<<8)|(45<<16)    // ATOM1_CH3 .. CH5
                                   | (1<<24));
GTM.ARU.CTRL = 0x12;   // enable ARU dynamic routing ring mode on ARU port 0
```

Code 7: ARU Dynamic routing ring mode

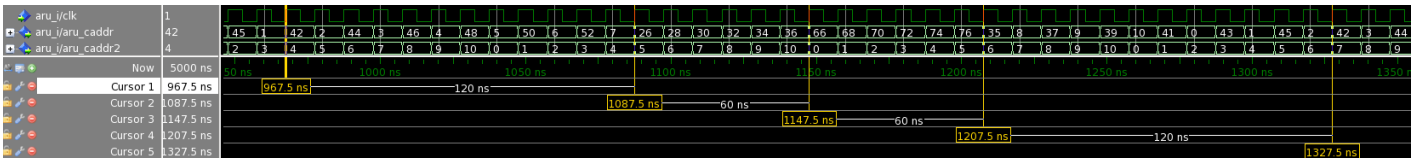

Figure 4.7: ARU Dynamic routing ring mode

Figure 4.8 shows the same addressing scheme, when both ARU ports work in ARU dynamic routing ring mode. This is done by configuring GTM.ARU.CTRL = 0x1A.

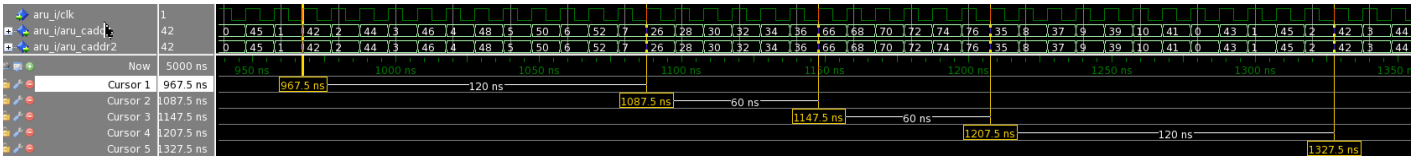

Figure 4.8: ARU Dynamic routing ring mode on two ARU ports

4.5 ARU Dynamic routing update mechanism via ARU

4.5.1 Basic concepts

Subsection 4.2.5 described the update of the dynamic routing ARU RD-IDs with CPU accesses to the corresponding shadow registers GTM.ARU.DYN_ROUTE_SR_LOW/HIGH. The ARU dynamic routing mechanism provides also the possibility to update those shadow registers via the ARU itself. Therefore, the ARU has it's own ARU RD-ID and ARU read address register GTM.ARU.DYN_RDADDR.
Figure 4.9 shows an excerpt of an example ARU RD-ID table, wherein the ARU RD-ID of the ARU is 1 for both ARU ports (ARU-0 and ARU-1). This ARU RD-ID is inserted into the dynamic routing mechanism as an additional addressing slot, which is served more regularly as with the round robin scheduling.

ARU read ID (dec)	ARU-0	ARU-1
0	reserved	reserved
1	**ARU-0**	**ARU-1**
2	**BRC** channel 0	**DPLL** action0
3	**PSM-0** channel 0	**PSM-1** channel 0
4	BRC channel 1	DPLL action 1
5	PSM-0 channel 1	PSM-1 channel 1
6	BRC channel 2	DPLL action 2

Figure 4.9: Example of an ARU RD-ID table

The reserved ARU RD-ID "0" is used for accessing the ARU routing from CPU injecting data into the router or getting data out of the router. This is done by using the ARU registers ARU_ACCESS, ARU_DATA_H, and ARU_DATA_L. The usage of this mechanism is described in more detail in section 5.
Figure 4.10 shows the ARU dynamic routing scheme on port ARU-0 when the ARU dynamic update feature is disabled. Since there is no update of dynamic routing shadow registers, there is no need to ask the ARU itself if an update of its shadow registers is requested.

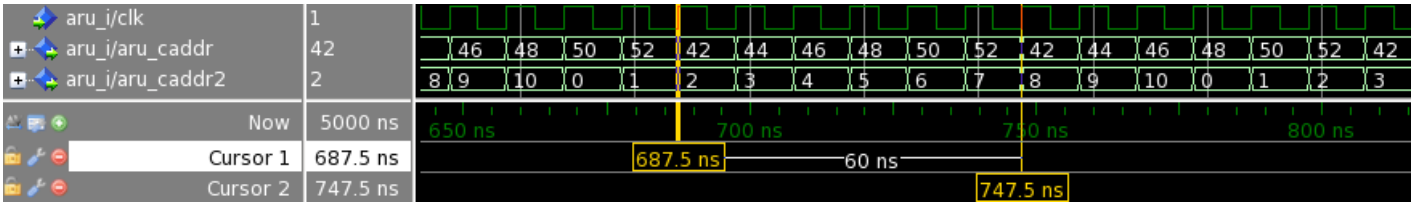

Figure 4.10: ARU Dynamic routing, with DYN_CLK_WAIT=0 and ARU update mechanism disabled

Figure 4.11 shows the same routing configuration, but here the ARU dynamic update bit in register GTM.ARU.DYN_CTRL is enabled. The value of DYN_CLK_WAIT is set to 0 in this case.

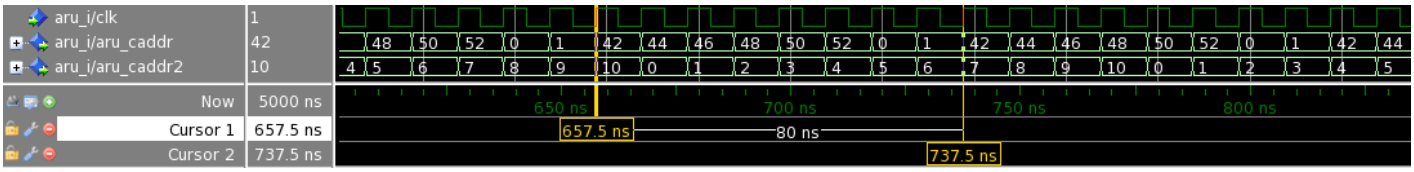

Figure 4.11: ARU Dynamic routing, with DYN_CLK_WAIT=0 and ARU update mechanism enabled

Figure 4.12 shows the addressing scheme, when the ARU update mechanism is enabled on port ARU-0 and DYN_CLK_WAIT=1. Please note the additional insertion of ARU RD-ID "1" after the RD-IDs in registers GTM.ARU.DYN_ROUTE_LOW/HIGH are served.

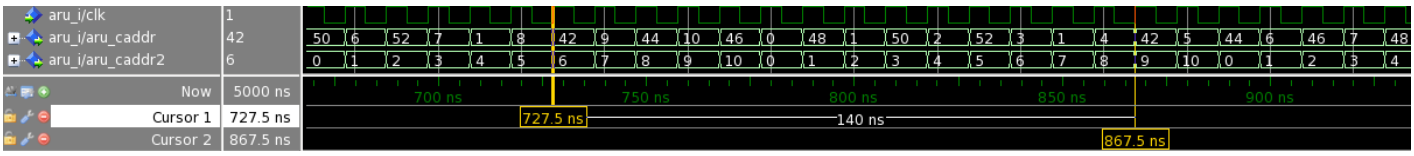

Figure 4.12: ARU Dynamic routing, with DYN_CLK_WAIT=1 and ARU update mechanism enabled

The examples in Figure 4.11 and Figure 4.12 should only show the ARU addressing behaviour, when the ARU update mechanism is enabled. For these two examples, the ARU RD-IDs are not updated, since there is no data source connected. The following subsections show, how data sources for the ARU RD-IDs can be configured, to provide new ARU RD-IDs.

4.5.2 Automatic update of ARU dynamic routing scheme via ARU and connected PSM

The following code example shows the setup of the ARU dynamic routing mechanism, when the addressing scheme should be reloaded via ARU and the routing schedule is located in a Parameter Storage Module (PSM). In this example the PSM is configured in Ring Buffer Mode. This means, that the data is provided to the ARU continuously in a never ending loop. Therefore, the ARU routing scheme is continuously changed. This helps to emulate the ARU dynamic routing ring mode, but in a new and very flexible manner, since the storage of the ARU RD-IDs in PSM submodules gives the freedom to define more than 24 ARU RD-IDs for dynamic routing.

