Implementation of Time-Triggered MPSoCs with support for tool-based system generation on FPGAs

Jan-Henrik Bruhn
University of Oldenburg, Germany
OFFIS - Institute for Information Technology, Oldenburg, Germany
Mission

- Implementation of a deterministic, time-triggered MPSoC
- Time-triggered inter-processor communication
- Support for automated generation of MPSoC-designs according to a predefined model
Time Triggered Architecture

- Split Hardware (Platform) into Tiles and VirtualChannels
- Split Software (Application) into Tasks and Messages
- Tasks get executed on a Tile at a predefined point in time
- Message get transmitted via VirtualChannels after execution of its sending task
- Loop
- Hyperperiod: least common multiple of all Task-Periods

Functional Test Environment for Time-Triggered Control Systems in Complex MPSoCs using GALI, Seyyedi, R. et al.
Time Triggered Communication

- On-Chip Communication
  - Network On Chip
    - Scales well for a very large number of processors
    - Complex to configure; Sacrifices a lot of FPGA-Resources
  - (TDMA) Bus
    - Comparably easy to implement; uses a small amount of resources
    - Does not scale well for a very large number of processors
  - Point-to-Point connections
    - Easy to implement; Little Resource-Use
    - Does not scale as well as a Bus; Might require a lot of connections
Implementation of Time-Triggered MPSoCs with support for tool-based system generation on FPGAs

Time-Triggered Communication

Tile 0

- PE 0
- Network Interface
- Task Timer

Tile 1

- PE 1
- Network Interface
- Task Timer

TDMA Bus
Implementation of Time-Triggered MPSoCs with support for tool-based system generation on FPGAs

Tile 0

ProcessingElement

Task 0
(Software)

C1.send(m0, 0)

C2.send(m1, 1)

Application (SW)

NetworkInterface (Hardware)

network_link + MAC

Task 0
(Software)

C1 send(m0, 0)

C2 send(m1, 1)

Physical (SW)

Presentation (SW)

Network (HW/SW)

Message Timer

send(uint32[] packets) {
  wait(myTdmaSlot)
  for (p in packets):
    phys.send(p)
}

send(uint32[] packets) {
  void* [] packet;
  getAddress(port) -> address;
  address -> packet[0];
  fragments.length -> packet[1];
  fragments -> packet[2..n];
  packets.push(packet); // FIFO Buffer
}

message_timer_interrupt() {
  link.send(packets);
}

send(void* [] fragments, int port) {
  void* [] packet;
  getAddress(port) -> address;
  address -> packet[0];
  fragments.length -> packet[1];
  fragments -> packet[2..n];
  packets.push(packet); // FIFO Buffer
}

send(type msg, int port) {
  uint32 fragments[M];
  msg -> fragments; // Aufteilung
  net.send(fragments, port)
}
\[ n_{\text{packetSize},x} = \sum_{m_y \in M_x} (\text{fragments}(m_y) + 2) \]

\[ n_{\text{maxPacket},i} = \max(n_{\text{packetSize},0}, \ldots, n_{\text{packetSize},n}) \]

\[ t_{\text{packetsTransfer},i} = n_{\text{maxPacket},i} \]

\[ t_{\text{timeSlot},i} = t_{\text{packetsTransfer},i} + t_{\text{sync}} \]

\[ t_{\text{wcDeliveryTime},i} = t_{\text{fifoSend}} + t_{\text{timeSlot},i} + t_{\text{fifoRecv}} \]

**Set of all fragments of all messages of Task x**

**Maximum amount of fragments to be sent by any Task mapped onto Tile i**

**Minimum length of a TDMA-timeslot, 1 clockcycle sync-time for processing**

**Worst Case delivery-time of any message of Tile i**

- Target address a
  - Length n
  - Fragment 0
  - Fragment ...
  - Fragment n
  - Target address a
  - Length n
  - Fragment 0
  - Fragment ...
  - Fragment n
All clocks are *fully synchronized*.

Messages are already in the receiving FIFO at task activation.
Implementation – Network Interface

- Vivado won’t synthesize the bus as a tri-state signal
- No tri-state signals available on the FPGA (Xilinx ZC702) anyways
- Solution: „Custom built“ Tri-State through integrated MUX
Implementation of Time-Triggered MPSoCs with support for tool-based system generation on FPGAs

NetworkInterface

AXI Bus

Receiving FIFO

Sending FIFO

AXI-Logic

MUX

TDMA_S0

TDMA_S1

...  ...

TDMA_S15

TDMA_in

TDMA Receive Logic

TDMA Send Logic

send_trigger

TDMA_out

TDMA_S0_en  ...  TDMA_S15_en
Simulation – Network Interface

- Testbench creates AXI-Commands to test two network interfaces connected together.

```verilog
wait for 10 ns;
wait until rising_edge(S_AXI_ACLK);

-- Write address to be 1
A_S_AXI_AADDR <= "000";
A_S_AXI_AVALID <= '1';
A_S_AXI_AWRITE <= 1'b0;
A_S_AXI_AMSTRB <= 1'b0;
A_S_AXI_AREADY <= '0';

-- Write message length to be 3
A_S_AXI_AADDR <= "001";
A_S_AXI_AVALID <= '1';
A_S_AXI_AWRITE <= 1'b0;
A_S_AXI_AMSTRB <= 1'b0;
A_S_AXI_AREADY <= '1';

wait for 10 ns;
wait until falling_edge(A_S_AXI_AREADY);

wait for 10 ns;
wait until falling_edge(A_S_AXI_AREADY);
```

Address Länge d₀ d₁ d₂
Evaluation

• Measurements on the system
  – Correct execution of software, message-transmission
  – Correct timing-behaviour (as calculated before)
  – Measurements via
    • Vivado Integrated Logic Analyzer [short periods]
    • Externer RedPitaya Logic Analyzer [long periods]
• Compare results with input schedules
Delivery of Messages via TDMA

Interrupt as message-trigger

Transmission via the bus
Evaluation - Results

- Task- und Message-Timer Interrupts
  - Schedules given as input get executed correctly
- ProgramCounter after Interrupt
  - Tasks start execution after ~3 ns ±2 ns (@100MHz)
- TDMA-Bus observation
  - Messages get delivered correctly and in precalculated time
Conclusion

- Time-triggered architecture successfully implemented on an MPSoC
- Implemented NetworkInterface accomplishes goals of deterministic timing and lean implementation
- Designed systems are automatically generateable through use of previously determined models