```
/* Configure PSM to operate in Ring Buffer mode and provide data to ARU */
GTM.PSM[0].FIFO.CH[0].CTRL     = 0x1;        // ring buffer mode
GTM.PSM[0].FIFO.CH[0].END_ADDR = 7;          // use 8 entries (8x3=24RD-IDs)
GTM.PSM[0].F2A.CH_STR_CFG[0]   = 0x00060000; // provide 53 bits of data to ARU
GTM.PSM[0].F2A.ENABLE          = 0x2;        // enable channel 0

/* Configure ARU routing scheme by filling PSM */
// 1. ARU Addressing scheme
GTM.PSM[0].AFD.CH[0].BUF_ACC = (42 | (44<<8)|(46<<16)); // ATOM2_CH0 .. CH2
GTM.PSM[0].AFD.CH[0].BUF_ACC = (48 | (50<<8)|(52<<16)   // ATOM2_CH3 .. CH5
                                   | (1<<24)|(1<<28));  // DYN_CLK_WAIT = 1
                                                        // DYN_UPDATE_EN = 1
// 2. ARU Addressing scheme
GTM.PSM[0].AFD.CH[0].BUF_ACC = (26 | (28<<8)|(30<<16)); // ATOM0_CH0 .. CH2
GTM.PSM[0].AFD.CH[0].BUF_ACC = (32 | (34<<8)|(36<<16)   // ATOM0_CH3 .. CH5
                                   | (0<<24)|(1<<28));  // DYN_CLK_WAIT = 0
                                                        // DYN_UPDATE_EN = 1
// 3. ARU Addressing scheme
GTM.PSM[0].AFD.CH[0].BUF_ACC = (66 | (68<<8)|(70<<16)); // ATOM3_CH0 .. CH2
GTM.PSM[0].AFD.CH[0].BUF_ACC = (72 | (74<<8)|(76<<16)   // ATOM3_CH3 .. CH5
                                   | (1<<24)|(1<<28));  // DYN_CLK_WAIT = 1
                                                        // DYN_UPDATE_EN = 1
// 4. ARU Addressing scheme
GTM.PSM[0].AFD.CH[0].BUF_ACC = (35 | (37<<8)|(39<<16)); // ATOM1_CH0 .. CH2
GTM.PSM[0].AFD.CH[0].BUF_ACC = (41 | (43<<8)|(45<<16)   // ATOM1_CH3 .. CH5
                                   | (0<<24)|(1<<28));  // DYN_CLK_WAIT = 0
                                                        // DYN_UPDATE_EN = 1

/* Configure ARU dynamic routing */
GTM.CTRL              = 0x0;    // disable RF_PROT, to write CADDR_END address
GTM.ARU.CADDR_END     = 10;     // set caddr overflow to ARU RD-ID 10
GTM.CTRL              = 0x1;    // enable RF_PROT again
GTM.ARU.DYNC_CTRL[0]  = 0x1;    // enable routing scheme update via ARU
GTM.ARU.DYN_RDADDR[0] = 0x051;  // read from PSM channel 0
GTM.ARU.CTRL          = 0x2;    // enable ARU dynamic routing on ARU port 0
```

Code 8: ARU Dynamic routing with ARU update mechanism via PSM

The result of example code 8 is shown in Figure 4.13. The waveform shows the behaviour of the ARU at the beginning of the code, when the ARU was just configured. Since there is no data yet loaded into the `GTM.ARU.DYN_ROUTE_LOW/HIGH` registers, the ARU performs a dynamic routing of address 0x0, since the reset value of registers `GTM.ARU.DYN_ROUTE_LOW/HIGH` is 0x0. This is done within six ARU clock cycles, representing the six DYN_READ_IDs of those registers.
After the addressing of ARU RD-ID “0”, ARU RD-ID “1” follows. This is the RD-ID of the ARU itself (see Figure 4.9), and a loading of the first ARU addressing scheme from PSM module is performed. The data is loaded into the `GTM.ARU.DYN_ROUTE_SR_LOW/HIGH` registers first. Thus, a second cycle is needed, to load the data into the `GTM.ARU.DYN_ROUTE_LOW/HIGH` registers.
Please note, that the ARU RD-ID “1” is served within the regular addressing scheme and after the addresses in the registers `GTM.ARU.DYN_ROUTE_LOW/HIGH` are served (the Reload-slots).

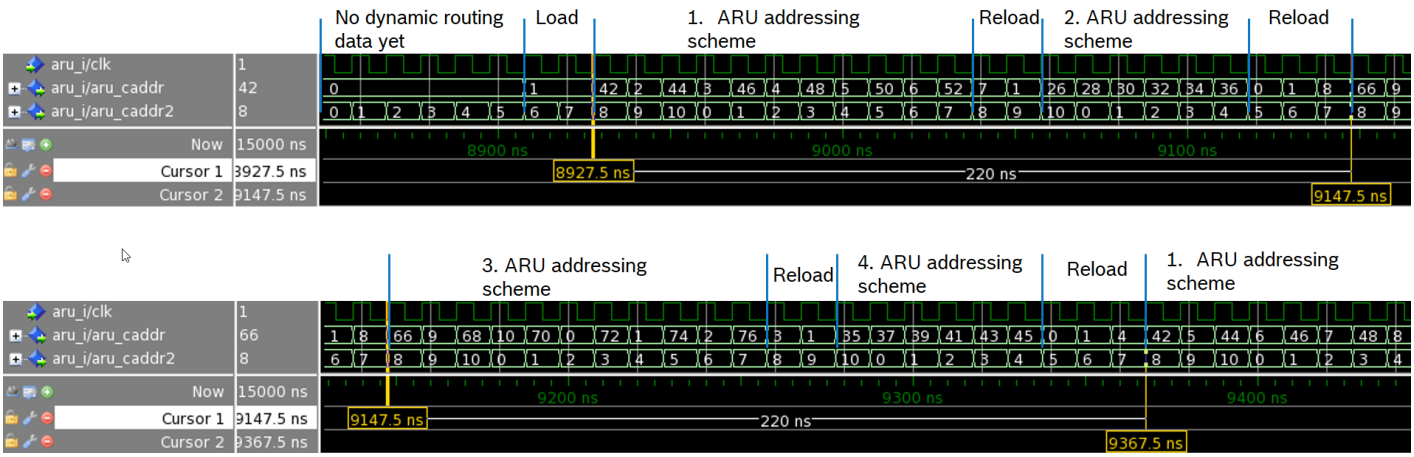

Figure 4.13: ARU Dynamic routing with ARU update mechanism via PSM

4.5.3 Timing considerations when updating via ARU

The GTM-IP architecture is designed to speed up the ARU round trip times by introducing two ARU read ports ARU-0 and ARU-1. Both ports serve different ARU Read channels in the same clock cycle. On the other hand, these ARU Read channels can request data from channels located within the same module or even from the same ARU Write channel. This is due to the configurable manner of the read address register of the ARU Read channel, which does not prevent to address the same ARU Write channel from multiple ARU Read channels. This configuration is shown in Figure 4.14.

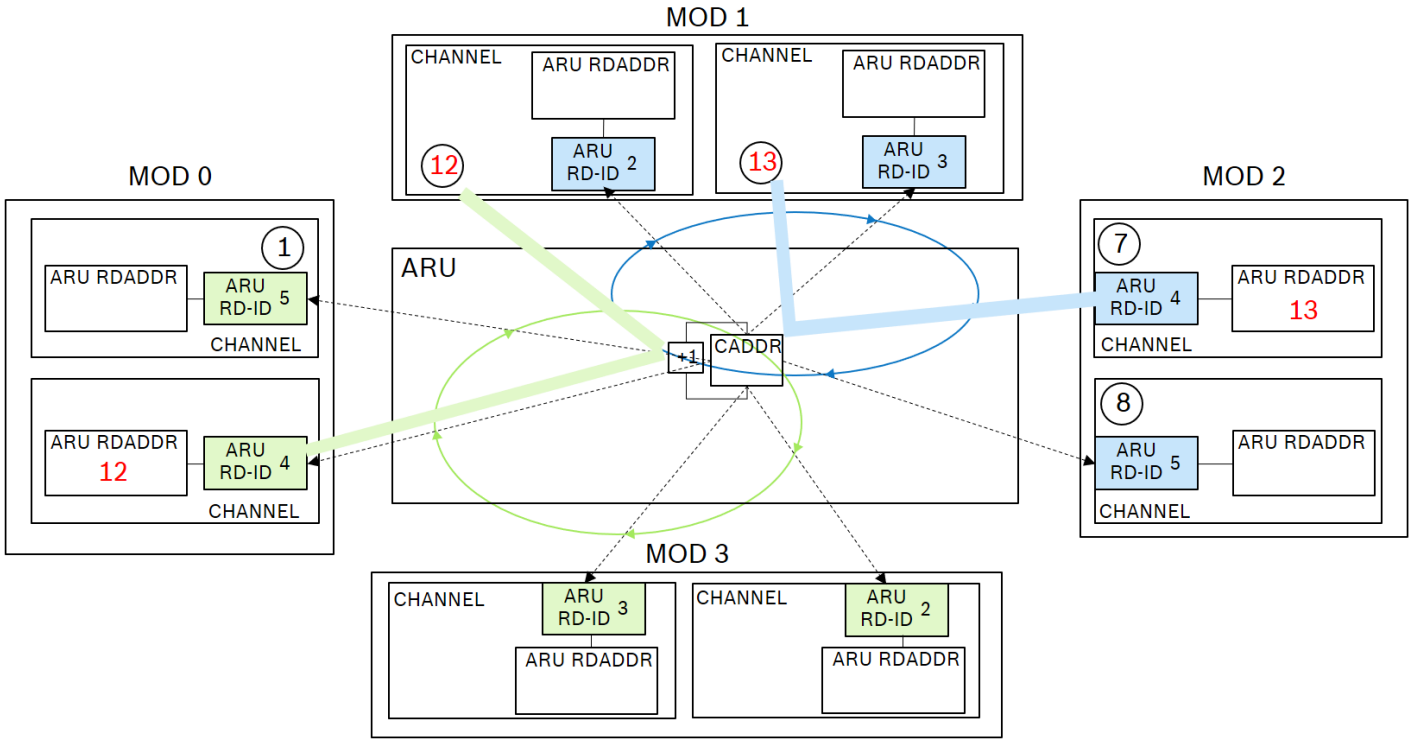

Figure 4.14: ARU data streams for two ARU ports addressing same submodule

In the Figure 4.14 the ARU Read channels of MOD2 and MOD0 both have the ARU RD-ID "4". Due to the fact, that the two address counters *aru_caddr* and *aru_caddr2* run in sync, both channels will request data from the two channels within MOD1.
It is possible, that the interface between the ARU Write channels and the ARU can become a bottleneck. This is shown with the following code example Code 9, where the ARU dynamic routing is enabled on both ARU ports and the addressing scheme is reloaded via ARU from two PSM channels.
The PSM0 channel 0 holds the dynamic routing ARU RD-IDs for ARU-0. The PSM0 channel 1 holds the ARU RD-IDs for ARU-1.

```
#define ARU_0_PSM_ID  0
#define ARU_1_PSM_ID  0

/* Configure FIFO channel 0 for ARU-0 */
GTM.PSM[ARU_0_PSM_ID].FIFO.CH[0].CTRL     = 0x1;         // ring buffer mode
GTM.PSM[ARU_0_PSM_ID].FIFO.CH[0].END_ADDR = 7;           // use 8 entries
GTM.PSM[ARU_0_PSM_ID].F2A.CH_STR_CFG[0]   = 0x00060000; // 53bits @ ARU
/* Fill the FIFO with the routing scheme */
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (20 | (21<<8)|(22<<16)); /*
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (23 | (24<<8)|(25<<16)    * 1.RD-IDs
                                           | (1<<24)|(1<<28));   */
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (26 | (27<<8)|(28<<16)); /*
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (29 | (30<<8)|(31<<16)    * 2.RD-IDs
                                           | (0<<24)|(1<<28));   */
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (32 | (33<<8)|(34<<16)); /*
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (35 | (36<<8)|(37<<16)    * 3.RD-IDs
                                           | (1<<24)|(1<<28));   */
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (38 | (39<<8)|(40<<16)); /*
GTM.PSM[ARU_0_PSM_ID].AFD.CH[0].BUF_ACC = (41 | (42<<8)|(43<<16)    * 3.RD-IDs
                                           | (0<<24)|(1<<28));
/* Configure FIFO channel 1 for ARU-1*/
GTM.PSM[ARU_1_PSM_ID].FIFO.CH[1].CTRL       = 0x1;         // ring buffer mode
GTM.PSM[ARU_1_PSM_ID].FIFO.CH[1].START_ADDR = 8;
GTM.PSM[ARU_1_PSM_ID].FIFO.CH[1].END_ADDR   = 15;          // use 8 entries
GTM.PSM[ARU_1_PSM_ID].F2A.CH_STR_CFG[1]     = 0x00060000; // 53bits @ ARU
/* Fill the FIFO with the routing scheme */
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (50 | (51<<8)|(52<<16)); /*
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (53 | (54<<8)|(55<<16)    * 1.RD-IDs
                                           | (1<<24)|(1<<28));   */
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (56 | (57<<8)|(58<<16)); /*
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (59 | (60<<8)|(61<<16)    * 2.RD-IDs
                                           | (0<<24)|(1<<28));   */
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (62 | (63<<8)|(64<<16)); /*
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (65 | (66<<8)|(67<<16)    * 3.RD-IDs
                                           | (1<<24)|(1<<28));   */
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (68 | (69<<8)|(70<<16)); /*
GTM.PSM[ARU_1_PSM_ID].AFD.CH[1].BUF_ACC = (71 | (72<<8)|(73<<16)    * 4.RD-IDs
                                           | (0<<24)|(1<<28));   */
/* Enable FIFO channels 0 and 1" */
GTM.PSM[ARU_0_PSM_ID].F2A.ENABLE = 0x2;
GTM.PSM[ARU_1_PSM_ID].F2A.ENABLE = 0x8;

/* Configure dynamic routing */
GTM.CTRL              = 0x0;
GTM.ARU.CADDR_END     = 10;
GTM.CTRL              = 0x1;
GTM.ARU.DYN_CTRL[0]   = 0x1;
GTM.ARU.DYN_RDADDR[0] = 0x051;     /* PSM0 channel 0 ARU Write address */
GTM.ARU.DYN_CTRL[1]   = 0x1;
if (ARU_1_PSM_ID==0) {
    GTM.ARU.DYN_RDADDR[1] = 0x052; /* PSM0 channel 1 ARU Write address */
} else {
    GTM.ARU.DYN_RDADDR[1] = 0x05A; /* PSM1 channel 1 ARU Write address */
}

/* Enable dynamic routing */
GTM.ARU.CTRL = 0xA;   // enable ARU dynamic routing on ARU-0 and ARU-0
```

Code 9: ARU Dynamic routing with ARU update via PSM

The code from Code 9 results in the waveform shown in Figure 4.15. There is a bottleneck at the PSM0 ARU port which leds to the fact, that for ARU-1, the first ARU RD-IDs are repeated while for ARU-0 the third ARU RD-IDs are repeated. After a total of 580ns, the routing scheme starts with the first ARU RD-ID addresses again.

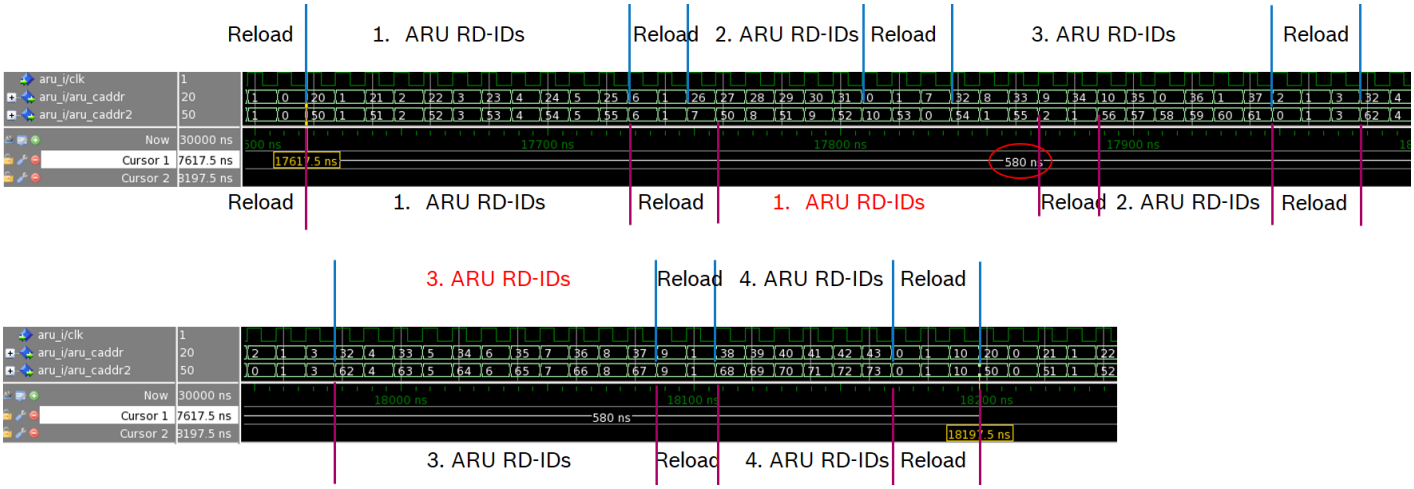

Figure 4.15: ARU data streams for two ARU ports, same PSM submodule

Figure 4.16 shows the situation, when two different PSM submodules are used. For this case, the define in Code 9 is changed to `#define ARU_1_PSM_ID 1`. Therefore, the ARU-0 gets its dynamic routing ARU RD-IDs from PSM0 channel 0 and ARU-1 gets the data from PSM1 channel 1. With this configuration, the repetition of some ARU RD-IDs within one PSM round trip can be avoided. This decreases the overall routing round trip time to 440ns.

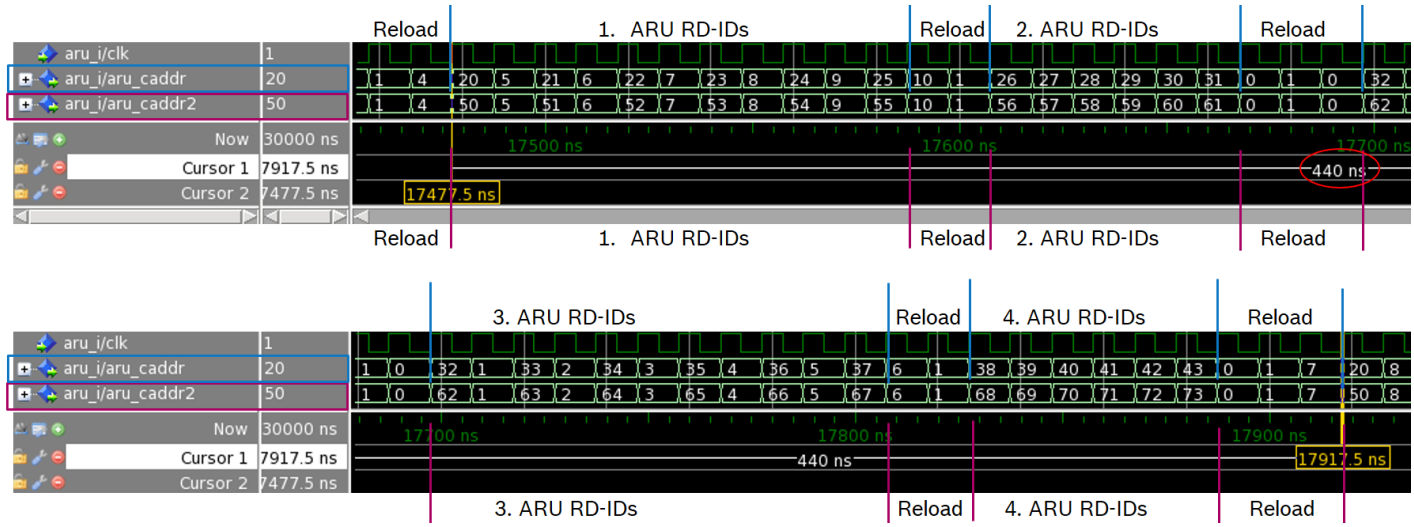

Figure 4.16: ARU data streams for two ARU ports, different PSM submodules

As it can be seen from the different examples of this application note, the mechanism of the ARU Dynamic routing mode can result in a significant decrease of data throughput times but can have some side effects. Therefore, this mechanism has to be used with care.

5 ARU accesses via CPU

5.1.1 Overview

The ARU offers register interfaces for the CPU to access and manipulate the data streams within the ARU. One interface can be used to manually inject data into the ARU or read data from the ARU. This interface is described in more detail in subsection 5.1.2. Another interface can be used for debugging purposes of ARU data streams. That interface provides the data to the CPU in addition to the configured ARU Read channel of a data stream and is described in subsection 5.1.3.

5.1.2 Default ARU access

The default ARU access can be used by the CPU to act as an additional ARU Read or Write channel. To establish this functionality, the default ARU RD-ID "0" is used by the ARU to serve CPU requests. Registers involved in the default ARU access mechanism are `GTM.ARU.ACCESS`, `GTM.ARU.DATA_H` and `GTM.ARU.DATA_L`. The register `GTM.ARU.ACCESS` is used to configure the ARU RD_ADDR or RD-ID and whether the CPU interface is an ARU Read or Write channel. The registers `GTM.ARU.DATA_H` and `GTM.ARU.DATA_L` receive data from the ARU in case the interface represents an ARU Read channel. Otherwise, the two registers can be used to provide data to a submodule connected to the ARU.
Code 10 shows an example, where the CPU interface is configured as an ARU Read channel. The data is provided via a PSM submodule and read out via the registers `GTM.ARU.DATA_H` and `GTM.ARU.DATA_L` afterwards by the CPU.

```
/*
 * Configure PSM0 CH0
 */
GTM.PSM[0].F2A.CH_STR_CFG[0] = 0x00060000; // provide 53 bits of data to ARU
GTM.PSM[0].F2A.ENABLE        = 0x2;
GTM.PSM[0].AFD.CH[0].BUF_ACC = 0xAAAAAAAA; // fill the FIFO with data
GTM.PSM[0].AFD.CH[0].BUF_ACC = 0xBBBBBBBB;
/*
 * Configure ARU_ACCESS to read from PSM0 CH0
 */
GTM.ARU.IRQ_EN = 0x4;              // enable ACC_ACK IRQ to signal new
                                   // available data at ARU
GTM.ARU.ACCESS = 0x00001051;       // trigger RREQ from address 0x051

while (!(GTM.ARU.IRQ_NOTIFY&0x4))  // poll ACC_ACK flag
    ;
GTM.ARU.IRQ_NOTIFY = 0x4;          // clear ACC_ACK flag
/*
 * TODO: read ARU data from registers GTM.ARU.DATA_H and GTM.ARU.DATA_L
 */
```

Code 10: Default ARU access via CPU, CPU acts as ARU Read channel

When new data is available in those two registers, the bit `GTM.ARU.IRQ_NOTIFY.ACC_ACK` bit is set. This can be seen in Figure 5.1. The figure also shows the contents of the two registers `GTM.ARU.DATA_H` and `GTM.ARU.DATA_L`. Please note, that despite the fact that the FIFO is filled with the two values `0xAAAA_AAAA` and `0xBBBB_BBBB`, the upper byte of the ARU low data word is skipped and replaced by the upper five bit of the ARU data word bits 52 to 48. This can be seen in the two registers `GTM.ARU.DATA_H` and `GTM.ARU.DATA_L`. The register bit field `GTM.ARU.ACCESS.ADDR` has to be configured with the ARU Write address of the PSM channel (in the example 0x51), the CPU wants to read from.

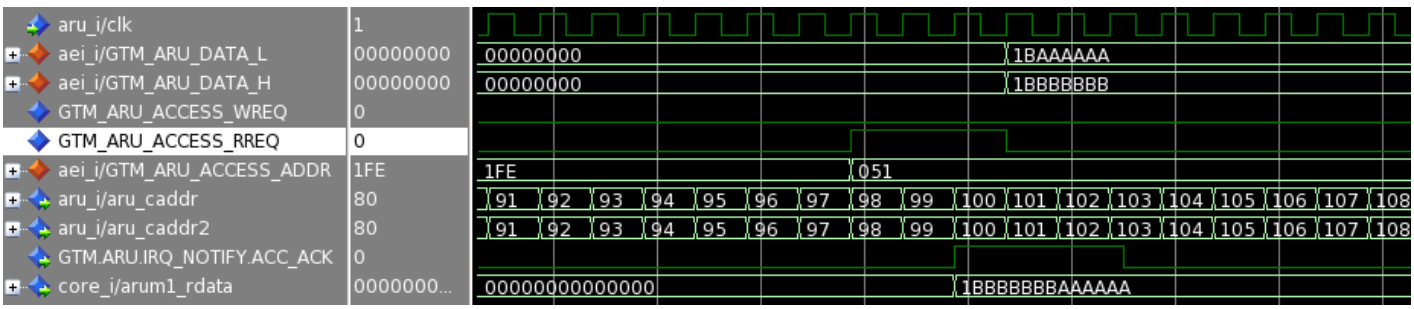

Figure 5.1: Default ARU access via CPU, CPU acts as ARU Read channel

Code 11 shows an example, where the CPU interface is configured as an ARU Write channel. The CPU writes a PWM characteristic to the ARU which is used by a connected ATOM channel to generate a PWM at its output. In this use case, the registers GTM.ARU.DATA_H and GTM.ARU.DATA_L hold the PWM characteristic. The ATOM channel uses the ARU RDADDR 0x0 as its ARU_RDADDR.

```
/*
 * Configure ATOM0 CH0
 */
GTM.ATOM[0].CH[0].CTRL      = 0x0000080A; // SOMP, ARU_EN, SL=1, CMU_CLK=0
GTM.ATOM[0].CH[0].RDADDR    = 0x0;        // default ARU Write address!
GTM.ATOM[0].AGC.GLB_CTRL    = 0x00020000; // Enable update of cmpregs (SR->CM)
GTM.ATOM[0].AGC.OUTEN_STAT  = 0x00000002; // Enable ATOM Output
GTM.ATOM[0].AGC.ENDIS_CTRL  = 0x00000002; // Enable ATOM Channel
GTM.ATOM[0].AGC.GLB_CTRL    = 0x00000001;
/*
 * Configure ARU_ACCESS to write PWM characteristic to ATOM
 */
GTM.ARU.DATA_H = 50;          // PWM duty cycle
GTM.ARU.DATA_L = 100;         // PWM period
GTM.ARU.ACCESS = 0x00002000;  // schedule ARU WREQ, GTM.ARU.ACCESS.ADDR=0!

while (!(GTM.ARU.IRQ_NOTIFY&0x4))  // wait until data is transfered
    ;
GTM.ARU.IRQ_NOTIFY = 0x4;          // clear ACC_ACK flag

/*
 * Transfer new characteristic to ATOM
 */
GTM.ARU.DATA_H = 20;          // PWM duty cycle
GTM.ARU.DATA_L = 100;         // PWM period
GTM.ARU.ACCESS = 0x00002000;  // schedule ARU WREQ, GTM.ARU.ACCESS.ADDR=0!

while (!(GTM.ARU.IRQ_NOTIFY&0x4))  // wait until data is transfered
    ;
GTM.ARU.IRQ_NOTIFY = 0x4;          // clear ACC_ACK flag
```

Code 11: Default ARU access via CPU, CPU acts as ARU Write channel

Figure 5.2 shows the behaviour of the ARU, when the CPU acts as a ARU Write channel. The data, which should be provided, is located in the registers GTM.ARU.DATA_L and GTM.ARU.DATA_H. In this example, the ARU blocking mechanism can be observed. The CPU issues the ARU write request (GTM_ARU_ACCESS_WREQ) immediately after the bit is set by the CPU with GTM.ARU.ACCESS = 0x00002000. The ATOM channel as the ARU Read channel is served by the ARU in a cyclic manner. Thus, the second write request from the CPU is blocked longer until the router schedules the ATOM channel to read from its source. The new PWM characteristic is visible at the atom_out after the end of the old PWM characteristic.

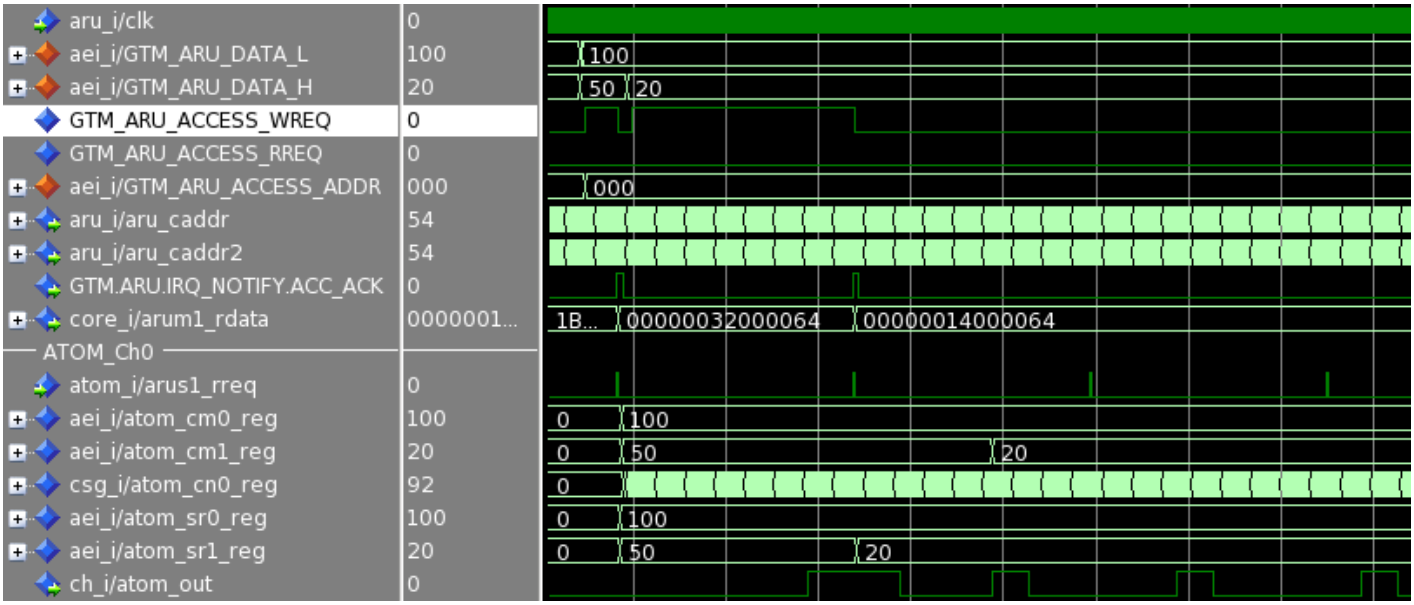

Figure 5.2: Default ARU access via CPU, CPU acts as ARU Write channel

5.1.3 ARU debug access

Subsection 5.1.2 showed, how data can be injected into the ARU or read out from the ARU. This interface is the same as for other submodules connected to the ARU. There is a second ARU access option, which can be used for debugging purposes. This feature has to be used with care, as it can interfere with software debuggers using the same interface to trace ARU data streams.
The registers `GTM.ARU.DBG_ACCESS0/1`, `GTM.ARU.DBG_DATA0/1_H` and `GTM.ARU.DBG_DATA0/1_L`, are involved when accessing ARU data streams. When used, these registers act as sniffers, and duplicate a data stream which is transferred through the ARU from a source to a destination. Code 12 shows an example, where an input signal is measured at a TIM input. When new measurement data at the TIM is available, it is routed through the ARU to the FIFO. The ARU interface is set up, to sniff this stream and provide the data also in the registers `GTM.ARU.DBG_DATA0_H` and `GTM.ARU.DBG_DATA0_L`.

```
/*
 * Configure ARU_DBG to read from TIM0 CH0
 */
GTM.ARU.IRQ_EN      = 0x1;         // enable ARU_NEW_DATA0_IRQ
GTM.ARU.DBG_ACCESS0 = 0x00000001; // read from TIM0 CH0
/*
 * Configure PSM0 CH0
 */
GTM.PSM[0].F2A.CH_STR_CFG[0]     = 0x00020000;  // read 53 bits from ARU
GTM.PSM[0].F2A.CH_ARU_RD_FIFO[0] = 0x1;         // read from TIM0 CH0
GTM.PSM[0].F2A.ENABLE            = 0x2;
/*
 * Configure TIM0 CH0
 */
GTM.TIM[0].CH[0].CTRL = 0x00000F21;    // TPWM, ARU_EN, GPRs=CNT, CNTS

for (idx_u8=0; idx_u8<5; idx_u8++) {
    while (!(GTM.ARU.IRQ_NOTIFY&0x1))  // wait until data is transfered
        ;
    GTM.ARU.IRQ_NOTIFY = 0x1;          // clear interrupt flag
    /*
     * Check if data is transferred to PSM and ARU debug interface
     */
    if ((GTM.ARU.DBG_DATA0_L != GTM.PSM[0].AFD.CH[0].BUF_ACC) ||
        (GTM.ARU.DBG_DATA0_H != GTM.PSM[0].AFD.CH[0].BUF_ACC)){
        // Error occurred!
    }
}
```

Code 12: Debug access to ARU via ARU CPU interface

6 References

[1] Robert Bosch GmbH, GTM-IP Specification, Revision 3.1.5.1, 24.03.2016

7 General Information about this document

7.1 LEGAL NOTICE

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.

INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS' FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.

GOVERNING LAW: THE RELATIONSHIP BETWEEN YOU AND ROBERT BOSCH GMBH SHALL BE GOVERNED SOLELY BY THE LAWS OF THE FEDERAL REPUBLIC OF GERMANY. THE STIPULATIONS OF INTERNATIONAL CONVENTIONS REGARDING THE INTERNATIONAL SALE OF GOODS SHALL NOT BE APPLICABLE. THE EXCLUSIVE LEGAL VENUE SHALL BE DUESSELDORF, GERMANY.

MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.

INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS: ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN, GERMANY AND ITS LICENSORS.

7.2 Revision History

Version	Date	Description
0.1	31.07.2018	Initial version
1.0	08.08.2018	Released
1.1	25.02.2020	Released Described ARU Round trip timing in more detail Added equations for calculation

Robert Bosch GmbH
AE/EIY4
Postfach 13 42
72703 Reutlingen
Germany

bosch.semiconductors@de.bosch.com